# FloatSD: A New Weight Representation and Associated Update Method for Efficient Convolutional Neural Network Training

Po-Chen Lin, Mu-Kai Sun, Chuking Kung, and Tzi-Dar Chiueh, *Fellow, IEEE*

***Abstract*— In this paper, we propose a floating-point signed digit (FloatSD) format for convolutional neural network (CNN) weight representation and its update method during CNN training. The number of non-zero digits in a weight can be as few as two during the forward and backward passes of the CNN training, reducing the convolution multiplication to addition of two shifted multiplicands (partial products). Furthermore, the mantissa field and the exponent field of neuron activations and gradients during training are also quantized, leading to floating-point numbers represented by eight bits. We tested the FloatSD method using three popular CNN applications, namely, MNIST, CIFAR-10, and ImageNet. These three CNNs were trained from scratch using the conventional 32-bit floating-point arithmetic and the FloatSD weight representation, 8-bit floating-point numbers for activations and gradients, and half-precision 16-bit floating-point accumulation. We obtained FloatSD accuracy results very close to or even better than those trained in 32-bit floating-point arithmetic. Finally, the proposed method not only significantly reduces the computational complexity for CNN training but also achieves memory capacity and bandwidth saving of about three quarters, demonstrating its effectiveness in the low-complexity implementation of CNN training.**

***Index Terms*— Convolutional neural network (CNN), weight quantization, low-complexity training, ImageNet, CIFAR-10, MNIST.**

## I. Introduction

In THE past few years, convolutional neural networks (CNNs) have drawn a great deal of attention from academia, industry and even the public media. Significant achievements have been made in many applications, such as image recognition, object detection, speech recognition, self-driving cars, etc. [1]–[4]. But CNN has a major disadvantage – a great number of arithmetic operations are required for CNN realization. As such, many studies have tried to propose low-complexity CNN processing schemes as well as CNN hardware architectures with higher performance and efficiency. In the industry and academia alike, highly efficient acceleration solutions for CNN training and inference based on GPU, FPGA, and dedicated ASIC are being developed in the hope of circumventing the major computational hurdle amid the current popular trends in artificial intelligence.

In a CNN, the process of feeding the input data through a pre-trained neural network model to obtain the CNN outcome is called *inference*. In order to obtain a neural network model that can solve the problem, we must train the network from a random initial configuration to one with a set of appropriate weights that generates outcomes with the lowest loss (error). Training a CNN entails passing the data in the training set through the CNN to obtain a loss, and then propagating the loss backward through the CNN to compute the corresponding weight adjustment. This procedure needs to be repeated for different training data in the training set and also for several times through the whole training set before the CNN can converge to a final configuration with satisfactory performance. Clearly, the computational cost of CNN training is much larger than that of the CNN inference operation. Nowadays, the majority of CNN training uses single-precision 32-bit floating-point (FP32) arithmetic. Due to the high cost in FP32 computational complexity and hardware power consumption, many researchers have begun to work on more efficient CNN training solutions.

Many CNN computation reduction methods first obtained the optimal CNN model using the FP32 arithmetic during training, and then reduced the network weights and/or neuron activations by different techniques, such as *quantization, pruning,* and *compression*. For instance, in [5] an FP32 trained neural network was quantized into a model with weights represented by a few bits. In [6], an FP32 trained neural network was simplified by singular value decomposition (SVD). In [7], the neural network model was pre-trained with the FP32 arithmetic and then the network weights were quantized and cropped to reduce the computational cost. In addition to using the quantization and cropping method of [7], the authors in [8] further applied Huffman coding to get a compressed network model. In [9], an FP32 trained neural network was

Manuscript received November 17, 2018; revised March 1, 2019; accepted March 29, 2019. Date of publication April 18, 2019; date of current version June 11, 2019. This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 106-2221-E-002-238. This paper was recommended by Guest Editor C.-Y. Chen. *(Corresponding author: Tzi-Dar Chiueh.)*

P.-C. Lin was with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan. He is now with Mediatek Inc. (e-mail: alexlin5411@gmail.com).

M.-K. Sun and C. Kung are with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan.

T.-D. Chiueh is with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: chiueh@ntu.edu.tw).

Digital Object Identifier 10.1109/JETCAS.2019.2911999

quantized by fine tuning the fixed-point word-length of the network weights to obtain a low-complexity model with results closer to that of the FP32-trained network. Another work [10] proposed a learnable quantization method to quantize the weights of CNNs obtained by training in FP32 arithmetic. It claimed that with down to 4-bit quantization of weights and activations, the quantized CNN exhibited no degradation in accuracy.

The aforementioned works all proposed solutions that started with a CNN model whose weights were trained from scratch using the FP32 arithmetic. Their focus was on obtaining a low-complexity CNN inference model, while maintaining the CNN accuracy performance. In [11], it was demonstrated that quantization of pre-trained CNN could be more easily achieved using theoretical guarantee that bounds the required precision for a target accuracy. However, note that even with the technique in [11], the computational complexity required for the initial CNN training in FP32 was not reduced.

In contrast with the above works, there are several other studies that attempted to simplify the CNN training process. For instance, BNN [12] quantized network weights and neuron activation outputs to one-bit signals by deterministic binarization. But it achieved only 67.1% top-5 accuracy on the ImageNet using the AlexNet model, which is 13.1% lower than the FP32 version of Alexnet. Another work, XNOR-Net [13], also adopted the deterministic binarization method to quantize the network weights and neuron activations. As a result, convolution of the weights and activations could be reduced to bit counting. Furthermore, the XNOR-Net extracted a common scaling term for weights/neurons in computing the same output channel of a layer, thus FP32 multiplications are still needed. With this modification, the ImageNet top-5 accuracy of XNOR-Net improved to 73.2%. QNN [14] adopted six-bit precision for weights, neuron activation, and backward gradient values in training the GoogLenet to achieve a top-5 accuracy of 83.1% on ImageNet. However, this accuracy is still 8.1% worse than the FP32-trained GoogLenet. The DoReFa network [15] quantized the weights, activations, and gradients to different number of bits. The 8-bit-precision version achieved only 53% accuracy using AlexNet for the ImageNet database, which was about 3% less than the FP32-trained AlexNet.

In [16], the Flexpoint format proposed that during training the weights or the neuron activations within a layer share an exponent field while the mantissa part of each weight or neuron activation is represented by a 16-bit integer. With 5-bit shared exponent and 16-bit mantissa for each weight/activation, the Flexpoint approach could achieve ImageNet accuracy performance similar to that of the FP32-trained CNN. In a similar way, the authors in [17] proposed to compute the convolution in half-precision arithmetic (FP16) while keeping a master copy of weights in single precision (FP32). In addition, they proposed a loss scaling technique to preserve gradient values with small magnitude for half-precision accumulation. This mixed-precision training (MPT) method could also achieve ImageNet accuracy close to that of the FP32-trained CNN.

BWN [13] simplified only the representation for weights by extracting a common scaling constant so that all weights associated with an output channel were represented by $+1$ or $-1$. This removed the multiplications in the convolution operation. However, extra multiplications were still needed to obtain the exact convolution layer outputs since the scaling factors had to be factored in. Moreover, in BWN the other signals such as neuron activations and gradients were still represented in the FP32 format. This scheme could achieve 83.0% and 86.1% top-5 ImageNet accuracies by ResNet-18 and GoogLenet, respectively. Note that these accuracies were still 6.2% and 3.9% worse than those of the FP32-trained networks, respectively. Similar to BWN, ternary weighted networks (TWN) [17] also applied reduction only to network weights and a truncated weight had three possible levels: $+1$, 0, or $-1$. The TWN scheme was shown to achieve 84.2% top-5 ImageNet accuracy using ResNet-18, which is about 2.5% worse than the FP32 case. As with BWN, the TWN scheme required that during CNN training all the neuron activations and gradient values be represented in the FP32 format.

In [19], the WAGE network tried 2-bit weights, 8-bit activations and gradient values and achieved top-1 AlexNet accuracy of 48.4%, which is worse than BWN, DoReFa-Net, and TWN. More recently, the FP8 network proposed in [20] by far outperformed all previous deep learning acceleration methods and achieved very little accuracy degradation in large-scale CNN applications, such as ResNet18 and ResNet50 for ImageNet. The proposed method adopted 8-bit floating-point numbers to represent weights, activations, and gradient values. In addition, the accumulation for the convolution was performed in 16-bit floating-point precision, rather than FP32 arithmetic.

In summary, most previous CNN complexity reduction proposals work quite well for inference, but not as well for CNN training from scratch. With the exception of the FP8 method in [20], quantizing network weights as well as neuron activations and gradients can significantly degrade the accuracy of the trained networks, especially for complicated CNNs. In order to reduce the CNN training complexity and establish an effective CNN training acceleration hardware architecture, this paper proposes the FloatSD number representation and an update method to reduce the number of significant bits in a weight and adopt even fewer bits to represent a weight in the convolutional multiplication of CNN training. Coupled with further quantification of the remaining parameters, including neuron activations and gradient values, the proposed method significantly reduces the required computational complexity and memory access bandwidth in CNN training. In the end, while simplifying network weights, neuron activations, and gradient values, the proposed FloatSD and the update method achieve CNN accuracy performances that are close to or even better than those of the FP32-trained CNN.

The rest of the paper is organized as follows. Section II introduces the mathematical formulas used in neural network training, including multi-layer perceptrons (MLP) and convolutional neural networks. Section III describes the FloatSD algorithm proposed in this paper and how it is applied to CNN training. Then in Section IV, to further reduce computational complexity and memory access bandwidth, quantification of

mantissa and exponent fields of other parameters used in the forward and backward passes of CNN training are studied. The FloatSD simulation results, including three representative CNN applications: MNIST, CIFAR-10, and ImageNet, are presented in Section V. In Section VI, we compare the proposed scheme with several previous works and discuss its advantages. Finally, conclusions are given in Section VII.

## II. Training and Inference in MLP and CNN

### A. Multi-Layer Perceptron (MLP)

A multi-layer percepton is made up of several fully-connected (FC) layers. The inference formula of neurons in the $l^{\text{th}}$ layer is given by

$$a_j^l = f(o_j^i) = f\left(\sum_{i=0}^{N_l-1} w_{ij}^l a_i^{l-1} + b_j^l\right), \tag{1}$$

where $a$ is the neuron activation in each layer; $f$ is the activation function; $o$ is the activation function input; $w$ is the network weight; $b$ is the bias; $l$ is the network layer index. The MLP training consists of the forward pass of network inference and the backward pass of error propagation. The error back propagation (EBP) formula is given by

$$\Delta w_{ij} \propto \frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_j}\frac{\partial a_j}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = a_i \delta_j, \tag{2}$$

where $C$ is usually called the *loss function* or cost function. When one uses the cross entropy as the loss function, the gradient value takes the form of

$$\delta_j = \frac{\partial C}{\partial a_j}\frac{\partial a_j}{\partial o_j} = \begin{cases} a_j - t_j & \text{output layer} \\ \sum_k \delta_k w_{jk} f'(o_j) & \text{otherwise} \end{cases} \tag{3}$$

Note that $t_j$ is the target value of the $j$th output neuron. From the above formulas, it is clear that the EBP procedure first calculates the gradient by weighted summing the gradients from the previous layer (in the backward order), and then multiplies the gradient with the activation from the input layer to get the weight update value.

### B. Convolution Neural Network (CNN)

A convolutional neural network usually consists of several convolutional layers, pooling layers, and a couple of FC layers. Forward inference of the convolutional layers can be expressed as

$$\begin{aligned} o_{i,j}^{l,d} &= \sum_{m=0}^{K-1}\sum_{n=0}^{K-1}\sum_{c=0}^{N-1} w_{m,n,c}^{l,d} a_{i+m,j+n,c}^{l-1} + b^{l,d} \\ &= \left(\sum_{c=0}^{N-1} w_{m,n,c}^{l,d} * a_{i,j,c}^{l-1}\right) + b^{l,d}, \end{aligned} \tag{4}$$

where $*$ is the convolution operator; the $d$th output channel in the $l$th layer has $N$ input channels; and these channels all have a $K \times K$ convolutional kernel. For notational simplicity, the following derivation assumes only one input channel and one output channel. As in MLP, the CNN training includes a backward pass, whose computation is also divided into two steps:

$$\Delta w_{m,n}^l \propto \frac{\partial C}{\partial w_{m,n}^l} = \delta_{i,j}^l * a_{m,n}^{l-1}, \tag{5}$$

and

$$\delta_{i,j}^l = \frac{\partial C}{\partial o_{i,j}^l} = \left(\delta_{i,j}^{l+1} * w_{K-1-m,K-1-n}^{l+1}\right) f'(o_{i,j}^l). \tag{6}$$

Note that the backward pass in CNN training contains convolution of the gradient values with neuron activations and another convolution of the weights with the gradient values in the previous layer (in the backward order).


Fig. 1. Format of the proposed SD representation. This example has eight groups, each with three digits.

TABLE I
Proposed Signed Digit Numbers in a 3-Bit Group

| | Index ($I$) | Value |
|---|---|---|
| Positive | 7 | 100 (+4) |
| | 6 | 010 (+2) |
| | 5 | 001 (+1) |
| Zero | 4 | 000 (0) |
| Negative | 3 | 00$\underline{1}$ (-1) |
| | 2 | 0$\underline{1}$0 (-2) |
| | 1 | $\underline{1}$00 (-4) |

## III. Proposed Number Representation and Update Mechanism for CNN Training

This section explains the proposed number representation that helps achieve low-complexity CNN training. The Canonical Signed Digit (CSD) representation was proposed to reduce the multiplication complexity because it represents the multiplier with fewer non-zero digits [21]. The CSD representation replaces any 0 followed by consecutive 1s in a binary representation by a $+1$ and a $-1$ at the position of least significant 1. For example, the decimal number 63 has the binary representation of 0111111, and its corresponding CSD representation is 1000001, where $\underline{1}$ represents $-1$. It is clear that representing 63 in CSD reduces the number of partial products needed in multiplying any number by 63 from six to two, saving significant computational complexity.

### A. Proposed Signed Digit (SD) Representation

The proposed signed digit (SD) representation differs from the CSD representation in that it divides the digits representing a number into several groups, each with a number of digits, as shown in Fig. 1. Also, to achieve low complexity, each

group allows no more than one non-zero digit. For instance in the three digit per group case, each group can represent seven different values: +4 (100), +2 (010), +1 (001), 0 (000), −1 (00$\underline{1}$), −2 (0$\underline{1}$0), −4 ($\underline{1}$00), as shown in Table 1.

As mentioned earlier, the complexity of multiplying two numbers depends on the number of non-zero digits in the multiplier. For a binary number, the probability of a bit being 0 is 50%, while the probability of a bit in a CSD number being 0 is approximately 66.6%. For the proposed SD representation with $K$ bits per group, there are $2K+1$ $K$-digit numbers and only $2K$ nonzero digits (see Table 1). Therefore, the probability of a digit in an SD number being 0 is

$$\frac{(2K+1)K-2K}{(2K+1)K}=\frac{2K-1}{2K+1}.$$

In the case of $K=3$, this probability is 71.4%.

It is noteworthy that three digits in the proposed SD representation can represent only seven instead of eight different values by three bits in the binary representation. As such, the proposed SD representation, albeit having low number of non-zero digits, cannot cover all possible binary numbers. However, neural networks are known to be tolerant to numerical inaccuracy and sometimes even benefit from such, e.g., deliberate noise injection [22], drop out [23], and other techniques. Therefore, in this work we adopted the proposed SD representation for neural network weights with the aim of reducing the required multiplication complexity in both forward and backward processes of the training.

### B. Update Mechanism for SD weights

The CNN training process requires that the weights be updated with the amount computed in the back propagation procedure (see Eqs. (2) and (5)). A straightforward update method for SD weights is to convert an SD weight into the binary format; perform addition; and then convert back to the SD representation. In this work, we proposed a low-complexity SD weight update method, which we call *Single Trigger Update (STU)*.

The STU mechanism considers the magnitude of the weight update amount and finds in which update group the amount lies, then generates a trigger to the corresponding SD weight group, as shown in Fig. 1. The trigger can be a *carry* or a *borrow* depending on the sign of the update amount. When receiving a carry/borrow, a group will move up/down one position in the SD number table listed in Table 1. When the group is at the topmost/bottommost position, it will reset to the other extreme value and send a carry/borrow to its more significant neighbor group.

The rule to determine the corresponding group for triggering is to find the leading 1 in the magnitude of the update amount, and check which update group range it falls in. Note that in Fig. 1 the update range is shifted by one digit with respect to the group boundary to approximate more closely the precise update of adding the update amount to the original weight. Finally, the STU mechanism is listed in Algorithm 1, where $I(k)$ denotes the index of the $k$th group value in Table 1;

**Algorithm 1**: Single Trigger Update for Neural Network Weights Using Signed Digit Representation

```
Input: I(k), s, k_init
Output: I(k)
Initialize: k ← k_init
while true do
    if k ≠ 1  //not at MSG
    then
        if (s = 1  and  I(k) = MAX) or (s = −1  and
          I(k) = MIN)  //Need carry/borrow
        then
            I(k) ← (I(k) = MAX) ? MIN : MAX ;
            k ← k − 1 ;
        else
            I(k) ← I(k) + s;
            break;
        end
    else
        //at MSG
        if (s = 1  and  I(1) = MAX) or (s = −1  and
          I(1) = MIN)  //Need carry/borrow
        then
            break ;
        else
            I(1) ← I(1) + s;
            break;
        end
    end
end
```

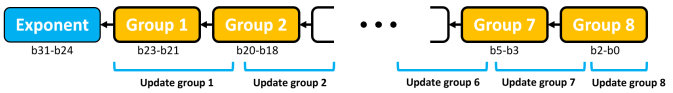



Fig. 2. Format of the proposed FloatSD representation. This example has an 8-bit exponent field and eight three-digit groups as mantissa.

$s=+1$ or $-1$ represents carry or borrow; $k$ is the index of the current group; (MAX, MIN) = (7, 1) according to Table 1.

Previously, a similar coefficient (weight) update rule for adaptive equalizer has been proposed in [24], which delivered very good equalization performance with only 25% hardware complexity of the traditional least-mean-square equalizer. However, unlike the adaptive equalizer in [24], the number of weight update operations in CNN training is significantly fewer than the number of convolution operations in the forward and backward passes, due to batch update and convolutional weight reuse. As such, acceleration of CNN training depends to a much higher degree on the complexity saving by SD weight multiplication than on the saving by STU in weight update.

### C. Floating-Point SD Representation

Training neural networks with fixed-point weights can only be successful in simple applications with small-sized networks. To successfully train a complex deep neural network, the training procedure must allow for weights and neuron activations with quite high dynamic range. Usually, this can involve careful analysis and profiling of the numerical values

**Algorithm 2**: Single Trigger Update for Neural Network Weights Using FloatSD Representation

**Input:** $I(k), s, k_{init}, exp$
**Output:** $I(k)$
**Initialize:** $k \leftarrow k_{init}$
**while** *true* **do**
  **if** $k \neq 1$ ***//not at MSG***
  **then**
    **if** $(s = 1$ ***and*** $I(k) = MAX)$ ***or*** $(s = -1$ ***and*** $I(k) = MIN)$ ***//Need carry/borrow***
    **then**
      $I(k) \leftarrow (I(k) = \text{MAX})$ ? MIN : MAX ;
      $k \leftarrow k - 1$ ;
    **else**
      $I(k) \leftarrow I(k) + s$;
      break;
    **end**
  **else**
    **//at MSG**
    **if** $(s = 1$ ***and*** $I(1) = MAX)$ ***or*** $(s = -1$ ***and*** $I(1) = MIN)$ ***//Need carry/borrow***
    **then**
      **if** $exp$ *is full* **then**
        break;
      **else**
        **forall** $k > 1$ ***//non-MSG group***
        **do**
          $I(k) \leftarrow 2I(k) > \text{MAX}+1$ ? $I(k)-1$ : (
          $2I(k) = \text{MAX}+1$ ? $I(k)$ : $I(k)+1$ ) ;
          **//shift** $I(k)$ **toward (MAX+1)/2**
        **end**
        $exp \leftarrow exp + 1$;
        break ;
      **end**
    **else**
      $I(1) \leftarrow I(1) + s$;
      break;
    **end**
  **end**
**end**

during the whole training procedure and setting the decimal point position for all variables [9]. Since we plan to apply the proposed weight representation to the training of as many neural networks as possible, we proposed to add an exponent field to increase the dynamic range of weight values. The new version with the exponent field is called the floating-point signed digit (FloatSD) representation and is depicted in Fig. 2.

With the extra exponent field and a floating decimal point in the FloatSD representation, the STU algorithm for FloatSD weights was modified to handle mantissa overflow. Similar to SD, the update procedure for FloatSD weights first finds the proper group to send a trigger by taking into account the exponent field of the current weight and that of the update amount. The trigger will propagate through the groups toward the most significant group (MSG). In the case of the MSG already at the upper or lower limit (MAX or MIN) while receiving a carry or borrow trigger, the decimal point will be moved to the left by one digit to allow for the carry/borrow operation. This is achieved by moving all mantissa groups by one position toward 0 (index 4) according to Table 1 and increasing the exponent field (exp) by 1. Finally, the STU mechanism for FloatSD weights is presented in Algorithm 2.

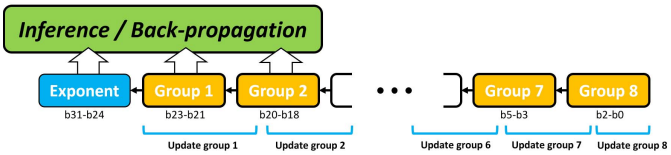



Fig. 3. Use only two groups in the FloatSD mantissa for inference and backward propagation computation.

### *D. Low-Precision Weights in Multiplication*

In [12], [13], the authors proposed to adopt high-precision 32-bit floating-point representation for weights and their update values in the stochastic gradient descent (SGD) training for deep CNN. To keep the CNN inference complexity low, the low-precision version of the weights (with only 1 bit) were used in the computation of the forward and backward propagation procedure of the CNN training. In this work, we adopted the same concept and used only a few mantissa groups of the FloatSD weights for CNN inference and weight update computation, as illustrated in Fig. 3.

## IV. Other Low-Complexity Considerations

The FloatSD representation introduced in the previous section is used for the network weights. However, CNN training and inference computation involves more than just network weights. Other quantities such as neuron activations and gradient values also have to be computed and accessed. By quantizing these quantities, in addition to FloatSD representation for weights, neural network training can benefit from not only low hardware complexity, but also low memory access bandwidth. Taking the second layer of ResNet-18 with a batch size of 64 as an example, there are $3 \times 3 \times 128 \times 128 = 147,456$ weights, while the number of input image pixels is $28 \times 28 \times 128 \times 64 = 6,422,528$, which is 43 times more than the number of weights. In light of this fact, in this work we also considered reducing the number of mantissa bits and that of exponent bits in representing the neuron activations and gradient values.

### *Forward Pass (Inference)*

The forward pass of CNN training is identical to its inference operation and the forward pass formula was given previously in Eq. (4). The network weights $w^l_{m,n}$ are represented in the FloatSD format with two most significant mantissa groups. Further simplification can be achieved by quantizing the mantissa part of the floating-point number representation for the neuron activation, $o^l_{i,j}$. Since rounding quantization has been shown to be more efficient than truncation, as described in [25], in this work we adopted rounding in mantissa quantization.

*Backward Pass*

In the backward pass, the formulas for computing the weight update values were given in Eqs. (5) and (6). In addition to representing the weights in FloatSD with only two mantissa groups, neuron activation $a_{m,n}^{l-1}$ and gradient values $\delta_{i,j}^{l}$ can be simplified to reduce hardware complexity of the backward pass computation.

*Exponent Range Reduction*

Profiling the distribution of network activations and gradients of neural networks during training revealed that most weight magnitudes are between $2^{-25}$ and $2^{5}$. With a proper offset in the exponent field, its width can be reduced from 8 bits to 5 bits. This modification can improve both the computation power consumption as well as the memory access bandwidth. To reduce the exponent field of the FloatSD weights, we applied the per-layer scaling constant technique mentioned in [26] and extracted a power-of-two constant for each layer in the CNNs. Then with this per-layer scaling constant technique, we found that the exponent field of the FloatSD weight could be reduced to 3 bits.

## V. Simulation Results

This section presents the simulation results of applying the proposed low-complexity CNN training scheme to three image recognition benchmark problems. We used *Caffe*, a deep learning framework described in [27], as the platform to implement the proposed algorithm in this study. Note that all the images in this study were not pre-processed. Only the built-in mirroring and random cropping were adopted to enhance the training datasets. All reported accuracies are based on one crop.

**MNIST** [28]

MNIST is a set of grayscale handwritten numeral images. The image size is $28 \times 28$; the output classes are from 0 to 9; the training set contains 60K images; and the test set size is 10K. For MNIST simulation, we adopted the LeNet network architecture [29] with "20-CONV5 + MP2 + 50-CONV5 + MP2 + 500FC + 10FC." The above shorthand means that the neural network consists of a $5 \times 5$ convolutional layer with 20 output channels, a $2 \times 2$ maximum pooling layer, followed by another $5 \times 5$ convolutional layer with 50 output channels, a $2 \times 2$ maximum pooling layer, a fully connected layer of 500 output neurons, and finally another fully connected layer of 10 output neurons (for 10 numerals). The training procedure adopted a momentum of 0.9, a weight decay of 0.0005, an initial learning rate of 0.01, and the "inv" learning rate adjustment method in Caffe with a gamma of 0.0001 and a power of 0.75.

**CIFAR-10** [30]

The second image recognition dataset based on CNN is CIFAR-10. CIFAR-10 is a color image recognition dataset, where the image size is $32 \times 32$ and there are 10 output classes. In this application, we used a training set with 50K images and a test set with 10K images. For CIFAR-10 application, we adopted the VGG-7 network architecture in [18] with " $2 \times$ (128-CONV3) + MP2 + $2 \times$ (256-CONV3) + MP2 + $2 \times$ (512-CONV3) + MP2 + 1024FC + dropout + 10FC." The momentum is 0.9; the weight decay is 0.004; the initial learning rate is 0.001; the learning rate adjustment is the step method in Caffe, and the learning rate is divided by 10 at $30,000$ and $40,000$ iterations.

**ImageNet** [31]

Unlike some previous CNN training acceleration studies that only experimented with the relatively smaller MNIST and CIFAR-10 datasets, we used ImageNet to verify the effectiveness of the proposed FloatSD scheme. ImageNet is a color image recognition dataset with $256 \times 256$ images, 1000 output categories, 1280K training set size, and 50K test set size. To perform recognition on the ImageNet, we adopted the ResNet-18 network architecture with residual learning proposed in [32]. The momentum, the weight decay, and the starting learning rate are set to 0.9, 0.0001, and 0.1, respectively. The learning rate adjustment method is the step method in Caffe, and the learning rate is divided by 10 at $140,000$, $180,000$ and $230,000$ iterations. In addition to ResNet-18, we also applied the FloatSD scheme to a deeper ResNet-50 to demonstrate higher than 90% top-5 accuracy.

In the following subsections, we report extensive simulation results that determined several key parameters crucial for hardware complexity and memory bandwidth. They include

- the group size in a FloatSD mantissa group,
- the number of more significant FloatSD groups and exponent range used in forward and backward passes,
- the mantissa word-length and exponent range of neuron activations in the forward pass,
- the mantissa word-length and exponent range of the gradients, and
- the mantissa word-length and exponent range of neuron activations in the backward pass.

### A. Group Size in FloatSD Mantissa

The group size, the number of digits in a group ($N_b$), of FloatSD mantissa greatly affects its precision and coverage. Larger group size results in less precision, while smaller group size sacrifices hardware efficiency. We simulated two group sizes: $N_b = 3, 4$ ($N_g$ is the number of groups used) for three applications. From Fig. 4, it is clear that in the MNIST and CIFAR-10 applications, the FloatSD weight scheme with $N_b = 3$ and $N_b = 4$ both deliver almost the same or even better accuracy than the 32-bit float-pointing (FP32) weight case. However, in the more complex ImageNet application using ResNet-18, the $N_b = 3$ and the $N_b = 4$ both fall behind the FP32 case by about 0.1% and 0.8% in top-5 accuracy, respectively. Since setting $N_b = 2$ will defeat the purpose of low multiplication complexity from using the FloatSD representation, we set $N_b = 3$ in the following simulations.

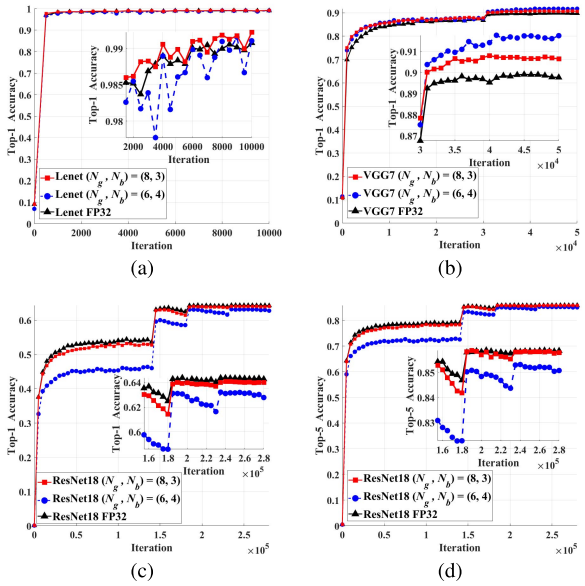



Fig. 4. Simulated accuracy results using different group sizes. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-1 accuracy), and (d) ResNet-18 on ImageNet (top-5 accuracy).

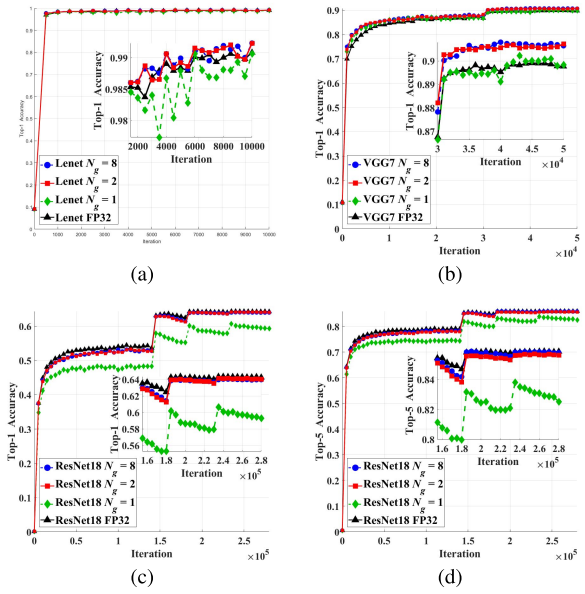



Fig. 5. Simulated accuracy results using different numbers of mantissa groups. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-1 accuracy), and (d) ResNet-18 on ImageNet (top-5 accuracy).

## *B. Number of Groups Used in Forward/Backward Passes*

As mentioned earlier, network weights are updated with high precision and they are used with low precision in multiplication. This simulation tried using different numbers of more significant groups in the FloatSD mantissa ($N_g$) for forward and backward computation. Figure 5 illustrates the simulated accuracy results for $N_g = 1, 2, 8$ cases and the (FP32) case. Note that the performance of $N_g = 2$ case is very similar to the case that used all eight groups ($N_g = 8$) in forward/backward computation. In all applications, FloatSD with $N_b = 3$, $N_g =$ 2 can deliver similar and sometimes higher accuracy than the FP32 case. With this setting, the multiplication using FloatSD weight as the multiplier requires only two partial products, while the FP32 multiplier needs about $24/2 = 12$ partial products. Moreover, reducing $N_g$ from 8 to 2 also significantly decreases the bandwidth and storage capacity for moving and storing the CNN weights.

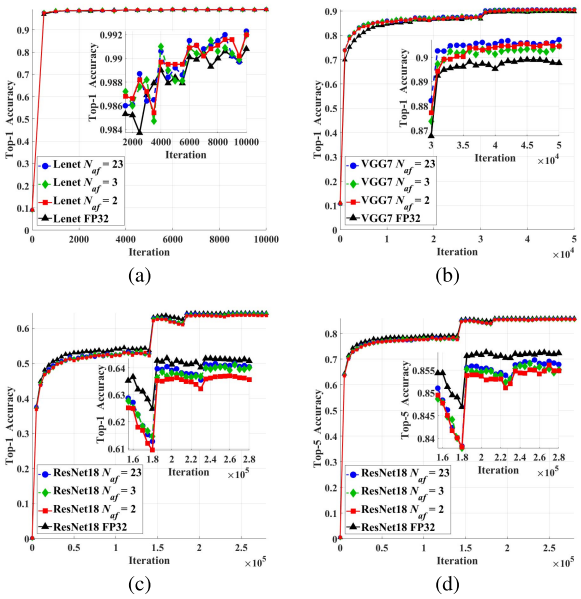



Fig. 6. Simulated accuracy results using different mantissa word-lengths for neuron activations in the forward pass. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-1 accuracy), and (d) ResNet-18 on ImageNet (top-5 accuracy).

## *C. Neuron Activation Quantization in Forward Pass*

To further reduce the computational complexity in CNN training, we investigated quantizing (by rounding) other variables in CNN training. This simulation looked into quantizing the mantissa of the neuron activations in the forward pass. Fig. 6 depicts the simulation results of quantizing the neuron activation's mantissa word-length from 23 to 3 and 2 bits ($N_{af} = 3, 2$). Note that in our simulation, the "leading-1" implicit bit in the FP32 format is still applied and the sign bit is not counted in the mantissa bits. It is clear from Figs. 6(a) and 6(b) that the mantissa of the activation in the forward calculation of the MNIST and CIFAR-10 CNN training can be rounded to 2 bits without observable degradation in accuracy. The bandwidth and storage saving from this quantization alone is about 65.6%, from 32 bits to 11 bits. Similarly, from Figs. 6(c) and 6(d), one can see that $N_{af} = 2$ is sufficient for ResNet-18 neural network training.

## *D. Backward Gradient Quantization*

To accelerate the backward pass of CNN training, we further explored quantizing the gradients in Eq. (6). The word-length of the mantissa for the gradients was rounded to $N_d$. In Fig. 7, all FloatSD simulations are based on

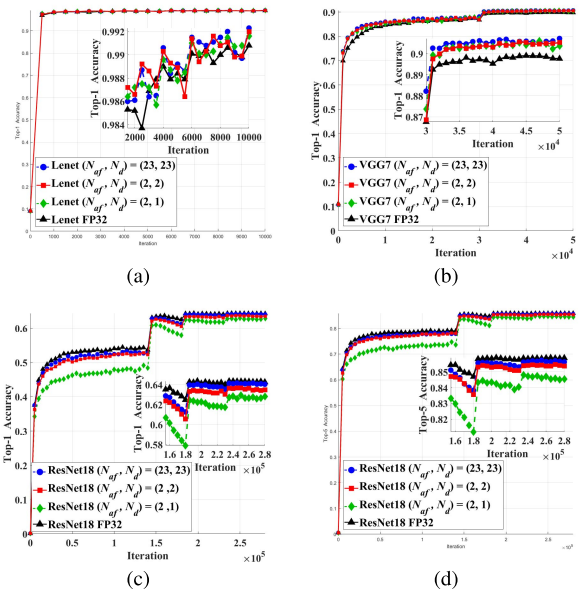



Fig. 7. Simulated accuracy results using different mantissa word-lengths for gradients in the backward pass. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-1 accuracy), and (d) ResNet-18 on ImageNet (top-5 accuracy).

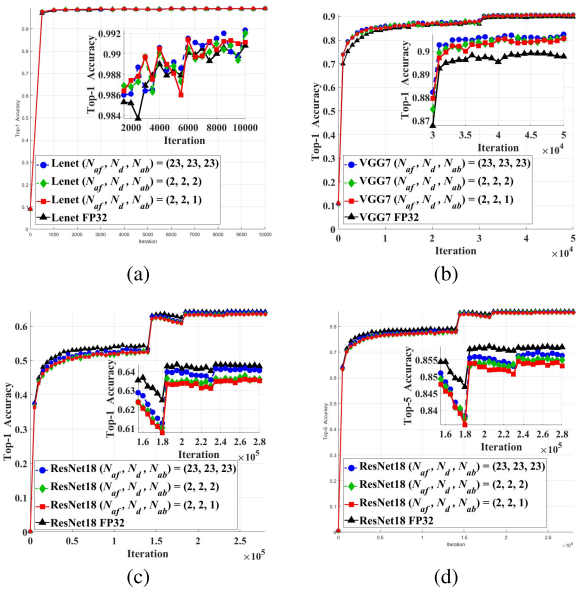



Fig. 8. Simulated accuracy results using different mantissa word-lengths for neuron activations in the backward pass. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-1 accuracy), and (d) ResNet-18 on ImageNet (top-5 accuracy).

$N_b = 3$, $N_g = 2$, $N_{af} = 2$. Figs. 7(a) and 7(b) show the accuracy of the MNIST and CIFAR-10 networks with $N_d = 23, 2, 1$, respectively. Simulated accuracy results indicate that $N_d = 2$ is sufficient for negligible accuracy loss in these two applications. Figs. 7(c) and 7(d) show the ImageNet ResNet-18 top-1 and top-5 accuracies based on FloatSD with $N_b = 3$, $N_g = 2$, $N_{af} = 2$ and reducing $N_d$ to 2 and 1. It is also clear that $N_d = 2$ is sufficient for ResNet-18 training.

### *E. Neuron Activation Quantization in Backward Pass*

The neuron activation values are convolved with the weights in the forward pass while they are convolved with the gradients in the backward pass. Therefore, they may react differently to quantization. In this simulation, we studied quantizing the mantissa of the neuron activations in the backward pass. The mantissa word-length for the neuron activations in the backward pass is defined as $N_{ab}$. Figs. 8(a) and 8(b) depict the simulated accuracy results of the MNIST and CIFAR-10 networks with $N_{ab} = 23, 2, 1$, respectively. It is clear that for these two applications, $N_{ab} = 1$ is sufficient. Similar observation can be made for ResNet-18 from Figs. 8(c) and 8(d).

### *F. FloatSD Exponent Range Reduction*

In the foregoing, the mantissa bits of all related signals in CNN training have been quantized and reduced. We believe that the exponent fields of those signals can also be further reduced. To this end, we further investigated the possibility of decreasing the number of exponent bits for reduction in memory access power and computational complexity. By analyzing the distribution of MNIST neuron activations during training, it was found that most of the values are distributed between $2^{-25}$ and $2^5$. Therefore, we reduced the exponents in all signals (except FloatSD weights) used in CNN training to 5 bits ($N_{ea} = 5$) with an offset for a range of $2^{-27}$ to $2^4$. With the 5-bit exponent field, all non-weight variables (e.g. neuron activations and gradients) in CNN training can be represented in one byte (8 bits). Furthermore, similar analysis was conducted for weight distribution during CNN training, and it was decided that the number of bits in the exponent of neural network weights can be reduced from 8 bits to 3 bits ($N_{ew} = 3$).

### *G. Putting Them All Together*

From the above simulation, we determined the key parameters of the proposed low-complexity scheme for the training of three popular CNN applications. Concerning the FloatSD weights, $(N_b, N_g) = (3, 2)$. For the mantissa and exponent fields of neuron activations and gradients, we set $(N_{af}, N_d, N_{ab}, N_{ew}, N_{ea}) = (2, 2, 1, 3, 5)$. Note that this version of FloatSD representation requires 9 bits to store one weight and we call it the FloatSD9 representation. In order to fit more weights into 32-/64-bit memory access bandwidth, we shrank the second mantissa group in FloatSD9 from three digits to two digits. This new representation has 7 possible values in group 1 (MSG) and 5 possible values in group 2. However, out of the 35 combinations, only 31 distinct combinations exist. We can use 5 bits to encode these two values and represent one neural network weight in 8 bits and call this representation *FloatSD8*. In addition to using the FloatSD8 representation for weights during CNN training, we also adopted larger batch sizes (up to 512), new training schedules, the loss scaling technique [17], and the IEEE half-precision floating-point accumulation (FP16) scheme, a variant of which was proposed in [20].

Fig. 9 illustrates the simulated accuracy performance of the three CNN applications using the proposed FloatSD8 method and the conventional FP32 training method. It is clear that in the two smaller applications (MNIST and CIFAR-10) the FloatSD8 method even outperforms the conventional

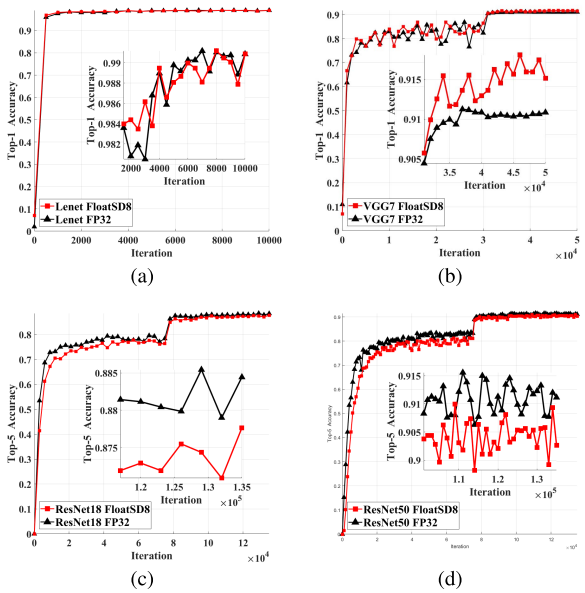



Fig. 9. Simulated accuracy results using the proposed FloatSD8 with quantized activation/gradient method and the conventional FP32 method. (a) MNIST, (b) CIFAR-10, (c) ResNet-18 on ImageNet (top-5 accuracy), and (d) ResNet-50 on ImageNet (top-5 accuracy).

TABLE II

ACCURACY RESULTS OF CNNS TRAINED BY THE FLOATSD8 METHOD AND THE CONVENTIONAL FP32 METHOD. TOP-5 ACCURACIES ARE LISTED FOR RESNET-18 AND RES-NET50

| Accuracy | MNIST | CIFAR | ResNet-18 | ResNet-50 |
|---|---|---|---|---|
| FP32 | 99.12% | 91.12% | 88.56% | 91.55% |
| FloatSD8 | 99.12% | 91.81% | 87.76% | 90.99% |

FP32 method in the accuracy of the final trained networks. This shows that the FloatSD8 training method not only achieves low power consumption but also can deliver higher accuracy. Similar results with both low complexity and high accuracy in network network training of MNIST and CIFAR were also presented in [12]. According to the study in [33], adding noise to the gradient update during training can reduce over-fitting and help smooth the training process of neural networks. In this work, the single trigger update mechanism in the proposed method is likely to be the reason of the observed higher accuracy.

For the ImageNet application, the top-5 accuracy of the ResNet-18 network trained by the FloatSD8 method is about 0.8% lower than that of same network trained by the FP32 method. To meet the 90% accuracy target, we conducted another simulation using the deeper ResNet-50 for the ImageNet application. In this case, the top-5 accuracies of the networks trained by the FloatSD8 method and the conventional FP32 method are 90.99% and 91.55%, respectively.

Finally, in Table 2 we summarize the accuracy performance of the proposed FloatSD8 method and compare its performance with the FP32 method. Note that the accuracy values listed are the maximum achieved accuracy values during the whole training process.

TABLE III

REQUIRED ARITHMETIC OPERATIONS IN THREE STEPS OF CNN TRAINING

| | Forward w*a | Backward gradient*w | Backward gradient*a |
|---|---|---|---|
| BNN[12] | 1-bit ADD | 32-bit ADD | 32-bit ADD |
| XNOR[13] | 1-bit ADD | 32-bit ADD | 32-bit ADD |
| QNN[14] | 6-bit MAC | 6-bit MAC | 6-bit MAC |
| DoReFa[15] | 8-bit MAC | 8-bit MAC | 8-bit MAC |
| Flexpoint[16] | 16-bit MAC | 16-bit MAC | 16-bit MAC |
| MPT[17] | 16-bit MAC | 16-bit MAC | 16-bit MAC |
| BWN[13] | 32-bit ADD | 32-bit ADD | 32-bit MAC |
| TWN[18] | 32-bit ADD | 32-bit ADD | 32-bit MAC |
| WAGE[19] | 8-bit ADD | 8-bit ADD | 8-bit MAC |
| FP8[20] | 16-bit ADD | 16-bit ADD | 16-bit ADD |
| **FloatSD8** | **16-bit ADD** | **16-bit ADD** | **16-bit ADD** |

## VI. COMPARISON AND DISCUSSION

There have been many works that simplify neural network implementation for CNN inference. However, these works often cannot perform satisfactorily in CNN training because they are not able to obtain CNNs with good accuracy. In the following, we select only those previous works that can effectively simplify CNN training for comparison.

### *A. Computational Complexity*

In Table 3, we compare the arithmetic operations during the three convolution steps of CNN training required by the proposed FloatSD8 method and other previous works that use simplified parameter representations and achieve good accuracy performance.

BNN quantizes 32-bit floating-point weights and neuron activations into 1-bit values during the forward pass, thus only 1-bit operations are required. In backward propagation, gradients are not quantized and 32-bit floating-point addition operations are still required. XNOR-Net also adopts 1-bit representation for weights and neuron activations. Yet, not all operations are 1-bit operations in XNOR-Net. There are extra 32-bit scaling factors, entailing extra floating-point multiplications to compute the final convolution outputs. QNN applies the quantization and uses 6-bit weights and 6-bit activations in the forward pass, so 6-bit MAC operations are required in the forward pass. In the backward pass, it also uses 6-bit gradients and thus 6-bit MAC operations are required in both steps. DoReFa-Net can achieve good enough ImageNet accuracy performance when weights, activation, and gradients are all represented in 8-bit integers. As such, all multiplications in the training process require 8-bit MAC.

Flexpoint introduces a shared exponent field for all weights in a layer. Each weight is still represented by a 16-bit mantissa,

thus 16-bit MAC operations are required in both the forward and backward passes. On the other hand, MPT reduces all convolution from FP32 operations to FP16 operations. Hence 16-bit floating MAC are required in this method. BWN represents a real-value weight using a binary value and one scalar that is shared among all weights associated with the same output channel in a layer. Since the neuron activations and gradients are not simplified, only 32-bit additions are required to compute to convolution involving the binary weights. However, 32-bit floating-point MAC operations are still required to compute the convolution between the activations and the gradients. TWN differs from BWN in only an extra zero value for the weights, making them having three possible levels. As such, TWN requires the same computational complexity in three CNN training steps as BWN.

WAGE constrained weights, activations, gradients, and errors during training to low word-length integers. Specifically, for ImageNet it uses 2-bit weights and 8-bit integers for all other variables. Therefore, all multiplications involving weights can be implemented in shift-and-add of 8-bit integers. However, 8-bit MAC operations are necessary for the convolution of the activations and the gradients. Finally, FP8 pushed the mantissa quantization to only 2-bits in their proposed 8-bit floating-point number representation. Since all accumulations after multiplication were performed in 16-bit floating-point operation, all convolutions can be achieved using 16-bit floating-point additions.

FloatSD8 represents a network weight with no more than two non-zero digits ($+1$ or $-1$) and an exponent term. For this reason, multiplication involving a FloatSD8 weight can be implemented by addition of two partial products (shifted and possibly negated multiplicand). As the neuron activations are quantized to 2-bit mantissa plus 5-bit exponent, only 16-bit additions are required in the forward pass. In the first step of the backward pass, 16-bit additions suffice because the gradients are also quantized to 2-bit mantissa plus 5-bit exponent. In the second step of the backward pass when the weight update amount is to be computed, the neuron activations are now quantized to 1-bit mantissa and 5-bit exponent. Therefore, this step can also be realized using 16-bit additions.

To sum up, both the FP8 method [20] and the proposed FloatSD8 method eliminates all multiplications and requires lower complexity than all methods except BNN and XNOR-Net. However, BNN and XNOR-Net cannot generate CNN with good enough accuracy when training large-size CNN for complicated applications, e.g., ResNet for ImageNet.

### *B. Memory Requirement*

Memory access speed has been know to be the major bottleneck for almost all CNN acceleration circuits. Moreover, power consumption of the memory subsystem in CNN hardware solutions almost always makes up a significant portion of the total power. In the proposed FloatSD8 method for CNN training, not only the network weights, but also the neuron activations and the gradient values used in the forward and backward passes have all been optimized for least memory usage. In Table 4, we list the required number of bits for representing weights, neuron activations, and gradients in the forward and backward passes of CNN training by all the aforementioned methods.

TABLE IV
NUMBER OF BITS TO REPRESENT WEIGHT, ACTIVATIONS, AND GRADIENTS DURING CNN TRAINING

| **No. of Bits** | $w$ | $n$(F) | $g$ | $n$(B) |
|---|---|---|---|---|
| BNN[12] | 1 | 1 | 32 | 1 |
| XNOR[13] | 1 | 1 | 32 | 1 |
| QNN[14] | 6 | 6 | 6 | 6 |
| DoReFa[15] | 8 | 8 | 8 | 8 |
| Flexpoint[16] | 16 | 16 | 16 | 16 |
| MPT[17] | 16 | 16 | 16 | 16 |
| BWN[13] | 1 | 32 | 32 | 32 |
| TWN[18] | 2 | 32 | 32 | 32 |
| WAGE[19] | 2 | 8 | 8 | 8 |
| FP8[20] | 8 | 8 | 8 | 8 |
| **FloatSD8** | **8** | **8** | **8** | **7** |

$w$: weights, $n$(F): neuron activation in forward pass
$n$(B): neuron activation in backward pass, $g$: gradients.

In BNN and XNOR-Net, weights and neuron activations are all quantized to one bit, while the gradients in the backward pass remain to be represented as 32-bit floating-point numbers. QNN and DoReFa-Net quantize all parameters to 6 bits and 8 bits, respectively. Flexpoint uses 16-bit integers to represent individual weight, activation, gradient value while there is an extra exponent field per layer, which is shared by all parameters in that layer. MPT uses 16-bit floating-point numbers to represent all variables. BWN quantizes the weights to binary, while the neuron activations and the gradients are 32-bit floating-point numbers. TWN and WAGE both use 2 bits to represent each weight, while TWN uses 32-bit floating-point numbers for activations and gradients and WAGE uses 8-bit integers. Finally, the FP8 method adopts 8-bit floating-point numbers for weights, activations, and gradients.

The FloatSD8 representation that we proposed employs two signed digit groups and a 3-bit exponent term, thus on the total each weight requires 8 bits (5 bits for two groups of signed digits). Moreover, two bits are sufficient to represent the activation mantissa in the forward pass and one bit is enough for the activation mantissa in the backward pass. Adding another bit for sign and 5 bits for exponent, the neuron activations require 8 bits and 7 bits, respectively, in the forward and backward passes. Similar to the activations in the forward pass, the gradient values in the backward pass can be represented in 8 bits.

Note that in larger CNNs, network activations and gradients constitute a major portion of variables in the training process. As a result, FloatSD8 representation actually requires less memory capacity in CNN training than Flexpoint, MPT, BWN, and TWN. The memory bandwidth requirement of DoReFa-Net, FP8, and WAGE is similar to that of FloatSD8. On the other hand, for BNN and XNOR-Net,

TABLE V
ACCURACY PERFORMANCES IN THREE CNN APPLICATIONS BY VARIOUS LOW-COMPLEXITY TRAINING METHODS

| Accuracy (%) | MNIST | CIFAR-10 | ImageNet[a] |
|---|---|---|---|
| BNN[12] | 98.60(-0.1) | 89.85(+0.79) | 67.1(-13.1)[b] |
| XNOR[13] | - | - | 69.2(-11.0)[b] |
| QNN[14] | - | - | 83.1(-8.1)[c] |
| DoReFa[15] | - | - | 55.9(-2.9)[d] |
| Flexpoint[16] | - | 95(+0.1)[e] | 80(0)[e] |
| MPT[17] | - | - | 76.04(+0.12)[f] |
| BWN[13] | - | - | 86.1(-3.9)[c]<br>83.0(-6.2)[g] |
| TWN[18] | 99.35(-0.06) | 92.56(-0.32) | 84.2(-2.5)[g]<br>86.2(-1.8)[h] |
| WAGE[19] | - | - | 75.86(-4.85)[b] |
| FP8[20] | - | 81.85(-0.35)[j]<br>92.21(-0.56)[k] | 66.95(-0.48)[m]<br>71.72(-0.42)[f] |
| **FloatSD8** | **99.12(+0)** | **91.81(+0.69)** | **87.76(-0.80)[g]**<br>**90.99(-0.56)[i]** |

[a]Top-5 accuracy of ImageNet, [b]AlexNet, [c]GoogLeNet
[d]Top-1 accuracy of AlexNet
[e]estimation (no exact values given)
[f]Top-1 accuracy of ResNet-50
[g]ResNet-18, [h]ResNet-18B (modified version)
[i]ResNet-50, [j]CNN, [k]ResNet
[m]Top-1 accuracy of ResNet-18

TABLE VI
POWER AND AREA COMPARISON BETWEEN CONVOLUTION ENGINES BASED ON FLOATSD8 AND FP32

| Process | Type | Period | Area | Power |
|---|---|---|---|---|
| 28nm CMOS | FP32 | 2.5ns | 48,518$\mu m^2$ | 14.22mW |
| 28nm CMOS | **FloatSD8** | 2.5ns | **6,231$\mu\mathbf{m^2}$** | **1.05mW** |

FloatSD8 may or may not excel in the required memory access bandwidth as this depends on the ratio of weights to gradients, which in turn is related to the CNN architecture.

### *C. Accuracy Performance*

We have compared the computational complexity and memory requirement by several low-complexity CNN training methods. However, achieved accuracy by the CNN training process is by far the most important criterion for comparison. Table 5 lists the simulated accuracies in three CNN applications by all the aforementioned methods. Since these works used different CNN training software packages and different image pre-processing, thus Table 5 also lists how much each work improves/degrades in accuracies when compared to the FP32 version according to its own implementation. The percentages within the parentheses denote these differences, with positive percentages representing accuracy improvement over the FP32-trained network and negative percentages denoting degradation. From Table 5, it is clear that the ImageNet accuracy degradation by BNN, XNOR-Net, and QNN is quite high (8%–13%). The significant loss in accuracy more than offsets their merits in lower complexity and lower memory bandwidth and capacity. This is particularly true as CNN training is usually performed off-line in computational servers, making trained network accuracy the top figure of merit. On the other hand, Flexpoint achieves accuracy with no visible degradation from that obtained by the FP32-trained CNN. However, note that the ImageNet top-5 accuracy is only 80%, meaning that the simulated CNN architecture is not as good as ResNet-18 and GoogLeNet used in the BWN, TWN, and FloatSD8 works. DoReFa-Net and WAGE both use all-integer variables for weight, activations, and gradients and thus their accuracy performance is not good, with about 3%-5% degradation.

BWN and TWN both extract a floating-point scalar from the weights associated with the same output channel. This reduces the number of bits to represent each weight to 1 and 2, respectively for BWN and TWN. The other variables in the training procedure are still 32-bit floating-point numbers. As a result, their backward propagation complexity is quite high. Since most of the training calculation is floating-point based, the ImageNet accuracy degradation can be as low as 1.8% for TWN. MPT adopts 16-point floating-point numbers in all variables, therefore its accuracy can be even better than the FP32 version. FP8 uses a new 8-bit floating-point numbers with 1 sign bit, 2 mantissa bits, and 5 exponent bits for all variables and thus achieves lower than 0.5% accuracy degradation.

Finally, the FloatSD8 method performs better than or as well as the FP32 method in two small CNN applications. This can be attributed to the noise injected during FloatSD8 training. For the ImageNet application, the FloatSD8 method obtains top-5 accuracies of 87.8% and 91.0% using the ResNet-18 and ResNet-50 networks, respectively. The ImageNet accuracy degradation by FloatSD8 is the smallest among all methods except Flexpoint, MPT, and FP8.

### *D. Circuit Design*

Based on the FloatSD8 representation and half-precision (FP16) accumulation arithmetic, we designed a 3x3 kernel convolution engine. This circuit takes 9 FloatSD8 weights and 9 input image pixels, each with one sign bit, two mantissa bits, and five exponent bits. Five-stage pipelining was implemented, with the following functional units. Partial product generation unit takes two nonzero digits from each FloatSD8 weight and generates 18 partial products. Partial product alignment unit finds the most significant partial product and aligns all other 17 partial products using shifters. A Wallace-tree carry save adder then adds all these shifted partial products with proper sign extension. Finally, a normalization unit properly normalizes the mantissa and exponent fields of the final convolution result to the half-precision (FP16) format.

In addition, we also designed a 3x3 kernel convolution engine based on single-precision (FP32) multipliers and adders. With proper pipelining, we also made the FP32 convolution engine run at the same speed as the convolution engine

based on FloatSD8. These two convolution engine circuits were synthesized using a 28nm CMOS process. Table 6 lists the area and power consumption of the two synthesized circuits. When running at 400MHz, the FloatSD8 convolution engine is about 7.79X smaller in circuit area and consumes 13.54X less power than the FP32-arithmetic convolution engine running at the same clock rate.

To sum up, the proposed FloatSD8 representation and the associated single-trigger update method for CNN training have several advantages over the previous works:

- Low loss in accuracy of the trained CNN.
- Lower arithmetic complexity, memory bandwidth, and memory capacity than BWN, TWN, and Flexpoint.
- No need to pre-train an FP32 CNN. FloatSD8 can train a CNN from scratch with low complexity and memory requirement.
- Compatible with hyper-parameters usually used in current CNN training packages, such as network architecture, learning schedule, etc.
- Floating-point representation amenable to other neural network models than CNN. Higher weight dynamic range enables wider application.

## VII. Conclusions and Future Work

In this paper, we proposed the floating-point signed digit (FloatSD) representation for weights of the convolutional neural networks. In addition, a corresponding weight update method, called single trigger update (STU) was introduced. To further optimize the training efficiency and lower the computational and memory costs, we truncated the mantissa and the exponent fields of the other variables in CNN training, while keeping the loss in CNN accuracy negligible. Simulation results of the FloatSD8 CNN training indicated that the proposed method can achieve almost the same and even better accuracies in three different CNN applications. The computation complexity and memory bandwidth/capacity requirement of the proposed method were shown to be one of the best among all the previous works that can deliver good enough trained accuracy in complicated CNNs. Moreover, we designed two convolution engine circuits, one based on FloatSD8 and the other based on FP32 arithmetic. The proposed FloatSD8 circuit outperforms the FP32 circuit in area and power by about 7.8X and 13.5X, respectively. Finally, we believe that with its high dynamic range and signed-digit mantissa, the FloatSD8 representation is suitable for the training of other neural network models and it will provide efficient and effective solution to many future deep learning tasks.

In the future, we would like to extend this work to recurrent neural network models and generative models. Different variable precision on a per-layer basis for CNN training by applying the results in [34] is also under consideration. Furthermore, system-on-chip and FPGA implementation of the FloatSD8-based deep learning acceleration hardware would more convincingly confirm the low complexity and high energy efficiency advantages of the proposed method.

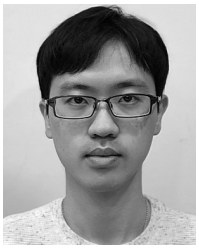
## Acknowledgement

The authors wish to thank the Ministry of Science and Technology, Taiwan for financial support under Grant no. MOST 106-2221-E-002-238.

## References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[3] M. Bojarski *et al.* (2016). "End to end learning for self-driving cars." [Online]. Available: https://arxiv.org/abs/1604.07316
[4] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[5] D. D. Lin, S. Talathi, and V. S. Annapureddy. (2016). "Fixed point quantization of deep convolutional networks." [Online]. Available: https://arxiv.org/abs/1511.06393
[6] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. (2014). "Exploiting linear structure within convolutional networks for efficient evaluation." [Online]. Available: https://arxiv.org/abs/1404.0736
[7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Dec. 2015, pp. 1135–1143.
[8] S. Han, H. Mao, and W. J. Dally. (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." [Online]. Available: https://arxiv.org/abs/1510.00149
[9] D. D. Lin and S. S. Talathi. (2016). "Overcoming challenges in fixed point training of deep convolutional networks." [Online]. Available: https://arxiv.org/abs/1607.02241
[10] S. Jung *et al.* (2018). "Learning to quantize deep networks by optimizing quantization intervals with task loss." [Online]. Available: https://arxiv.org/abs/1808.05779
[11] C. Sakr, Y. Kim, and N. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, 2017, pp. 3007–3016.
[12] M. Courbariaux, Y. Bengio, and J.-P. David. (2015). "BinaryConnect: Training deep neural networks with binary weights during propagations." [Online]. Available: https://arxiv.org/abs/1511.00363v3
[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (2016). "XNOR-Net: ImageNet classification using binary convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1603.05279v4
[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. (2016). "Quantized neural networks: Training neural networks with low precision weights and activation." [Online]. Available: https://arxiv.org/abs/1609.07061v1
[15] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. (2016). "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients." [Online]. Available: https://arxiv.org/abs/1606.06160
[16] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Dec. 2017, pp. 1742–1752.
[17] P. Micikevicius *et al.* (2018). "Mixed precision training." [Online]. Available: https://arxiv.org/abs/1710.03740
[18] F. Li, B. Zhang, and B. Liu. (2016). "Ternary weight networks." [Online]. Available: https://arxiv.org/abs/1605.04711v2
[19] S. Wu, G. Li, F. Chen, and L. Shi. (2018). "Training and inference with integers in deep neural networks." [Online]. Available: https://arxiv.org/abs/1802.04680
[20] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2018, pp. 7685–7694.
[21] R. M. Hewlitt and E. S. Swartzlantler, "Canonical signed digit representation for FIR digital filters," in *Proc. IEEE Workshop Signal Process. Syst.*, Lafayette, LA, USA, Oct. 2000, pp. 416–426.
[22] Y. Grandvalet, S. Canu, and S. Boucheron, "Noise injection: Theoretical prospects," *Neural Comput.*, vol. 9, no. 5, pp. 1093–1108, 1997.
[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] K.-H. Chen, C.-N. Chen, and T.-D. Chiueh, "Grouped signed power-of-two algorithms for low-complexity adaptive equalization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 12, pp. 816–820, Dec. 2005.
[25] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. (2015). "Deep learning with limited numerical precision." [Online]. Available: https://arxiv.org/abs/1502.02551
[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
[27] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[28] *MNIST*. Accessed: Jan. 15, 2019. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[30] *CIFAR-10*. Accessed: Jan. 15, 2019. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[31] (2016). *ImageNet*. [Online]. Available: http://image-net.org
[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[33] A. Neelakantan *et al.* (2015). "Adding gradient noise improves learning for very deep networks." [Online]. Available: https://arxiv.org/abs/1511.06807
[34] C. Sakr and N. Shanbhag, "An analytical method to determine minimum per-layer precision of deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 1090–1094.



**Po-Chen Lin** was born in Hsinchu, Taiwan, in 1993. He received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 2015, the M.S. degree in electronics engineering from National Taiwan University, Taipei, Taiwan, in 2017. Since 2018, he has been an Engineer with MediaTek Inc. His research interests lie in the areas of deep neural network algorithms and digital circuit design.

**Mu-Kai Sun** received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 2017. He is currently pursuing the M.S. degree in electronics engineering with National Taiwan University, Taipei, Taiwan. His research interests include deep neural network algorithms and digital circuit design.

**Chuking Kung** received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, where he is currently pursuing the M.S. degree in electronics engineering. His current research topic is about deep learning accelerator.

**Tzi-Dar Chiueh** (M'90–SM'03–F'13) was born in Taipei, Taiwan, in 1960. He received the B.S.E.E. degree from National Taiwan University, Taipei, in 1983, and the M.S. and Ph.D. degrees in electrical engineering from the California Institute of Technology, Pasadena, CA, USA, in 1986 and 1989, respectively.

Since 1989, he has been with the Department of Electrical Engineering, National Taiwan University, where he is currently a Distinguished Professor. From 2004 to 2007, he was the Director of the Graduate Institute of Electronics Engineering, National Taiwan University. He has held visiting positions at ETH Zurich, Switzerland, from 2000 to 2001, and the State University of New York, Stony Brook, from 2003 to 2004. From 2010 to 2014, he was the Director General of the National Chip Implementation Center, Hsinchu, Taiwan. He was the Vice President of the National Applied Research Laboratories, from 2015 to 2017. His research interests include IC design for digital communication systems, neural networks, and signal processing for bio-medical systems.

Prof. Chiueh is a fellow of the IEEE. His teaching efforts were recognized nine times by the Teaching Excellence Award from NTU. He was a recipient of the Outstanding Research Award from the National Science Council, Taiwan, in 2004 and 2007. He received the Outstanding Electrical Engineering Professor from the Chinese Institute of Electrical Engineers, Taiwan, in 2005, and the Himax Chair Professorship from NTU in 2006. He received the Outstanding Industry Contribution Award from the Ministry of Economic Affairs, Taiwan, in 2009. He received the Outstanding Technology Transfer Contribution Award from the Ministry of Science and Technology, Taiwan, in 2016.