Evaluations on Deep Neural Networks Training Using Posit Number System

Jinming Lu, Chao Fang, Mingyang Xu, Jun Lin, *Senior Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

Abstract—The training of Deep Neural Networks (DNNs) brings enormous memory requirements and computational complexity, which makes it a challenge to train DNN models on resource-constrained devices. Training DNNs with reduced-precision data representation is crucial to mitigate this problem. In this article, we conduct a thorough investigation on training DNNs with low-bit posit numbers, a Type-III universal number (Unum). Through a comprehensive analysis of quantization with various data formats, it is demonstrated that the posit format shows great potential to be employed in the training of DNNs. Moreover, a DNN training framework using 8-bit posit is proposed with a novel tensor-wise scaling scheme. The experiments show the same performance as the state-of-the-art (SOTA) across multiple datasets (MNIST, CIFAR-10, ImageNet, and Penn Treebank) and model architectures (LeNet-5, AlexNet, ResNet, MobileNet-V2, and LSTM). We further design an energy-efficient hardware prototype for our framework. Compared to the standard floating-point counterpart, our design achieves a reduction of 68, 51, and 75 percent in terms of area, power, and memory capacity, respectively.

Index Terms—Deep neural networks, training, quantization, posit number system, hardware accelerator

1 Introduction

Deep neural networks (DNNs) have made a great advance in various artificial intelligence applications, including computer vision, natural language processing, automatic speech recognition, and self-driving automobile. With many novel architectures and innovative algorithms emerging, DNNs continuously refresh state-of-the-art performance records in many tasks, but what follows is the significant increase of computational complexity and memory space requirements. In most cases, DNNs are implemented on high-end GPUs especially for training, which leads to high energy consumption. Therefore, it has been a serious challenge to efficiently deploy DNN models on resource-constraint devices.

Recently, there has been increasingly more attention paid to DNN model compression and acceleration technologies, aiming to generate compact and efficient DNN models [1], [2], [3]. Typical approaches contain model pruning, quantization, knowledge distillation, and low-rank approximation. Nevertheless, most works are mainly concentrated on the inference phase, since it is much easier than training phase. Compared to inference, DNN training additionally involves gradients backpropagation and parameters update. Consequently, it is more sensitive to some compression methods.

Some previous works have been reported to train DNN models with limited-precision numbers [4], [5], [6], [7]. However, part of them resulted in nonnegligible accuracy loss because of the aggressive quantization, and part of them incurred complex auxiliary tricks or floating-point arithmetic. In our opinion, a data format with higher representation ability is urgently needed, which should not only have enough dynamic range for big values but also have high precision for most of the values. Posit exactly meets this requirement.

Posit is a Type-III universal number (Unum) [8], parameterized by n (the word size) and es (the exponent field size). Being different from standard floating-point numbers (float), posit has a regime field encoded with the run-length method. Hence, posit shows a better trade-off between dynamic range and numerical precision. Some related researches have described the potential of posit in DNNs, but practical implementations and detailed evaluations are absent [8], [9].

In this work, we propose a DNN training framework using reduced-precision posit numbers and evaluate it on multiple typical deep learning applications. To take the best advantages of posit, an energy-efficient systolic array architecture supporting posit arithmetic is also proposed. The major contributions of this work are as follows:

- We make comprehensive comparisons among posit, float, and fixed-point. From multiple perspectives, including decimal accuracy, mean related error, and mean absolute error, the outstanding representation ability of posit is well justified.
- We propose a DNN training framework using low-precision posit. In this framework, a tensor-wise scaling factor, derived from data distribution, is introduced to make full use of the code space of posit. Besides, we also adopt a warmup training strategy to determine data distribution and guarantee convergence.

- *The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210008, China. E-mail: {jmlu, fantasysee, 171180570}@smail.nju.edu.cn, , {jlin, zfwang}@nju.edu.cn.*

Manuscript received 14 Oct. 2019; revised 23 Mar. 2020; accepted 1 Apr. 2020. Date of publication 14 Apr. 2020; date of current version 14 Jan. 2021. (Corresponding authors: Jun Lin and Zhongfeng Wang.) Recommended for acceptance by A. Louri. Digital Object Identifier no. 10.1109/TC.2020.2985971

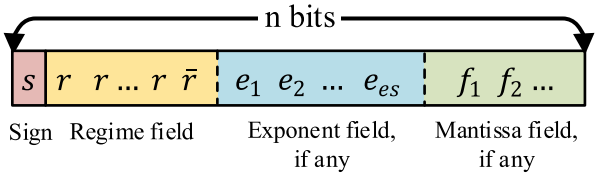


Fig. 1. The basic structure of an (n, es) posit number.

- We evaluate our framework using 8-bit posit on both convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Compared to the 32-bit floating-point (FP32) baseline, our framework shows less than 0.5 percent accuracy degradation on AlexNet, ResNet18, and MobileNet-V2 with ImageNet. For the word-level language modeling on Peen Treebank dataset, the perplexity loss is less than 0.1 on a 3-layer LSTM model.
- We further design an energy-efficient systolic array architecture with posit arithmetic to support our training framework. An optimized posit encoder and decoder are proposed. The implementation results show that an energy reduction of 51 percent and an area reduction of 68 percent are achieved compared with its FP32 counterpart. Meanwhile, a reduction of 75 percent reduction can be achieved in terms of bandwidth and memory requirements.

The organization of the rest of this paper is as follows: Section 2 gives an overview of DNNs and posit number system, and introduces some related works. Section 3 presents the quantization analysis and details of our DNN training framework. Section 4 reports the experiment results and some discussions. Section 5 introduces the hardware prototype. Section 6 draws the conclusion.

2 Preliminaries

2.1 Deep Neural Network

Deep neural networks play a significant role in artificial intelligence and have been widely used in many real applications because of their excellent performances. As a type of representation learning, DNNs learn complex features of data with layer-by-layer nonlinear transformation, where each layer is comprised of many neurons to perform matrix-vector multiplication. Usually, a nonlinear activation function $a^{(i)}(\cdot)$ is needed after each layer to achieve nonlinear transformation, where i denotes layer index. The common activation functions include the sigmoid $\sigma(\cdot)$ and the rectified linear unit (ReLU). Taking the notation $W^{(i)}$ as the weight parameters of layer i, the computation of an L-layer DNN model can be formatted as

$$f(x) = a^{(L)}\Big(\ldots a^{(2)}\Big(W^{(2)}a^{(1)}\Big(W^{(1)}x\Big)\Big)\Big). \tag{1}$$

Based on the difference of the connectivity between adjacent layers, there are several typical DNN architectures, such as fully connected networks (FCNs), CNNs, and RNNs. CNNs are outstanding for capturing spatial information with parameters sharing and local connectivity. RNNs are usually applied for sequential tasks such as natural language processing and speech recognition.

TABLE 1
The Mapping Between Regime Binary Codes and Regime Values

Binary code	0001	001X	01XX	10XX	110X	1110
Value	-3	-2	-1	0	1	2

In the computation of DNNs, training and inference are two key phases. In the training phase, the raw data are fed into the DNN model, and the loss function is computed by comparing the predictions with the associate real labels. All parameter gradients with respect to the loss function, derived from chain rule, are back-propagated layer-by-layer. Then the model parameters are updated by an optimization algorithm, which is usually stochastic gradient descent (SGD) or one of its variations such as Adagrad, Adadelta, Adam, and *etc*. The above process will be executed iteratively on the training dataset to improve model performance until convergence. In the inference phase, the trained model is used to predict targets for new inputs. It generally happens in the model test phase or real application scenes. By contrast, DNN training is more complicated than inference because of the existence of the backpropagation and weight update process.

2.2 Posit Number System

The universal number system was introduced by Gustafson *et al.* [10] for replacing IEEE standard 754 floating-point numbers (float). Several different formats have been presented in the Unum framework. The Type-I Unum can express interval arithmetic efficiently and is compatible with the float. However, extra costs are needed to handle its variable length. The Type-II Unum has fixed size and good mathematical properties, but the requirement of a huge look-up table limits its scalability.

To overcome the above shortcomings of these two kinds of Unums, *posit number system*, namely the Type-III Unum, was proposed[8]. In fact, posit is of float-like format, which is able to share most arithmetic operation units with the float. Compared to float, posit has a better trade-off between the dynamic range and numerical precision. Besides, posit has simpler hardware implementation and more powerful expression ability, making it very advantageous for deep learning applications.

A posit number is defined by two parameters, including word size n and exponent field size es. The es is used to control the trade-off between dynamic range and numerical precision. As shown in Fig. 1, an (n, es) posit number consists of four parts, including *sign* bit, *regime* field, *exponent* field, and *mantissa* field. The regime field is encoded by a run-length method, leading to the boundary between exponent and mantissa is changeable, which is the principal difference between posit and float. About the value of regime field, consecutive K '1' followed by a '0' equals $K-1$, and consecutive K '0' followed by a '1' equals $-K$. For clarity, the mapping between regime binary codes and regime values is shown in Table 1, in which the 4-bit case is taken as an example. In addition, unlike float that leaves some bit patterns to represent exception cases, such as ± 0, $\pm\infty$, and NaN, posit only uses a bit pattern for zero, and a bit pattern for infinity. In this way, more bit patterns can be reserved for meaningful values.

TABLE 2
Dynamic Ranges of Various Data Types

Data Type	Bit Configuration	Maxpos	Minpos
Fixed	8	127	1
	16	32765	1
Float(n, e)	(8, 5)	57344	1.52e-5
	(16, 5)	65535	5.96e-8
Posit(n, es)	(8, 0)	64	0.015625
	(8, 1)	4096	2.44e-4
	(8, 2)	1.68e8	5.96e-8
	(16, 0)	16384	6.10e-5
	(16, 1)	2.68e8	3.72e-9
	(16, 2)	7.20e16	1.39e-17

For an (n, es) posit with binary code p, its numerical value x_p is given by Eq. (2), where $useed = 2^{2^{es}}$, r is the regime value, e is the unsigned exponent value, and m is the mantissa value in fractional format. By packaging the regime field and exponent field with formula $useed^r \times 2^e$, the result actually plays the role of exponent field in the float. To avoid confusion, the packaged result will be referred to as *effective exponent* hereinafter.

$$x_p = \begin{cases} 0 & p = 00\ldots00, \\ \pm\infty & p = 10\ldots00, \\ s \times useed^r \times 2^e \times (1+m) & otherwise. \end{cases} \tag{2}$$

2.3 Related Work

With the significant advance of DNNs, many researchers' efforts have been devoted to the efficient model acceleration approaches. Quantization, namely reduced-precision representation, has made great contributions in this area.

BWN [1] preserved weights in binary. BNN [2] and XNOR-Net [11] further binarized both weights and activations. Hence most forward computations can be accomplished by bit-operations. In [12], the binary logarithmic was exploited to represent data in both training and inference procedures. Then the multiplication can be replaced by a shift operation. Dorefa-Net [4] proposed a framework for training DNNs with different bit widths for weights, activations, and gradients. However, these works caused noticeable accuracy degradation for large-scale models and datasets. Guapta *et al.* [13] explored to train DNNs with 16-bit fixed-point, in which stochastic rounding method was utilized. An adaptive scaling factor was then introduced in Flexpoint [7] to improve model accuracy, but the calculation of the scaling factor was complex. To realize the baseline accuracy with reduced-precision, Micikevicius *et al.* [14] declared that training DNNs with FP16 could reach baseline accuracy on multiple applications. In [14], the loss scaling was proposed to make gradients propagation effectively, and a FP32 weight master was used for weight update. With chunk-based accumulation, mixed FP8/FP16 also showed excellent performance in [15]. Besides, the modified FP16 (BFLOAT16) format [16] was studied for DNN training.

As for the posit application in DNNs, there are also some initial works. Deep Positron [17], a DNN inference accelerator which employs exact-multiply-and-accumulates (EMACs) for 8-bit posit, demonstrated better accuracy than 8-bit fixed-point and floating-point. Cheetah [5] evaluated various posit formats both for training and inference on MNIST, Fashion MNIST, and CIFAR-10. *J.Johnson* [18] designed a log-posit arithmetic unit for inference and evaluated it on ResNet50 with ImageNet, which revealed negligible accuracy loss and higher energy efficiency compared to 8-bit integer format.

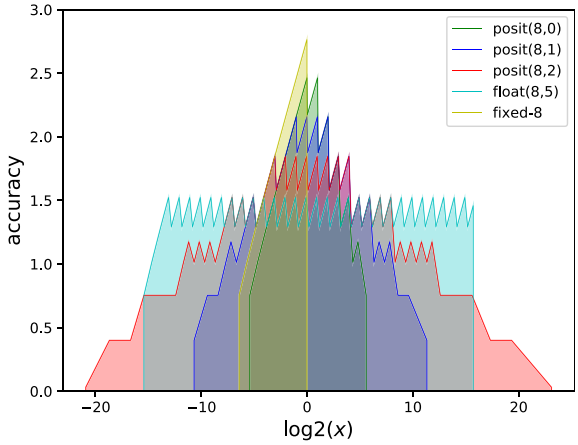


Fig. 2. Decimal accuracies of various data types.

3 DNN Training Framework With Posit

3.1 Quantization Analysis

In this section, we conduct a detailed analysis of the representation ability of various data types, including fixed-point, floating-point, and posit. To measure the representation ability of a data type, both dynamic range and numerical precision are essential metrics. Furthermore, the relative error between computed value and real value is usually applied to indicate quantization error. Therefore, we compare various data types in terms of dynamic range, decimal accuracy, and mean relative error, respectively.

As one can see from Table 2, the dynamic range of posit numbers can be adjusted by es values. By choosing suitable es, an 8-bit posit is able to suppress 16-bit float in terms of range. Of course, it is inevitable to bring the degradation of numerical precision for part of values. The variety of numerical precision with the magnitude of values can be quantified by *decimal accuracy* [8], which is defined as

$$-\log_{10}\big(|\log_{10}(q/x)|\big), \tag{3}$$

where x is a real number and q is the quantized value of x.

For each data type, we define a number set $\mathcal{S}$ which includes all positive points. Then these sets are interpolated uniformly and used to compute decimal accuracy. The decimal accuracies of various data types are plotted in Fig. 2. For clarity, the fixed-point number is normalized by its maximum value. As shown in Fig. 2, the fixed-point number has the narrowest range, and its accuracy drops quickly as the value decreases. However, its maximum accuracy is the highest in comparison with those of other formats. Therefore, fixed-point is more suitable for data with a narrow range and uniform distribution. As for float, it has constant accuracy in the normal interval and tapered accuracy in the subnormal interval. Unless the data has a uniform distribution in terms of the order of magnitude, it will be unnecessary and inefficient to

use float, as many bit patterns are not exploited sufficiently. By contrast, posit shows different behavior. Its peak accuracy is reached with $\log_2 x = 0$, i.e., $x = 1$. As the value increases or decreases, the accuracy will gradually decrease. This trend naturally reminds us of the single peak data distribution. Based on our observations and previous works, most data in DNNs have normal distribution approximately. The order of magnitude of the data, i.e., $\log_2(|X|)$, has an *exp-normal distribution* [19], which is also a single peak distribution. In conclusion, posit is very suitable for representing data in DNNs.

By comparing the accuracy curves of posit data with various es values in Fig. 2, we can learn how the dynamic range and numerical precision are controlled by the choices of n and es. This provides us a preliminary intuition on setting an appropriate posit configuration for DNNs. Detailed and comprehensive comparisons will be discussed later.

To verify the superiority of posit in the numerical representation of DNNs, we compute the quantization errors of these data types in analytical and empirical processes, respectively. In the analytical process, we adopt the mean relative errors over the normal distribution as the measurement, where the relative error $\mathcal{E}$ is given by $|(x-q)/(x)|$.

Suppose a continuous random variable X has normal distribution whose mean is zero, written as $X \sim \mathcal{N}(0, \sigma^2)$. Here $\sigma^2 = \mathrm{var}[X]$ is the variance. Its probability density function is given by

$$p(x) = \mathcal{N}(x|0, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right). \tag{4}$$

The intuitive consideration about the calculation of mean relative error is to integrate the error function over the distribution as

$$\begin{aligned} \mathbb{E}_x[\mathcal{E}] &= \int \mathcal{E}\, p(x)\, \mathrm{d}x \\ &= \int \left|\frac{x-q}{x}\right| \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx. \end{aligned} \tag{5}$$

However, this equation can not be solved. We transform Eqs. (5) to (6) by approximating the $\mathcal{E}$ with $\mathcal{E}^2$.

$$\begin{aligned} \mathbb{E}_x\left[\mathcal{E}^2\right] &= \int \mathcal{E}^2\, p(x)\, \mathrm{d}x \\ &= \int \left(\frac{x-q}{x}\right)^2 \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx. \end{aligned} \tag{6}$$

Its result is as follows:

$$\begin{aligned} \mathbb{E}_x\left[\mathcal{E}^2\right] = \frac{1}{\sqrt{2\pi\sigma^2}} \Bigg\{ & \frac{\sqrt{2\pi}(\sigma^2 - q^2)\mathrm{erf}\left(\frac{x}{\sqrt{2}\sigma}\right)}{2\sigma} \\ & - \frac{\exp\left(-\frac{x^2}{2\sigma^2}\right) q^2 + \mathrm{Ei}\left(-\frac{x^2}{2\sigma^2}\right) x\, q}{x} \Bigg\}, \end{aligned} \tag{7}$$

where $\mathrm{erf}(\cdot)$ is the *error function* defined as

$$\mathrm{erf}(x) \triangleq \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t,$$

and $\mathrm{Ei}(\cdot)$ is the *exponential integral* given by

$$\mathrm{Ei}(x) \triangleq -\int_{-x}^{\infty} \frac{e^{-t}}{t} \mathrm{d}t.$$

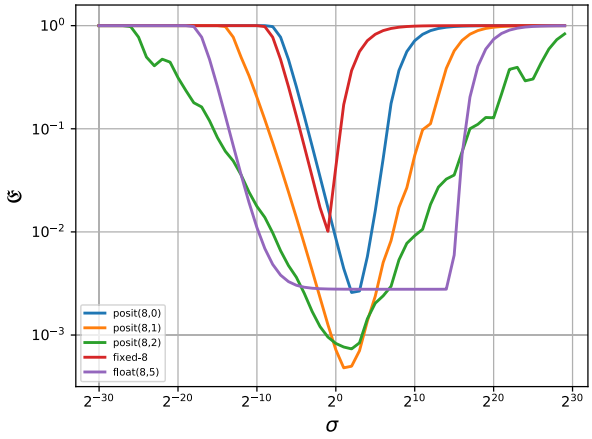


Fig. 3. Mean relative errors of various data types.

Though the above two functions do not have closed expression, many mathematical tools are capable of giving the corresponding numerical solutions.

Then let us discuss the total mean relative error for a data format. For each data type, there is a number set $\mathcal{Q}$. Let the set $\mathcal{Q}$ be ordered and non-repetitive

$$\mathcal{Q} = \{q_1, q_2, \ldots q_i, \ldots q_N\}, \quad \text{s.t.} \begin{cases} q_1 < q_2 < \ldots < q_i \ldots < q_N, \\ q_i \in \mathbb{R}. \end{cases}$$

As the variable X is formatted, the values in an interval are mapped to one point q_i in $\mathcal{Q}$, whose boundary depends on round-off method. Here we adopt the round-to-nearest, so the boundary is the middle of two adjacent points in $\mathcal{Q}$, which means the radiation interval of q_i is

$$\begin{cases} [\frac{q_{i-1}+q_i}{2}, \frac{q_i+q_{i+1}}{2}) & \text{for } i = 2, 3, \ldots N-1, \\ (-\infty, \frac{q_i+q_{i+1}}{2}) & \text{for } i = 1, \\ [\frac{q_{i-1}+q_i}{2}, +\infty) & \text{for } i = N. \end{cases}$$

The mean error $\mathfrak{E}_i$ at q_i can be obtained by substituting the boundary into Eq. (7). The total mean error $\mathfrak{E}$ will therefore be the summation over the whole set $\mathcal{Q}$.

$$\begin{aligned} \mathfrak{E} = \mathbb{E}_{\mathcal{Q}}[\mathfrak{E}_i] = & \int_0^{\frac{q_1+q_2}{2}} \left(\frac{x-q_1}{x}\right)^2 p(x)\mathrm{d}x \\ & + \sum_{i=2}^{N-1} \int_{\frac{q_{i-1}+q_i}{2}}^{\frac{q_i+q_{i+1}}{2}} \left(\frac{x-q_i}{x}\right)^2 p(x)\mathrm{d}x \\ & + \int_{\frac{q_{N-1}+q_N}{2}}^{\infty} \left(\frac{x-q_N}{x}\right)^2 p(x)\mathrm{d}x. \end{aligned} \tag{8}$$

By applying Eq. (8) on those data formats with different σ values, the comparison results are shown in Fig. 3. For posit and fixed-point, we can see that there is a V-shaped curve. It means that there would be an optimal σ value that minimizes $\mathfrak{E}$ for a certain data type. By multiplying with a proper factor, the data distribution will have the optimal variance. However, we see a U-shaped error curve for floating-point. As we claimed above, there is a plateau in a relatively large range, which means that we cannot improve the accuracy of floating-point unless data are out of the dynamic range.

As for the empirical test about the effectiveness of posit, we also calculated the quantization errors on synthesized

TABLE 3
Quantization Errors of Various Data Types on Selected Data Sources

Data Source	Data Type	Mean Relative Error	Mean Absolute Error
$\mathcal{N}(0,1)$	fixed-8	0.070	1.56e-2
	float(8, 5)	0.045	3.58e-2
	posit(8, 0)	0.019	**5.36e-3**
	posit(8, 1)	**0.012**	8.99e-3
	posit(8, 2)	0.022	1.80e-2
$\mathcal{N}(0,0.1^2)$	fixed-8	0.083	1.95e-3
	float(8, 5)	0.045	3.58e-3
	posit(8, 0)	0.139	1.84e-3
	posit(8, 1)	**0.015**	**9.10e-4**
	posit(8, 2)	0.022	1.80e-3
Weights	fixed-8	0.070	1.60e-2
	float(8, 5)	0.045	3.60e-2
	posit(8, 0)	0.019	**5.00e-3**
	posit(8, 1)	**0.012**	9.00e-3
	posit(8, 2)	0.022	1.80e-2
Gradients	fixed-8	0.093	1.56e-5
	float(8, 5)	0.045	2.84e-5
	posit(8, 0)	0.025	**4.21e-6**
	posit(8, 1)	**0.012**	7.10e-6
	posit(8, 2)	0.022	1.42e-5

and real data. The synthesized data are drawn from normal distributions $\mathcal{N}(0,\sigma^2)$ with the size of $[10000, 10000]$. The real data are some tensors sampled from the weights, and gradients during the DNN training process on ImageNet, where the gradient values are multiplied with a factor of 1,000 as discussed in [14]. When quantized through fixed-point, the data is normalized by a tensor-wise scaling factor that is the maximum absolute value of the tensor. As for posit and float, here we directly apply them to quantize data, even though it may not be the best solution for minimizing error. We will discuss how to make a reasonable decision to take the best advantages of posit in Section 3.2. For a comprehensive comparison, here we calculate both the mean relative error and mean absolute error as listed in Table 3. For simplicity, only 8-bit quantization results are shown, and the real data comes from *layer1.1.conv1* in ResNet18. From the comparison results, we can find that 8-bit posit provides the best accuracy performance for both relative error and absolute error. In conclusion, significant benefits will be obtained by applying reduced-precision posit into DNN models.

3.2 Training DNNs With Posit

The proposed framework aims to substitute FP32 with low-bit posit numbers in DNN training while maintaining the model performance of FP32 baseline. In this section, we will introduce the posit transformation algorithm, computation flow and several effective techniques in the training process.

3.2.1 Posit Transformation

Once the word size n and exponent size es are determined, an arbitrary real number x can be transformed into the corresponding posit value x_p by the procedures in Algorithm 1.

TABLE 4
Notations Involved in Posit Transformation

Notation	Definition
n	word size of posit
es	exponent field size of posit
s	sign of the number x
E	effective exponent value of x
r	regime value of x_p
e	exponent value without considering word size limitation
m	mantissa value without considering word size limitation
b_r	regime field size of x_p
b_e	actual exponent field size of x_p
b_m	actual mantissa field size of x_p
e_p	final exponent value of x_p
m_p	final mantissa value of x_p

The related notations in Algorithm 1 are defined in Table 4. First we have to limit the magnitude of x based on the dynamic range of posit(n, es) and then extract sign, regime, exponent, and mantissa field. Next, because of the restriction of word size, the width of each field is adjusted. Therefore, the rounding operations are applied to the value of each field to fit the adjusted width. Finally, the posit result x_p is attained by combining these parts based on Eq. (2). When compared to the original posit in [8], the underflow case is allowed in our transformation algorithm (lines 4-6). This is inspired by the fact that the small values can be set to zero in DNNs without hurting model performance [3].

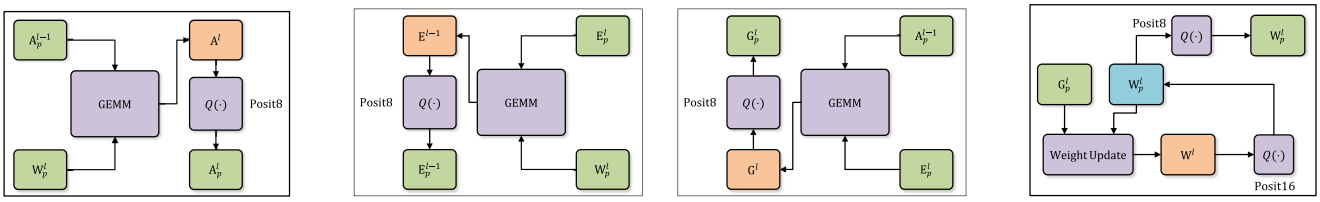
Algorithm 1. Posit Transformation

Require: $x \in \mathbb{R}$, posit word size $n \in \mathbb{N}^+$, and exponent field size $es \in \mathbb{N}$

1: $useed \leftarrow 2^{2^{es}}$
2: $maxpos \leftarrow useed^{n-2}$
3: $minpos \leftarrow useed^{2-n}$
4: $threshold \leftarrow minpos \div 2$
5: **if** $abs(x) < threshold$ **then**
6: $\quad x_p \leftarrow 0$
7: **else**
8: $\quad s \leftarrow Sign(x)$
9: $\quad \hat{x} \leftarrow Clamp\{abs(x), minpos, maxpos\}$
10: $\quad E \leftarrow \lfloor \log_2 \hat{x} \rfloor$
11: $\quad r \leftarrow \lfloor E \div 2^{es} \rfloor$
12: $\quad e \leftarrow E \pmod{2^{es}}$
13: $\quad m \leftarrow \hat{x} \div 2^E - 1$
14: $\quad$ **if** $r \geq 0$ **then**
15: $\quad\quad b_r \leftarrow r + 2$
16: $\quad$ **else**
17: $\quad\quad b_r \leftarrow -r + 1$
18: $\quad$ **end if**
19: $\quad b_e \leftarrow min\{n - 1 - b_r, es\}$
20: $\quad b_m \leftarrow min\{n - 1 - b_r - b_e, 0\}$
21: $\quad e_p \leftarrow \lfloor e \times 2^{b_e - es} \rfloor \times 2^{es - b_e}$
22: $\quad m_p \leftarrow \lfloor m \times 2^{b_m} \rfloor \times 2^{-b_m}$
23: $\quad x_p \leftarrow s \times useed^r \times 2^{e_p} \times (1 + m_p)$
24: **end if**
25: **return** x_p

3.2.2 Computation Flow

The computation flow of the framework is shown in Fig. 4, containing data format configurations during forward



(a) Forward propagation (b) Backward propagation 1 (c) Backward propagation 2 (d) Weight update

Fig. 4. DNN training computation flow graph with posit transformation. In the graph, $Q(\cdot)$ means the transformation operation, whose subscript (n, es) is omitted for simplicity. Besides, W, A, G, and E stand for the weights, activations, weight gradients, and activation gradients in a layer, respectively. The symbols with subscript p are in posit format.

propagation (Fig. 4a), backward propagation (Figs. 4b and 4c), and weight update process (Fig. 4d). Here we take the general matrix multiply (GEMM) as the core computations for a brief description. Depending on the model architecture, this component can be just used to complete different kinds of modules such as convolutional layers and fully connected layers. From Figs. 4a, 4b, and 4c, all involved data in forward and backward propagation, including weights (W), activations (A), weight gradients (G), and activation gradients (error, E), are converted into 8-bit posit (posit8) numbers with the proposed Algorithm 1. As for the weight update process in Fig. 4d, we take weight master copy technique [14] for storing and updating weights in 16-bit posit (posit16) numbers. In the posit transformation, the choice of rounding methods, including round-to-zero, round-to-nearest, and stochastic rounding, is carefully considered. In our framework, the round-to-nearest method is finally adopted because of its simplicity and comparable model performance.

3.2.3 Motivations

As mentioned above, posit has many superior properties for DNN training. However, if we replace FP32 with reduced-precision posit directly, models will be limited in those simple tasks as presented in [5]. Once being applied for large-scale datasets (e.g., ImageNet), models are likely to fail to converge.

We think there are several possible reasons as follows:

- From the results of quantization analysis about posit, we know that the decimal accuracy is basically symmetrical around 1, and the quantization error has a V-shaped curve with respect to the variance of data distribution. While one tensor data (such as the weights of a layer) in a DNN model is distributed over a limited range, sometimes it has a too big or too small variance that leads to a mismatch between data distribution and posit accuracy distribution. In this situation, the quantization error will be relatively larger, just as $X \sim \mathcal{N}(0, 0.1^2)$ represented by posit(8,0) in Table 3.
- In different layers, the data have different ranges, which means some data distributions are more concentrated and the others are relatively decentralized. In addition, some layers are more sensitive to error, for instance, the first layer and the last layer. Even though the quantization errors are the same, these layers cause more impacts on model performance than other layers. Therefore, it is sub-optimal to represent them in the same representation precision (n and es).

Our proposed methods mainly focus on solving the above problems and enable effective DNN training with reduced-precision posit.

3.2.4 Distribution-Based Scaling Factors

When quantizing a real number x to its reduced-precision format, the vanilla strategy is mapping it to the nearest value in the number set $\mathcal{Q}$ of a certain data type, just like we have done in Section 3.1. It naturally brings inevitable numerical error. To solve the problem of mismatch between data distribution and posit accuracy distribution and minimize the quantization error of a determined posit format, an intuitive consideration is to shift the data into a more appropriate range. The shift-based mapping method in [20], which is adopted for fixed-point number, normalizes a tensor data by dividing its maximum absolute value before quantization. Inspired by this way, a tensor-wise scaling factor for posit quantization is proposed in this work.

Because the decimal accuracy of posit is symmetrical about 1 (i.e., 2^0) as seen from Fig. 2, we can shift the center of data distribution in log-domain to 1 as shown in Eq. (9) [21].

$$s_l = \text{pow}\left(2, \frac{1}{N}\sum_{i=1}^{N}\log_2(|x_i|)\right), X_l^q = Q\left(\frac{X}{s_l}\right)s_l, \tag{9}$$

where X indicates an input tensor data that has N elements x_i, $\text{pow}(2,\cdot)$ means an exponential function based with 2, s_l is the tensor-wise scaling factor derived from log-domain distribution, $Q(\cdot)$ is the quantization function (i.e., Algorithm 1), and X_l^q denotes the quantized input by using the scaling factor s_l. In this way, the numerical interval which contains higher data density will be assigned higher representation precision.

From another perspective, we get a V-shaped curve for posit in Fig. 3. The quantization error of posit is almost minimum when the variance of data distribution is near to 1. The data distribution can be scaled by changing the variance as shown in Eq. (10) to minimize error.

$$s_v = \sqrt{\Sigma_{i=1}^{N}(x_i - \bar{x})^2}, X_v^q = Q\left(\frac{X}{s_v}\right)s_v, \tag{10}$$

where $\bar{x}$ is the mean of X, s_v is the variance-based scaling factor, i.e., standard deviation, and X_v^q denotes the quantized data by using the scaling factor s_v.

In fact, there is an equivalence relationship between these two scaling factors when data have a normal distribution. The proof is as follows:

TABLE 5
Quantization Errors of Various Scaling Factors on Selected Data Sources

Data Source	V_l	Data Type	Scaling Factor	Mean Relative Error	Mean Absolute Error
Weights of *layer2.1.conv2* in ResNet18 (s_l = 0.0247, s_v = 0.0259)	2.635	posit(8, 1)	None	0.020	**4.54e-4**
			s_l	**0.012**	5.00e-4
			s_v	**0.012**	4.92e-4
		posit(8, 2)	None	0.023	8.57e-4
			s_l	0.022	8.57e-4
			s_v	0.022	8.56e-4
Activations of *layer2.0.conv1* in ResNet18 (s_l = 2.27, s_v = 1.73)	2.4704	posit(8, 1)	None	0.015	6.31e-2
			s_l	**0.012**	**4.26e-2**
			s_v	0.013	4.92e-2
		posit(8, 2)	None	0.022	7.64e-2
			s_l	0.022	7.60e-2
			s_v	0.022	7.65e-2
Gradients of *layer4.0.conv2* in ResNet18 (s_l = 2.84e-7, s_v= 3.59e-6)	8.9099	posit(8, 1)	None	1.000	2.44e-6
			s_l	0.019	1.46e-7
			s_v	**0.018**	**4.40e-8**
		posit(8, 2)	None	0.315	1.79e-7
			s_l	0.026	2.83e-7
			s_v	0.023	5.66e-8
Gradients of *layer1.1.bn* in ResNet18 (s_l = 3.13e-21, s_v = 1.30e-6)	1836.08	posit(8, 1)	None	0.443	9.25e-7
			s_l	0.443	9.25e-7
			s_v	**0.006**	**1.60e-8**
		posit(8, 2)	None	0.074	8.59e-8
			s_l	0.443	9.25e-7
			s_v	0.010	2.17e-8

Assuming random variable x has a normal distribution, $x \sim \mathcal{N}(0, \sigma^2)$, the mean value of $\log_2|x|$ can be calculated by

$$\begin{aligned}\mathbb{E}_x[\log_2|x|] &= \int_{-\infty}^{\infty} \log_2|x| p(x)\, \mathrm{d}x \\ &= \int_{-\infty}^{\infty} \log_2|x| \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\ &= -\frac{\gamma + \log(2) - 2\log(\sigma)}{2\log(2)}.\end{aligned} \tag{11}$$

Here, γ is the Euler's constant with numerical value $0.577216\ldots$ [22].

Therefore the relationship between s_v and s_l can be formulated as

$$\begin{aligned}s_l &= \mathrm{pow}\left(2, -\frac{\gamma + \log(2) - 2\log(\sigma)}{2\log(2)}\right) \\ &= \frac{e^{-\gamma/2}\sigma}{\sqrt{2}} \\ &\approx 0.52984 s_v.\end{aligned} \tag{12}$$

Table 5 gives the quantization errors for posit with three different quantization strategies, including vanilla method (no scaling factor), log-mean-based scaling factor (s_l), and variance-based scaling factor (s_v). Here we multiply s_v with the ratio coefficient in Eq. (12). The data are sampled from a ResNet18 model during training on ImageNet, including weights, activations, and gradients in different layers. Various posit formats are considered to further guide the choice of es. From Fig. 3, we can see that the mean errors of posit(8, 0) are no better than those of float (8, 5) for different σ values, even at the minimal point. Therefore, posit(8, 0) is deemed as an inappropriate candidate for DNN training, and is omitted here. As we can see, quantization errors are relatively smaller in most cases where either s_l or s_v is used, which proves the effectiveness of our scaling methods. However, there are some special cases to be explained.

For weights and activations, with the increase of es, the influence of the choices of scaling factors declines, since both s_v and s_l are similar and close to 1. For gradients in CONV layer, the superiority of scaling method becomes significant, because the gradients are very small and exceed the representation range of 8-bit posit. Nevertheless, for gradients in batch normalization (BN) layer, the values of s_l and s_v show huge difference, and the results of s_l are even worse than that of the vanilla method. Because this layer is adjacent to ReLU, where large sparsity exists. The sparsity causes the data distribution to deviate severely from normal, which makes the s_l inappropriate. But s_v is more robust in this case. Hence we will take s_v as the choice in our framework implementation.

Since the scaling factor s_v is determined, let us make a comparison among various posit formats. It is obvious that posit(8, 1) shows better accuracy than posit(8, 2) in all cases listed in Table 5, both for relative error and absolute error. Therefore, we choose posit(8, 1) in subsequent training experiments.

Here the variance of data in log-domain (V_l) is exploited to measure the deviation from the normal distribution. Intuitively, this statistic implies the range of the order of magnitude. From Eq. (13), the variance of $\log_2|x|$, where $x \sim \mathcal{N}(0, \sigma^2)$, is a constant. As shown in Table 5, weights and activations have similar V_l to normal distribution. In a CONV layer, the V_l of gradients is slightly larger than that of weights, leading to the relative error larger than those of weights and activations, especially when es is small. As for the gradients of BN layer, because of the sparsity, the value of V_l is unreasonably large.

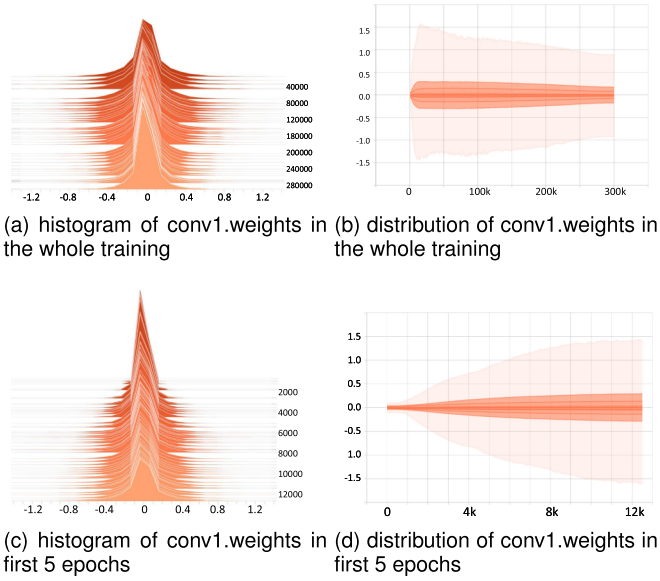


Fig. 5. The histograms and distributions of CONV in ResNet18 training process on ImageNet.

$$V_l = \mathrm{Var}_x[\log_2|x|] = \int_{-\infty}^{\infty} \left(\log_2|x| - \mathbb{E}_x[\log_2|x|]\right)^2 \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \frac{\pi^2}{8\log(2)^2} \approx 2.5678. \tag{13}$$

Moreover, considering these large values have more importance in DNNs [3], a constant coefficient β is multiplied with s_v. From the perspective of probability, the βs_v, i.e., $\beta\sigma$ of $\mathcal{N}(0, \sigma^2)$, corresponds to the point of pth percentage of data, which means the p% (68 percent if $\beta = 1$) smaller data is represented by 50 percent bit patterns.

3.2.5 Warmup Training

By observing the distributions of data in the training process in Fig. 5, the distributions of the weights have a steep change in the first several epochs, which is caused by the warmup learning rate schedule. Fortunately, the data distribution becomes stable in later epochs. On account of this phenomenon, a warmup training using FP32 companied with a warmup learning rate schedule for the same epochs is carried out. At the end of the warmup process, data distribution of each layer can be calculated and the corresponding scaling factors can be determined. The scaling factors could be fixed later unless the model loss does not decrease any longer during the training process. It is noticed that the warmup training strategy is not necessary unless the warmup learning schedule is adopted.

4 Experiments Results

In this section, we have performed experiments over various datasets and models to verify the effectiveness of our posit training strategy. All experiments are evaluated with PyTorch framework on NVIDIA Tesla V100 GPUs. The basic configurations for our framework in the experiments are as follows:

- *Bit Configurations*: We adopt posit(8, 1) for all layers but the last layer of models, while the last layer is represented in posit(16, 1). Posit(16, 1) is also applied for weights master copy.
- *Scaling Factors*: As discussed in Section 3.2.4, the variance-based scaling factor s_v is selected in experiments.

4.1 Image Classification

Our framework is tested on CNNs for image classification tasks: MNIST, CIFAR-10, ImageNet. The introduction of datasets and related training strategies are as follows:

- MNIST dataset has 28×28 pixels grey images with 10 classes, whose training set has 60,000 pictures and testing set has 10,000 pictures. LeNet-5 [25] is trained on it for 15 epochs. Here we use stochastic gradient descent with momentum 0.5 as the optimizer. The learning rate is fixed at 0.01. The number of warmup epochs is set as 1.
- CIFAR-10 dataset contains 32×32 pixels images with 10 classes, in which 50,000 pictures for training and 10,000 pictures for testing. CIFAR-ResNet20 [26] is trained for 200 epochs. In the training process, the model uses Nesterov accelerated gradient with momentum 0.9 as the optimizer. The learning rate is initialized to 0.1 and divided by 10 at 100th, 150th epoch. The images of the training set are enhanced with randomly cropping, horizontally flipping, and normalizing RGB channels. The number of warmup epochs is set as 1.
- ImageNet dataset includes images of 1,000 classes totally, where 1 million pictures for training and 50,000 pictures for validation. We evaluated AlexNet [27], ResNet18 [26], and MobileNet-V2 [28] for ImageNet. All models use Nesterov accelerated gradient with momentum 0.9 as the optimizer. For ResNet18 and MobileNet-V2, the learning rate is linearly increased to 0.1 in 5-epoch warmup and then decreased with the cosine annealing strategy. For AlexNet, the learning rate is initialized to 0.01 and divided by 10 every 40 epochs. About data augmentation, we follow the baseline training procedures listed in [29].

The validation accuracies of these models are summarized in Table 6.

Table 7 shows a comparison of various reduced-precision training strategies on ImageNet. The posit training framework shows the lowest accuracy loss and the highest compression ratio among these methods. We compute the overall compression ratio in terms of memory requirements. For each module, including CONV, BN, and FC, all involved data are considered during different phases, i.e., W, A, G, E, and UP. The total memory requirements are calculated by Eq. (14).

$$\begin{aligned} Total = {} & Num\_W \times Bitwidth\_W + Num\_A \times Bitwidth\_A \\ & + Num\_G \times Bitwidth\_G + Num\_E \times Bitwidth\_E \\ & + Num\_UP \times Bitwidth\_UP. \end{aligned} \tag{14}$$

It is noticed that the compression ratio is related to batch size because A and E are considered. Here we only list the results under the batch size equals 256 for saving space.

TABLE 6
Image Classification Top-1(Top-5) Validation Accuracy

Dataset	Model	Batch Size	Epochs	Baseline Top-1/Top-5 (%)	Posit Top-1/Top-5 (%)	Accuracy Degradation Top-1/Top-5 (%)
MNIST	LeNet-5	64	15	98.90	98.90	0.00
CIFAR-10	CIFAR-Resnet20	128	200	92.21	92.07	0.14
ImageNet	AlexNet	256	160	56.37/79.11	56.04/78.74	0.33/0.37
	ResNet18	256	120	70.94/89.95	70.83/90.05	0.11/-0.10
	MobileNet-V2	512	150	71.93/90.46	71.63/90.35	0.30/0.09

TABLE 7
Comparison of Reduced Precision Training Strategies on ImageNet

Method	Top-1 (%) Baseline / Reduced Precision / Accuracy Loss		Compression Ratio †		Bit Configuration					BN Quantization
	AlexNet	ResNet18	AlexNet	ResNet18	W	A	G	E	UP	
Dorefa [4]	55.90/51.60/4.30	N/A	1.70	1.51	1	2	32	6	32	No
DFP [23]	57.43/56.94/0.49	N/A	1.43	1.24	16	16	16	16	32	No
WAGEUBN [24]	N/A	68.70/66.92/1.78	1.95	2.57	8	8	8	8/16	24	Yes
FP8 [15]	58.04/57.55/0.49	67.43/66.95/0.48	3.47	1.58	8	8	8	8	16	No
Posit	56.37/56.04/**0.33**	70.94/70.83/**0.11**	3.47	3.98	8	8	8	8	16	Yes

† : *The compression ratios are calculated under the batch size equals 256.*

4.2 Language Modeling

We also conduct experiments on a word-level language modeling task over the Penn Treebank (PTB) dataset[30]. The model has 3 layers of 1,150 LSTM cells with a 400-dimension embedding layer where weight tying is employed. SGD is applied for 500 epochs with the learning rate is 30. The batch size is 20, and the sequence length is 70. L2 weight decay of 1.2e-6 and dropout rate of 0.4 are adopted for regularization. The test results are listed in Table 8.

4.3 Discussion

4.3.1 Precision of the Last Layer

Normally the last layers of DNNs are considered to be very sensitive to quantization [4]. Here we give a detailed comparison of the different parts in the last fully connected (FC) layer. To explore the impact of the precision of the last layer on model accuracy, the bit widths of data are reduced to 8-bit one by one. As shown in Fig. 6a, the model displays baseline accuracy while all data are preserved in posit16 in the last layer. As only the activations are kept in 16-bit (i.e., weights, biases, and all gradients are reduced to 8-bit), the model still keeps performance (16-bit for Activation in FC). However, an accuracy gap (1.3 percent) appears as soon as the activations are represented in 8-bit (16-bit for Error in FC). If posit8 is used for all computations in the last layer, the model performance drops more (1.9 percent).

4.3.2 Scaling Factor

The scaling factor method is essential in most quantization related works. When represented in low-precision floating-point [6], [15], the loss-scaling method is adopted to preserve the dynamic range of gradients with small magnitudes. But it is not a suitable solution to our posit training framework. We conduct experiments on AlexNet to investigate the impact of the scaling factor method. From Fig. 6b we can see, the model that just uses a loss-scaling factor of 1000 suffers obvious accuracy degradation (2.3 percent), demonstrating the necessity of our distribution-based scaling factor method.

TABLE 8
Language Model Results on Word-Level PTB

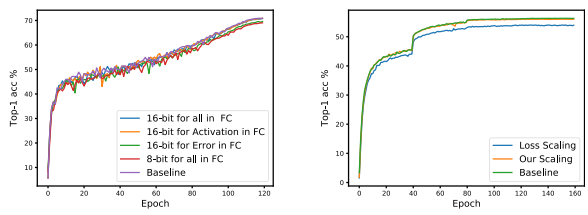
Dataset	Penn Treebank	
Model	Baseline	Posit
Perplexity	61.6825	61.7072
Bit Per Character	5.9468	5.9474
Loss	4.1220	4.1224

5 Hardware Prototype

It is inefficient to train a deep neural network in posit number system with GPUs. For one thing, training with GPUs, which are well-known power-hungry devices, leads to massive energy consumption. For another, currently there is no off-the-shelf computing architecture to support efficient posit arithmetic operation in GPUs, which heavily slows down the processing speed. Moreover, the GEMM dominates DNN training computation as depicted in Fig. 4. Therefore, in this section, a systolic array comprised of posit processing elements (PPEs) is proposed to support efficient GEMM in posit format, where the computational process is completely consistent with the simulation on GPUs. The proposed array also convincingly demonstrates the improvement of the posit quantization algorithm from a hardware-oriented perspective.

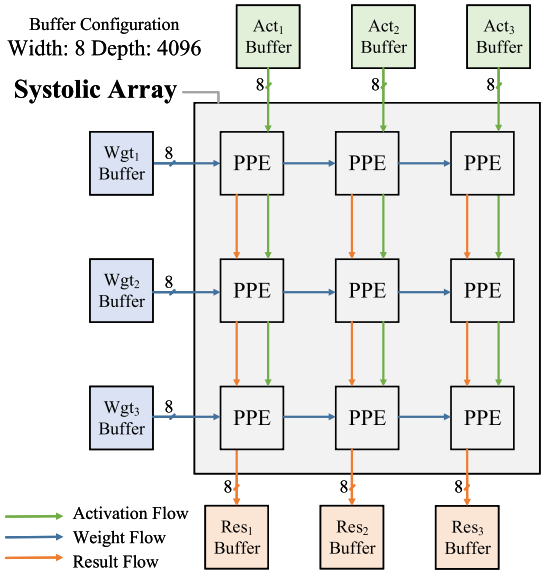
5.1 Posit Systolic Array System

The novel 2-D systolic array architecture for posit GEMM is presented in Fig. 7, with details of its interconnection

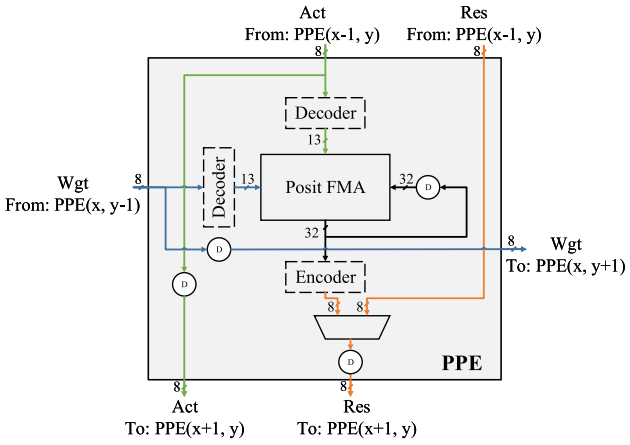


(a) Precision of Last Layer (b) Scaling factor

Fig. 6. The impact of (a) the last layer precision on ResNet18 , (b) scaling factors on AlexNet on model convergence.



(a) Structure of a 3x3 posit systolic array system



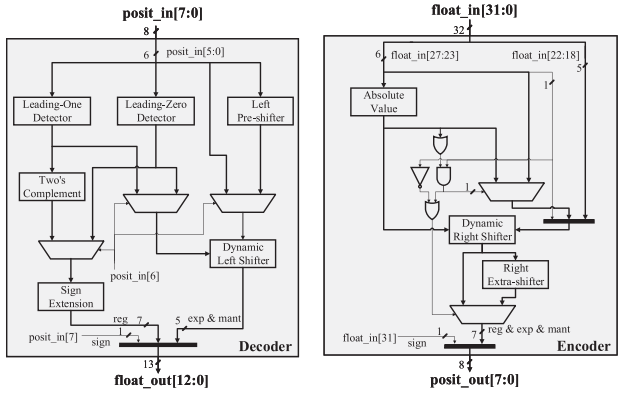
(b) Dataflow inside a PPE.

Fig. 7. Hareware prototype for efficient GEMM in posit format.

between PPEs and the intraconnection inside a PPE. The posit data loaded from external memory are temporarily stored in a specific order in the activation and weight buffers, and the results in posit format are stored in the result buffers. Buffers in Fig. 7 are dual-port SRAMs with a width of 8-bit and a depth of 4,096.

For the interconnection of PPEs shown in Fig. 7a, each PPE horizontally shifts the local weight and vertically shifts the local activation to the neighbouring PPEs in a clock cycle. The interconnection structure is clearly and succinctly demonstrated in a 3x3 array, while the proposed array can be easily scaled to any size. The systolic array system tackles the timing issue for massive parallelization within GEMM. The global and large fan-out interconnections are split into the local interconnection between neighbouring PEs.

The intraconnection of a PPE is presented in Fig. 7b. For the PPE with index (x, y), it passes weight data right to PPE (x, y+1) and activation data down to PPE(x+1, y). When the 8-bit weight and activation in posit format are acquired from neighbouring PPEs, they are first transformed into a 13-bit floating format. The most significant bit represents



(a) The optimized decoder (b) The optimized encoder

Fig. 8. The architecture of optimized decoder and encoder.

sign, the following 8 bits represent exponent coded in 2's complement, and the 4 least significant bits represent effective mantissa. The 13-bit floating format can be denoted as float(1, 8, 4). For posit(8, 1) numbers, 5 bits and 4 bits are required to store its exponent and mantissa, respectively. Nevertheless, in order to maintain the consistency with GPU simulation, we need to expand the dynamic range of posit(8, 1) number to match that of a single-precision floating point number. As a result, we scale the exponent part up to 8 bits by sign extension. There is no need to extend the mantissa part, since it does not affect the accuracy of GEMM. After decoding, the weight and activation in float (1, 8, 4) are passed into a customized posit fused multiply-add(FMA) unit with an accumulated operand in single-precision floating point format, which can be denoted as (1, 8, 23). The posit FMA executes integrated operation including the multiplication of the weight and the activation, and the accumulation of the partial sum. The posit FMA result is stored in float(1, 8, 23) and reused as the accumulated operand in the next cycle. The result also can be passed into the posit encoder and packaged into posit(8, 1) format. Whether the local posit-encoded accumulated result or the result received from PPE(x-1, y) would be transmitted to PPE(x+1, y) depends on the control signal.

To achieve better performance in hardware-oriented posit coding, we specifically optimize the design of decoder and encoder, removing the unnecessary logic in consideration of algorithm-hardware co-design. Fig. 8 provides the schematic diagram of both decoder and encoder. In Fig. 8, the thick lines represent concatenation with no cost. Besides, reg, exp, and mant are short for regime, exponent, and mantissa part, respectively. The posit decoder is used to transform a posit(8,1) number into a float(1,8,4) number, and the posit encoder is used to transform a float(1,8,23) number into a posit(8,1) number. In posit arithmetic, it is required to obtain the 2's complement before decoding the regime, exponent, and fraction if the sign bit is negative, which causes resource waste to execute an extra addition and increases the critical path, incurring additional latency of decoder design. Therefore, from the algorithm-hardware co-design perspective, we determine to bypass the potential 2's complement conversion, and directly decouple real values ignoring the sign. Correspondingly, the excess addition in the encoder is eliminated in the same way.

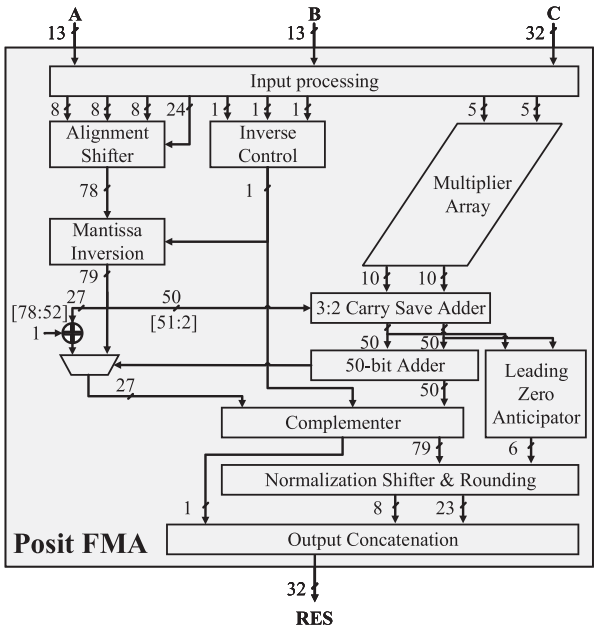


Fig. 9. Datapath of the proposed posit FMA architecture.

On hardware-level optimization, leading zero and leading one is detected in decoder at the same time to speed up the regime value searching, at the cost of the double area of a leading digit detector. Furthermore, a pre-shift module is implemented and overlapped with the process of regime detection to easily obtain the final shift value without addition operation. The encoder is also optimized with this method.

The decoder and encoder depicted in the dotted box in Fig. 7b represent that they can be moved out of the PPE when all PPEs are grouped into a large array for better performance, which would be explored in Section 5.3.

5.2 Posit FMA Design

Since posit is a novel data type totally different from IEEE-754 floating-point, there is no matured IP to support its computation flow. A posit arithmetic core generator is proposed in [31], while it only supports basic adder and multiplier, which are inefficient to handle massive multiply-accumulate operations. Compared to separate posit adder and multiplier, a posit FMA unit has several advantages: 1) In the integrated operation $A \times B + C$, the decoder and encoder are employed only once at the beginning and the end, respectively, instead of multiple times in divided arithmetic unit, and 2) several components, including normalization shifter and leading zero anticipator, are shared for both additions and multiplications, which achieves an area and power reduction in posit FMA unit. The works in [9], [18] presented several efficient posit multiply-accumulate units, whereas they are not consistent with GPU simulation under the proposed algorithm. The accumulator in units above is not large enough to safely store an accurate GEMM result. Moreover, the decoder and encoder are embedded in the units, which would limit flexibility when building a systolic array. The decoder and encoder are more complicated than our designs.

Hence, we propose a posit FMA unit to address the above issues. The design of the posit FMA unit is based on the proposed algorithm. Therefore, the computation result of the posit systolic array is consistent with that of the GPU simulation. Furthermore, the area and power efficiencies of the posit FMA unit are much higher than the computing unit in GPUs. Fig. 9 illustrates the detailed architecture of the proposed posit FMA unit. The bit-width of each signal is marked on the diagram. The posit FMA unit is expected to accept two 13-bit float(1, 8, 4) numbers, namely A and B, and a 32-bit float(1, 8, 23) number, namely C. The FMA executes the integrated operation $A \times B + C$. The result is stored in float(1, 8, 23) format in the posit FMA unit.

To achieve high efficiencies in power and area, we first reduce the size of mantissa multiplication from 24 bits in the traditional float(1, 8, 23) FMA unit to only 5 bits, for the reason that there are only 5 effective bits in float(1, 8, 4) format including the hidden leading-one bit and 4 mantissa bits. The 10-bit product would be filled with zeros from the right of least significant bit, and transferred into a 48-bit product following the traditional float(1, 8, 23) FMA unit.

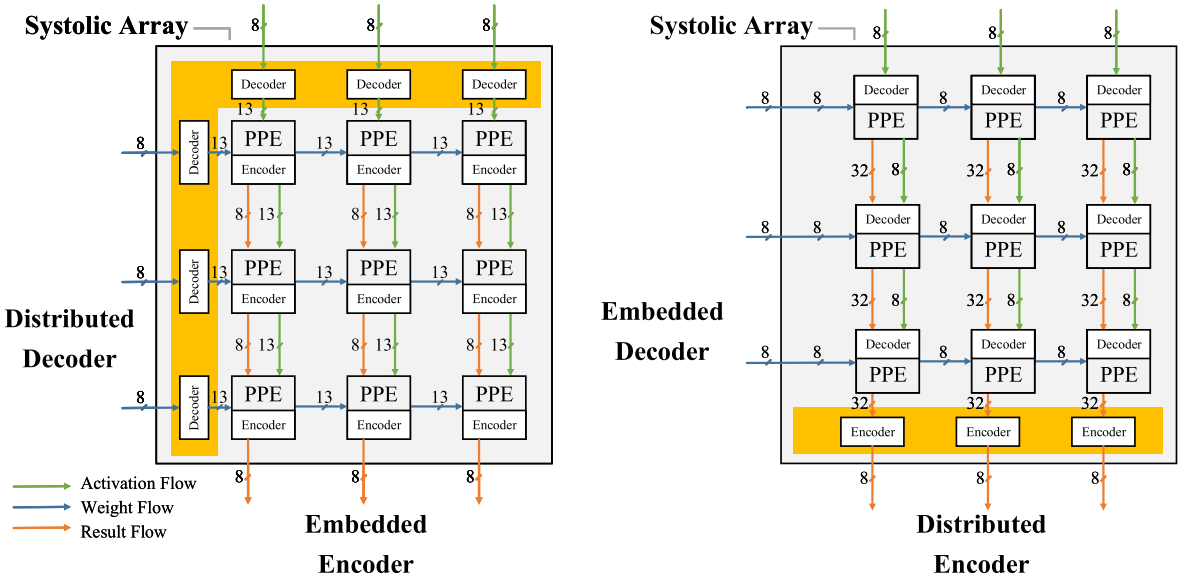


Fig. 10. Left: An example of distributed decoders and embedded encoders. Right: An example of embedded decoders and distributed encoders.

TABLE 9
Synthesis Results with Timing Constraint at 5 ns

	Area (μm^2)	Comb. / Seq. / Mem.	Area Reduction	Power (mW)	Comb. / Seq. / Mem.	Power Reduction
Float(1,8,23) Baseline	28151572	6458250 (22.94%) 467238 (0.02%) 21226084 (75.40%)	-	222.85	54.81 (24.59%) 70.68 (31.71%) 97.37 (43.69%)	-
Embedded Dec. Embedded Enc.	9241154	2028271 (21.95%) 275845 (0.03%) 6937038 (75.07%)	67.17%	115.05	31.96 (27.78%) 40.67 (35.35%) 42.41 (36.87%)	48.37%
Embedded Dec. Distributed Enc.	9019837	1916307 (21.25%) **166491 (0.02%)** 6937038 (76.91%)	67.96%	**108.37**	30.73 (28.36%) **35.23 (32.51%)** 42.41 (39.13%)	**51.37%**
Distributed Dec. Embedded Enc.	9161846	1992125 (21.74%) 232682 (0.03%) 6937038 (75.72%)	67.46%	119.64	**28.67 (23.96%)** 48.56 (40.59%) 42.41 (35.45%)	46.31%
Distributed Dec. Distributed Enc.	**9006612**	**1864343 (20.70%)** 205230 (0.02%) 6937038 (77.02%)	**68.01%**	115.63	30.18 (26.10%) 43.03 (37.22%) 42.41 (36.67%)	48.11%

Since there is no exception observed in the GPU simulation, we remove the exception hardware to acquire an efficient posit FMA design.

We keep the computation result in the proposed FMA consistent with the GPU simulation by following considerations. First, the 50-bit accumulator in the proposed unit is the same as that in the standard single-precision floating-point unit in the GPU. Second, the product of two posit multiplicands is accurate. Finally, the rounding patterns in the unit follow the proposed algorithm, thus keep the consistency with the GPU simulation.

5.3 Exploring Optimization for Posit Systolic Array

The posit decoder and encoder are embedded in the PPE as presented in Fig. 7b at first. However, we observe that the numbers of decoders and encoders can be reduced by utilizing a posit component shared on the array boundary. If posit components including the decoder and encoder are shared, we call it distributed mode since they are arranged at the boundary of the systolic array, or otherwise we name it embedded mode. Both decoder and encoder can be placed in either distributed mode or embedded mode. The details of distributed and embedded posit components placement are depicted in Fig. 10. For simplicity, we only present two feasible arrays to show the placements of decoders and encoders. There are four array arrangement solutions in total by deciding whether decoders and encoders are embedded or distributed.

The number of posit decoders effectively decreases in the distributed arrangement since decoding only occurs at the boundary of the systolic array, which leads to combinational circuit area reduction. Nevertheless, power and area in the sequential logic increase because there are more registers required to temporarily store data.

For posit encoders, more benefits can be obtained by adopting the distributed arrangement instead of the embedded. There is an obvious reduction of combinational logic gates and sequential registers in the encoder-distributed array compared with the encoder-embedded array. The area saving in the combinational circuit is obvious. The encoders are removed in the PPE in the distributed arrangement, and therefore the result bit-width turns to 32 bits and is stored in float(1, 8, 23) instead of posit(8, 1) format. Hence, there is no need to add more registers to store the encoded posit result in the PPE. The 32-bit register can move across the multiplexer considering that the processes of executing FMA operation and transferring result are in different computing stages. Therefore, the number of registers decreases, as well as area and power in the encoder-distributed placement.

The effect of different placement strategies will be examined in Section 5.4 with detailed synthesis evaluation and analysis.

5.4 Performance Evaluation and Analysis

For performance evaluation, we synthesize all solutions of posit systolic array mentioned in Section 5.3 by Synopsys Design Compiler in TSMC 90nm platform. For a fair comparison, a systolic array using classic float(1, 8, 23) FMA unit is also synthesized under the same structure. The classic FMA unit is instantiated by Synopsys DesignWare IP tool. The float(1, 8, 23) systolic array is a baseline as an ideal simulation in GPUs. All the buffers in the synthesized systolic arrays are generated by Artisan 90nm memory generator. The size of all the synthesized arrays is 16x16 and the timing constraint is configured as 5ns. The synthesis results are illustrated in Table 9. A breakdown of major components in the systolic array is also presented in Table 9, including combinational and sequential components in the array, and the memory buffer outside the array.

Compared to the baseline, posit solution can achieve up to 67.17 and 48.37 percent reduction in area and power, respectively. As shown in Table 9, the memory area of posit array drops 67.32 percent compared with the baseline, and thereby power consumption in memory shrinks 56.44 percent. Correspondingly, the combinational logic in the array achieves 68.59 and 41.69 percent decline in area and power, respectively. The sequential logic also has an area reduction of 40.97 percent and a power reduction of 42.46 percent.

When it comes to the optimization exploration, it is observed that decoder-embedded and encoder-distributed

placement acquires 3.00 percent power saving by sharing registers in PPEs in comparison with the both-embedded placement. The decoder-distributed and encoder-distributed placement has an area reduction of 0.87 percent by decreasing the number of decoders and encoders. Taking the area and power consumption into consideration, a decoder-embedded and encoder-distributed placement is the most efficient way to construct a posit systolic array for GEMM.

Furthermore, memory capacity and bandwidth can be reduced by 75 percent since the data width is reduced to 8-bit from 32-bit. As we know, memory access to external DRAM always brings high power consumption, and the bandwidth is a key factor limiting the processing speed. Therefore, many more potential benefits can be obtained with our design.

6 Conclusion

After a comprehensive quantization analysis, a DNN training framework with posit number system is introduced. We demonstrate that training DNNs with 8-bit posit number achieves state-of-the-art performance cross various datasets and model architectures without hyper-parameters tuning. To minimize the quantization error of posit, we propose a tensor-wise scaling factor based on data distribution. With a dedicated posit FMA and a thorough exploration of encoder/decoder placement, an efficient systolic array is designed for posit GEMM. Compared to the corresponding FP32 implementation, our design achieves a reduction of 68, 51, and 75 percent in terms of area, power, and memory capacity, respectively.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61604068 and in part by the Fundamental Research Funds for the Central Universities under Grant 021014380065.

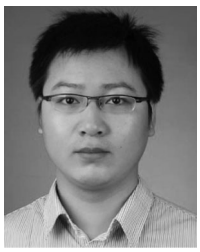
References

[1] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv , and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*.
[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.
[4] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.
[5] H. F. Langroudi, Z. Carmichael, D. Pastuch, and D. Kudithipudi, "Cheetah: Mixed low-precision hardware & software co-design framework for DNNs on the edge," 2019, *arXiv: 1908.02386*.
[6] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul, "Mixed precision training with 8-bit floating point," 2019, *arXiv: 1905.12334*.
[7] U. Köster *et al.*,"Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1742–1752.
[8] J. L. Gustafson and I. T. Yonemoto,"Beating floating point at its own game: Posit arithmetic," *Supercomputing Front. Innovations*, vol. 4, no. 2, pp. 71–86, 2017.
[9] H. Zhang, J. He, and S.-B. Ko, "Efficient posit multiply-accumulate unit generator for deep learning applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.
[10] J. L. Gustafson, *The End of Error: Unum Computing*. London, U.K.: Chapman and Hall/CRC, 2017.
[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[12] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," 2016, *arXiv:1603.01025*.
[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.
[14] P. Micikevicius *et al.*, "Mixed precision training," 2017, *arXiv: 1710.03740*.
[15] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7675–7684.
[16] D. Kalamkar *et al.*, "A study of BFLOAT16 for deep learning training," 2019, *arXiv: 1905.12322*.
[17] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, "Deep positron: A deep neural network using the posit number system," 2018, *arXiv: 1812.01762*.
[18] J. Johnson, "Rethinking floating point for deep learning," 2018, *arXiv: 1811.01721*.
[19] I. Pinelis, "The exp-normal distribution is infinitely divisible," 2018, *arXiv: 1803.09838*.
[20] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," 2018, *arXiv: 1802.04680*.
[21] J. Lu *et al.*, "Training deep neural networks using posit number system," 2019, *arXiv: 1909.03831*.
[22] J. Lagarias,"Euler's constant: Euler's work and modern developments," *Bull. Amer. Math. Soc.*, vol. 50, no. 4, pp. 527–628, 2013.
[23] D. Das *et al.*, "Mixed precision training of convolutional neural networks using integer operations," 2018, *arXiv: 1802.00930*.
[24] Y. Yang, S. Wu, L. Deng, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," 2019, *arXiv: 1909.02384*.
[25] 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.
[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[29] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 558–567.
[30] S. Merity, N. S. Keskar, and R. Socher, "An analysis of neural language modeling at multiple scales," 2018, *arXiv: 1803.08240*.
[31] M. K. Jaiswal and H. K.-H. So, "PaCoGen: A hardware posit arithmetic core generator," *IEEE Access*, vol. 7, pp. 74 586–74 601, 2019.

Jinming Lu received the BS degree in microelectronics from Nankai University, Tianjin, China, in 2018. Currently he is working toward the PhD degree in information and communication engineering at Nanjing University, Nanjing, China. His current research interests include automatic speech recognition and deep learning, especially its hardware acceleration and compression algorithms.

Chao Fang received the BS degree from Tianjin University, Tianjin, China, in 2019. He is currently working toward the PhD degree at Nanjing University, Nanjing, China. His current research interests include model compression algorithm and VLSI architecture design for machine learning.

Mingyang Xu is currently working toward the BS degree in the School of Electronic Science and Engineering, Nanjing University, Nanjing, China. His current research interest includes VLSI architecture design for machine learning.



Jun Lin (Senior Member, IEEE) received the BS degree in physics and the MS degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the PhD degree in electrical engineering from Lehigh University, Bethlehem, in 2015. From 2010 to 2011, he was an ASIC design engineer with AMD. During summer 2013, he was an intern with Qualcomm Research, Bridgewater, New Jersey. In June 2015, he joined the School of Electronic Science and Engineering, Nanjing UniversityNanjing, China, where he is currently an associate professor. He is a member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the IEEE Signal Processing Society. His current research interests include low-power high-speed VLSI design, specifically VLSI design for digital signal processing and deep learning. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems, in 2008, the Best Paper Award of ISVLSI 2019. He was a recipient of the 2014 IEEE Circuits & Systems Society (CAS) Student Travel Award.

Zhongfeng Wang (Fellow, IEEE) received both BS and MS degrees from Tsinghua University, Beijing, China, and the PhD degree from the University of Minnesota, Minneapolis, in 2000. He has been working for Nanjing University, China, as a distinguished professor since 2016. Previously he has worked for Broadcom Corporation, California, from 2007 to 2016 as a leading VLSI architect. Before that, he worked for Oregon State University and National Semiconductor Corporation. He is a world-recognized expert on Low-Power High-Speed VLSI Design for Signal Processing Systems. He has published more than 200 technical papers with multiple best paper awards received from the IEEE technical societies, among which is the VLSI Transactions Best Paper Award of 2007. He has edited one book "VLSI" and held more than 20 U.S. and China patents. In the current record, he has had many papers ranking among top 25 most (annually) downloaded manuscripts in the *IEEE Trans. on VLSI Systems*. In the past, he has served as an associate editor for the *IEEE Trans. on Circuits and Systems I*, *IEEE Transactions on Circuits and Systems II*, and *IEEE Transactions on Very Large Scale Integration* for many terms. He has also served as a TPC member and various chairs for tens of international conferences. Moreover, he has contributed significantly to the industrial standards. So far, his technical proposals have been adopted by more than fifteen international networking standards. In 2015, he was elevated to the Fellow of IEEE for contributions to VLSI design and implementation of FEC coding. His current research interests are in the area of optimized VLSI design for digital communications and deep learning.