ESRU: Extremely Low-Bit and Hardware-Efficient Stochastic Rounding Unit Design for Low-Bit DNN Training

Sung-En Chang[1]∗, Geng Yuan[1]∗, Alec Lu[2]∗, Mengshu Sun[1], Yanyu Li[1], Xiaolong Ma[1], Zhengang Li[1], Yanyue Xie[1], Minghai Qin, Xue Lin[1], Zhenman Fang[2] and Yanzhi Wang[1]
[1]*Northeastern University,* [2]*Simon Fraser University*
Boston, USA, Burnaby, Canada
{chang.sun, yuan.geng, sun.meng, li.yanyu, ma.xiaol, li.zhen, xie.yany, xue.lin, yanz.wang}@notheastern.edu
{alec_lu,zhenman}@sfu.ca

***Abstract*—Stochastic rounding is crucial in the low-bit (e.g., 8-bit) training of deep neural networks (DNNs) to achieve high accuracy. One of the drawbacks of prior studies is that they require a large number of high-precision stochastic rounding units (SRUs) to guarantee low-bit DNN accuracy, which involves considerable hardware overhead. In this paper, we use extremely low-bit SRUs (ESRUs) to save a large number of hardware resources during low-bit DNN training. However, a naively designed ESRU introduces a biased distribution of random numbers, causing accuracy degradation. To address this issue, we further propose an ESRU design with a plateau-shape distribution. The plateau-shape distribution in our ESRU design is implemented with the combination of an LFSR (linear-feedback shift register) and an inverted LFSR, which avoids LFSR packing and turns an inherent LFSR drawback into an advantage in our efficient ESRU design. Experimental results using state-of-the-art DNN models demonstrate that, compared to the prior 24-bit SRU with 24-bit pseudo-random number generators (PRNG), our 8-bit ESRU with 3-bit PRNG reduces the SRU hardware resource usage by $9.75\times$ while achieving slightly higher accuracy.**

***Index Terms*—DNNs, low-bit training, stochastic rounding**

I. Introduction

DNNs have achieved extraordinary performance in various application domains, such as computer vision, speech recognition, and natural language processing. However, training DNNs needs a large number of computational resources, training time, and power consumption, which is challenging their extensive applications in the industry. To reduce the training cost for DNNs, recent studies [6], [20], [22]–[25] have tried to use low bit-width representation during DNN training, which is known as low-bit training. Low-bit training needs to quantize the weights, activations, and gradients to the low-bit fixed-point representations for both the forward and backward propagation passes during the training. Specifically, for the most challenging gradient quantization, all of these aforementioned studies have asserted that it is necessary to use stochastic rounding instead of nearest rounding during training to maintain accuracy. However, these prior studies require a large number of high bit-width random number generators in the stochastic rounding units (SRUs), which involves considerable hardware overhead. For example, as will be presented in Section V-C, even if the 24-bit SRU with the lightweight 24-bit LFSR-based PRNG is used during training, it would add an extra 23.5% LUTs usage compared to the training accelerator itself [12].

To reduce such high hardware resource overhead, we propose to use extremely low-bit (e.g., 3-bit) random numbers to approximate those high-precision random numbers in our ESRU. The intuition is that stochastic rounding in DNN training itself is a form of statistical approximation and does not necessarily need precise random numbers. However, there are two major challenges. First, naively mapping the high-precision random numbers onto the extremely low-bit ones would cause a biased distribution and thus degrade the DNN accuracy (Section III-B). While such mapping bias is hard to notice with the 8-bit approximation, it gets more significant when there are fewer bits to represent the random number in the SRU. This phenomenon is often overlooked by prior studies, as none of them has considered using extremely low-bit representations in the SRUs. Second, to efficiently generate those extremely low-bit random numbers in the ESRU on hardware, we consider using the lightweight linear feedback shift register (LFSR) [10], which is widely used as a hardware-friendly PRNG. However, using vanilla low-bit LFSR would lead to accuracy degradation because low-bit LFSR would generate the random numbers with a biased distribution (cannot generate number zero) [10].

To address these challenges, we propose an accurate and efficient ESRU design alternative with the plateau-shaped random number distribution. In the plateau-shape design, we combine an LFSR and an inverted LFSR to generate a plateau-shape distribution; the inverted LFSR is efficiently implemented by adding an inverted signal to the original LFSR instead of adding another LFSR. The such design turns the inherent LFSR limitation into an advantage in our efficient ESRU design. The detailed explanation is in Section IV.

Experimental results using state-of-the-art DNN models in image classification, super-resolution, image segmentation, and natural language processing, demonstrate that our 8-bit ESRU design with 3-bit PRNG can achieve a negligible accuracy drop in the DNN training compared to the floating-point based models. Compared to prior SRU designs with 8-bit, 16-bit, and 24-bit LFSR-based PRNG, our designs reduce the SRU hardware resource usage by $3.75\times$ to $9.75\times$, while achieving a slightly higher model accuracy.

In summary, this paper makes the following contributions:

∗Equal contribution.

1) The first work explores the extremely low-bit (i.e., 3-bit) representation in the SRUs for the low-bit (i.e., 8-bit) DNN training.
2) An in-depth analysis of limitations in existing SRU designs.
3) An accurate and hardware-efficient ESRU design for low-bit DNN training, which approximates high-precision random numbers using extremely low-bit (3-bit) random numbers in the plateau-shape distribution.
4) Experiments to demonstrate the significant hardware resource savings ($3.75\times$ to $9.75\times$) and superior accuracy of our 8-bit training with ESRU over state-of-the-art low-bit training frameworks.

II. Background and Related Work

A. Nearest Rounding vs. Stochastic Rounding

Nearest rounding [9] is the most common rounding scheme for quantization in low-bit training. Take a fixed-pointed quantization (FQ) as an example, the equation of the FQ is:

$$FQ(x, m) = 2^{1-m} \times rounding_n(x/2^{1-m}), \quad (1)$$

where x is the floating-point input, m is the quantization bit-width, and 2^{1-m} is the distance between each quantization level. The nearest rounding scheme $rounding_n[.]$ would always round the floating-point input to the nearest quantization level.

Another scheme is stochastic rounding [8], [6]. Rather than always rounding the floating-point input to the nearest quantization level, stochastic rounding rounds the input as follows:

$$rounding_s(x) = \begin{cases} \lfloor x \rfloor, & \text{w.p.} \quad 1 - (x - \lfloor x \rfloor) \\ \lceil x \rceil, & \text{w.p.} \quad x - \lfloor x \rfloor \end{cases}, \quad (2)$$

where "w.p." stands for "with probability".

B. Implementation of Stochastic Rounding

In order to apply the stochastic rounding during the low-bit training, DoReFa-Net [24] proposed that an efficient way is to add a random number $r \in Uniform(-0.5, 0.5)$ to the inputs x before each rounding step, i.e.,

$$\begin{aligned} rounding_s(x) &= rounding_n(x + r), \\ r &\in Uniform(-0.5, 0.5). \end{aligned} \quad (3)$$

However, in Equation (3), the control logic is still needed to compare the fraction value with 0.5 in the rounding step (e.g., $X.49$ will be rounded to X because $0.49 < 0.5$), which introduces extra overhead. To eliminate this control logic, [13] found that the stochastic rounding can be implemented by adding x with a random number $r \in Uniform(0, 1)$ and then dropping the fraction bits. As a result, the carry-over to the integer bits automatically completes the rounding. Figure 1 shows an example to implement stochastic rounding (from 16-bit to 8-bit) in this more efficient way.

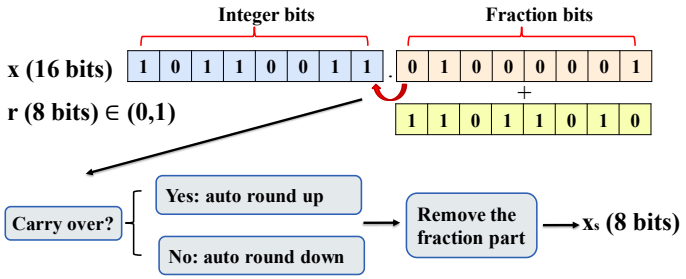

Fig. 1. The implementation of stochastic rounding. x: the gradient, r: random number, x_s: the gradient after the stochastic rounding process.

C. Stochastic Rounding in Low-Bit Training

In low-bit DNN training, there are three main components that need to be quantized: weight, activation, and gradient. For the weight and the activation quantization, applying nearest rounding works very well. However, for the gradient quantization, Höhfeld and Fahlman [8] found that most of the gradients would be rounded to zero if they simply used the nearest rounding scheme, because the magnitudes of most of the gradients are relatively small.

Thus, using stochastic rounding for gradient quantization in low-bit training has been explored in the past few years. First, Gupta et al. [6] introduced stochastic rounding on the gradient quantization of the DNNs in low-bit training. Further, DoReFa-Net [24] applied the stochastic rounding on the low precision gradient with the large-scale dataset (ImageNet). Then, WAGE [20] and WAGEUBN [22] used stochastic rounding to quantize the gradient to an 8-bit integer. However, both WAGE and WAGEUBN suffer from non-negligible accuracy degradation. Recently, FP8 [18], Uint8 [25], and ADint8 [23] quantized the gradient to an 8-bit integer with stochastic rounding and achieved comparable accuracy to the 32-bit floating-point training.

All of these works only mention that they used stochastic rounding during the low-bit training process. However, they overlooked that the stochastic rounding needs to use a large amount of high-precision random numbers (r in Equation (3)), which introduces considerable hardware resource overhead in the low-bit DNN training.

D. Low-Bit Training on Hardware Accelerators

While 32-bit floating-point precision is desirable during DNN training to achieve high prediction accuracy, it leads to high resource usage in hardware accelerator designs. Previous studies have explored various approaches to lower the resource demand in order to bring higher computation parallelism for training speedups while maintaining a similar accuracy as the 32-bit floating-point models [3], [6], [14], [15], [18]. Initially, in [3], a hardware accelerator for DNN training was implemented with 32-bit fixed-point to bring significant resource saving. Later on, more works were able to effectively reduce the bit-width of the training data without a significant accuracy loss, so long as stochastic rounding was applied. For example, an ASIC implementation of low-bit RNN training with 24-bit fixed-point numbers was proposed in [14], which employed a 48-bit input SRU design with a 24-bit LFSR to realize

stochastic rounding. In [6], low-bit training with 16-bit fixed-point numbers was implemented while achieving a similar accuracy as the 32-bit floating-point training. Their design used a 48-bit input SRU with a 16-bit LFSR PRNG. Some recent studies even explored 12-bit and 8-bit floating-point numbers with stochastic rounding in their hardware training accelerators and have achieved decent accuracy [16], [18].

However, in these previous designs, the resource utilization of the stochastic rounding units with their random number generator designs typically scales with the precision of the training data. We overcome this resource scaling constraint in our proposed ESRU design to bring further resource saving by exploring a hardware-efficient low-bit random number generator during stochastic rounding.

III. Limitations of Existing SRU Designs

A. Issue with Default High Bit-Width SRUs

In order to apply stochastic rounding in low-bit DNN training, DoReFa-Net [24] and [13] proposed that an efficient way is to add the random numbers on the gradients before each rounding step. However, the number of gradients is relatively large in state-of-the-art DNN models. For example, there are around 10^7 weights in the widely used ResNet-50. So in the backpropagation process, there are around 10^7 gradients to update the corresponding weights. In each iteration, it needs to use around 10^3 high bit-width SRUs (assuming there are 10^3 parallel computations in the hardware), which introduce a large amount of extra resource usage for the hardware accelerators.

B. Distribution Bias with Extremely Low Bit-Width SRUs

To reduce the extra resource usage during the DNNs training, an intuitive way is to use low-bit representations for the random numbers in the SRUs. Suppose we have a 32-bit floating-point random number r and we need to map it to the m-bit random number r', where m is extremely small (e.g., $m = 3$). The basic idea is to divide all the 32-bit random number distributions into $n = 2^m$ levels and use nearest rounding to round those numbers into each level l_i, the m-bit representation of the i-th level random number.

In a straightforward mapping, one may set a uniform distance between every two consecutive levels (i.e., $\epsilon = l_{i+1} - l_i = 1/n$) and set $l_0 = 0$. As a result, the floating-point random number r is approximated as a m-bit r' using the following equation:

$$r' = \begin{cases} l_0 & \text{if} \quad r \in [l_0, l_0 + \frac{\epsilon}{2}) \\ l_1 & \text{if} \quad r \in [l_1 - \frac{\epsilon}{2}, l_1 + \frac{\epsilon}{2}) \\ ... & \\ l_{n-1} & \text{if} \quad r \in [l_{n-1} - \frac{\epsilon}{2}, l_{n-1} + \frac{\epsilon}{2}) \end{cases} \tag{4}$$

Unfortunately, such a naive mapping leads to a biased distribution of the random numbers. To better demonstrate this, we have mapped the 32-bit random numbers onto 8 levels using 3-bit representations based on Equation (4) and visualized such random number distribution in Figure 3. The x-axis shows the values of l_0 to l_7 and their corresponding binary representation in the fraction part; the y-axis shows the number of 32-bit random numbers distributed in each range. As shown in Figure 3, the range of the first level (i.e., the green part) is only half of the other levels. Moreover, the last range (i.e., the orange part) does not belong to any levels.

Moreover, Figure 2 visualizes such bias when mapping to 8-bit, 4-bit, and 3-bit representations. While it is hard to notice such bias using 8-bit representations in the SRUs, the bias is much more significant when we use fewer bits (e.g., 3-bit) to represent the random numbers in the SRUs. As will be presented in Section V-B1, such a biased distribution would lead to about 2% DNN accuracy loss when the random numbers are approximated with 3-bit representations. This bias is often overlooked in the existing studies, as none of them has used extremely low-bit representations in the SRUs.

C. Lightweight LFSR and Its Limitations

A lightweight SRU alternative is to use the linear feedback shift register (LFSR) [10], which is widely used as pseudo-random number generators (PRNG), due to its small logic resource footprint and low latency design [1], [10].

For every iteration, an LFSR updates its random sequence by propagating the shift register values and feeding the XOR'ed value from certain shift registers of the LFSR based on a pre-defined mask to its first shift register. As such, a m-bit LFSR-based PRNG will generate all $2^m - 1$ patterns (without zero) in a random sequence before repeating the same random sequence. However, simply using vanilla LFSRs to generate the random numbers in the lightweight SRUs comes up with a shortcoming: Roth et al. [10] found that a vanilla LFSR cannot have 0 as its seed (starting point) and cannot generate 0 in its output. Thus, its output distribution is biased and would cause accuracy degradation in low-bit DNN training.

IV. Our Proposed ESRU Design

To design an accurate and hardware-efficient SRU for low-bit DNN training, we consider using extremely low-bit representations (e.g., 3-bit) for the random numbers in the SRU and generate such random numbers using our optimized lightweight LFSR variants. To address the issue of biased distribution and LFSR limitations presented in Section III, we propose an ESRU design with the plateau-shape distribution.

A. Idea of Plateau-shape Distribution

The key idea to solve the biased distribution from the naive mapping is to keep $l_0 = 0$ and adjust $\epsilon = l_{i+1} - l_i = 1/(n-1) = 1/(2^m - 1)$, where l_i is the m-bit representation of the i-th level random number and ϵ is the distance between every two consecutive levels. As a result, we can cover the whole distribution from 0 to 1 without missing any value.

Figure 4(b) shows an example with 3-bit representation (i.e., $m = 3$), by slightly increasing ϵ from $1/8$ (in the naive mapping, shown in Figure 4(a)) to $1/7$ to cover the whole distribution from 0 to 1. This mapping will lead to a non-uniform plateau-shape distribution as will be explained below.

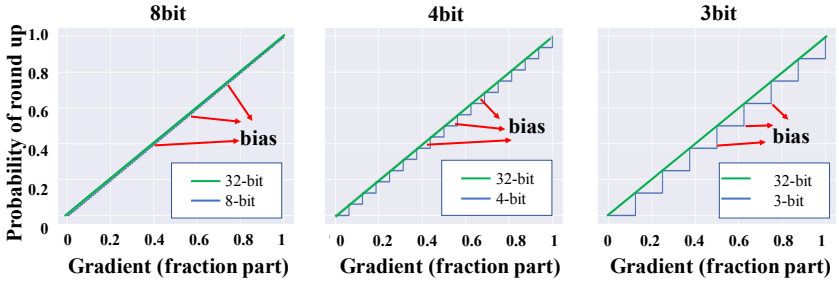

Fig. 2. Biased distribution of the naive mapping from floating-point random numbers to low-bit ones in the SRU. The blue lines are the low-bit SRU design and the blue lines are always "below" the green line (32-bit baseline), which introduce the bias.

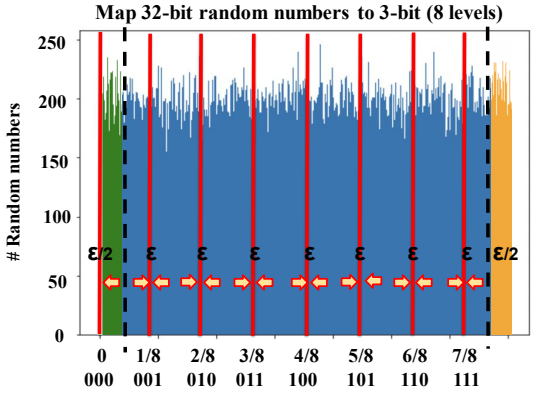

Fig. 3. The naive mapping from 32-bit random numbers onto 3-bit ones with the biased distribution.

Specifically, in this mapping where $l_0 = 0$ and $\epsilon = 1/(2^m - 1)$, the 32-bit floating point random number r can be mapped to the m-bit (e.g., $m = 3$) number r' using the following equation:

$$r' = \begin{cases} l_0 & \text{if} \quad r \in [l_0, l_0 + \frac{\epsilon}{2}) \\ l_1 & \text{if} \quad r \in [l_1 - \frac{\epsilon}{2}, l_1 + \frac{\epsilon}{2}) \\ \dots & \\ l_{n-1} & \text{if} \quad r \in [l_{n-1} - \frac{\epsilon}{2}, l_{n-1}) \end{cases} \tag{5}$$

Based on Equation (5), we can have the mapping range in the first level l_0 and the last level l_{n-1} as $\frac{\epsilon}{2}$, which is only half of the range in the middle levels l_1, l_2..., l_{n-1}. Take a 3-bit (i.e., $m = 3$) representation of the low-bit random numbers as an example, we need to map the 32-bit random numbers to $2^3 = 8$ levels. Figure 4(b) shows that when we divide the 32-bit random number distributions $Uniform(0, 1)$ into 8 levels, the mapping range in the first and the last levels (i.e. the green part) is only half of that for the middle levels. Therefore, the plateau-shape distribution of 3-bit random numbers is observed in Figure 4(c), which requires special consideration when designing its hardware architecture.

B. Hardware Design of Plateau-shape based ESRU

We propose a hardware-aware technique to generate the random numbers in the SRU by considering both the limitations of LFSR and matching the plateau-shape distribution. Figure 5(a) shows the overall design architecture of our ESRU. It applies stochastic rounding to an intermediate MAC (multiply–accumulate) data from backpropagation calculation by first adding a random number, generated by an LFSR-based PRNG, to the fraction bits. Then it crops the addition result to derive the final gradient result. For a training model with error data and model parameters, both represented in 8-bit fixed-point, the intermediate MAC data during gradient calculation is then represented in 16-bit fixed-point representation, for such, we show a 16-bit adder used in our ESRU design.

Optimization 1: lookup table-based mapping vs. shift-based mapping. Although random number generation logic is rather simple, we find that implementing a resource-efficient design is nontrivial and could lead to a 3.75x resource (in terms of LUTs on an FPGA) usage difference as we observe between two LFSR design alternatives. Both design variants include a 3-bit LFSR as shown in figure 5(a), which will populate $2^3 - 1 = 7$ patterns (without zero) in a random sequence before repeating the same sequence as explained in Section III-C. However, for mapping our low-bit LFSR number to a higher bit-width, the first design variant uses a look-up table, implemented with LUTs, to keep a record of the corresponding high bit-width mapping value (i.e., l_i in Equation (5)) of our 3-bit random number. Our optimized approach, shown in figure 5(b) uses a simple shift operator to align the 3-bit random number with the most significant bit in the fraction bits of the high bit-width number to proportionally scale the random number.

Optimization2: LFSR + inverted LFSR to generate the plateau distribution. To generate the plateau distribution of random numbers in low-bit representations described in Section IV-A, we also add an inverted signal to select the bit-wise inverted random number. This feature effectively addresses two key issues. First, it allows our PRNG to populate zero during its sequence generation by setting the invert signal to high when the LFSR generates the random number with all ones. Second, when sequentially used with the non-inverted PRNG sequence (i.e., random sequence repeats after enumerating all non-inverted and inverted sequences), this produces random numbers in the correct random number distribution as shown in Figure 5(c). By using this technique, we successfully take advantage of the characteristic of the LFSR variant in ESRU to generate the plateau-shape distribution.

V. Experiment Results

A. Experiments Setup

We evaluate the low-bit DNN training with 8-bit integers for activations/weights/gradients and our hardware-efficient ESRU design for a wide range of applications, including image classification, super resolution, segmentation, and natural language

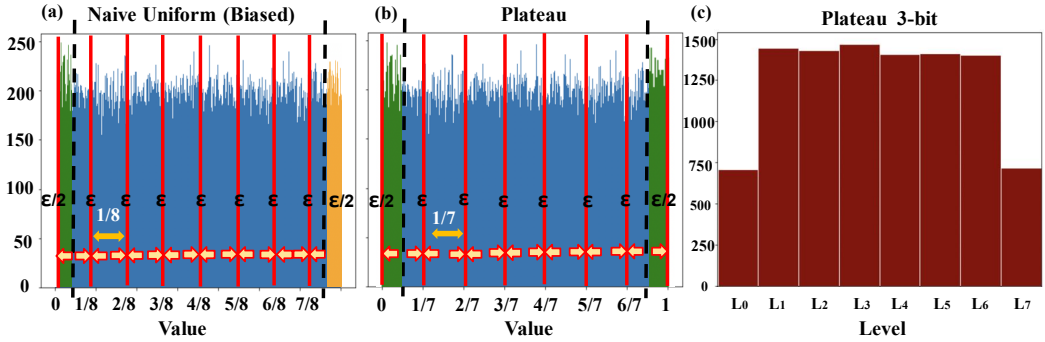

Fig. 4. Comparison between the naive mapping and our Plateau-shape distribution.

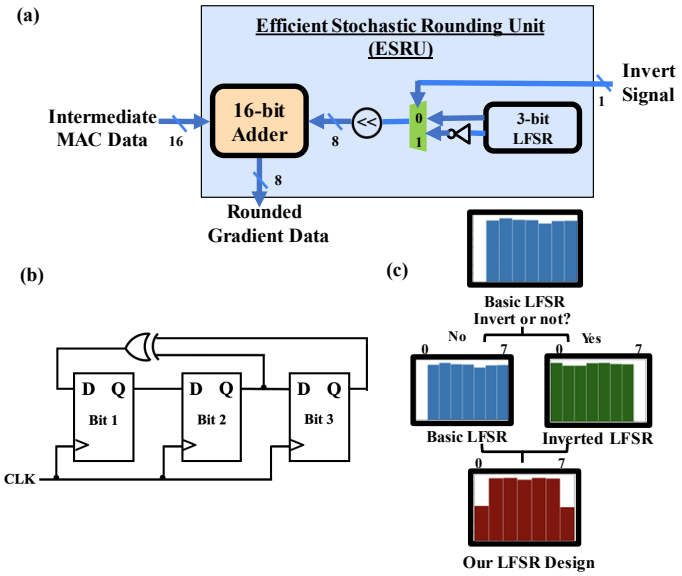

Fig. 5. Hardware architecture of our plateau-shape based ESRU.

processing (NLP). We describe the detailed setup of each application in Section V-B. All the models with the baseline 32-bit floating-point training and the 8-bit training with our ESRU design are conducted on NVIDIA TITAN RTX GPUs, with CUDA 11.2 and PyTorch 1.8 frameworks running on the Ubuntu 18.04 OS. The 8-bit training with our ESRU design utilizes the same data augmentation techniques as those used in the baseline 32-bit floating-point training.

B. Accuracy Results

1) Accuracy Results for Image Classification: For image classification tasks, the evaluated models include ResNet-18 and ResNet-50 [7] on ImageNet [11] dataset.

2) Accuracy Results for Image Segmentation: For image segmentation tasks, we evaluate the DeeplabV3 and DeeplabV3plus [2] models on the Pascal VOC2012 dataset [5]. The initial learning is 0.1 and the models are trained for 100 epochs. Our plateau-shape based ESRU has similar accuracy (i.e., $-0.0070 \sim +0.0021$ mIoU) to the floating point training.

3) Accuracy Results for NLP: For NLP tasks with the BERT [4] model, we evaluate on a variety of datasets from the General Language Understanding Evaluation (GLUE) [17] benchmark. The pre-trained BERT models are from HuggingFace Transformer [19]. Quantization and finetuning are simultaneously performed for 3 epochs with the initial learning rate of 2×10^{-5}. Table III shows that our 8-bit training with plateau-shape based ESRU only has 0.38 average point degradation across different evaluation metrics on the corresponding tasks.

TABLE I
ACCURACY COMPARISON WITH EXISTING WORKS USING RESNET-18 AND RESNET-50 MODELS ON IMAGENET. W: WEIGHT, A: ACTIVATION, G: GRADIENT.

Model	Method	Precision (W/A/G)	Low-bit LFSR?	Accuracy
ResNet-18	FP	32bit	-	71.10
	WAGEUBN [22]	8bit	×	67.40
	FP8 [18]	8bit	×	67.34
	Uint8 [25]	8bit	×	69.67
	ADint8 [23]	8bit	×	70.21
	Naive Mapping	8bit	✓	69.07
	Ours(Plateau)	8bit	✓	**70.91**
ResNet-50	FP	32bit	-	77.59
	WAGEUBN [22]	8bit	×	69.07
	FP8 [18]	8bit	×	76.20
	Uint8 [25]	8bit	×	76.34
	ADint8 [23]	8bit	×	76.59
	Naive Mapping	8bit	✓	76.03
	Ours(Plateau)	8bit	✓	**77.56**

TABLE II
COMPARISON OF OUR LOW-BIT TRAINING WITH ESRUS AND FLOATING-POINT TRAINING FOR IMAGE SEGMENTATION ON THE VOC2012 DATASET. MIOU: MEAN INTERSECTION OVER UNION, THE HIGHER IS BETTER.

Model	Backbone	Method	precision (W/A/G/R)	mIoU
DeepLabV3	ResNet-50	FP	32/32/32/-	0.7604
		Ours(Plateau)	8/8/8/3	**0.7619**
DeepLabV3Plus	MobileNetV2	FP	32/32/32/-	0.7110
		Ours(Plateau)	8/8/8/3	**0.7040**
DeepLabV3Plus	ResNet-50	FP	32/32/32/-	0.7649
		Ours(Plateau)	8/8/8/3	**0.7670**

C. Hardware Results

To gain a better perspective of the hardware resource and latency impact from the SRUs, we use the FPGA training accelerator design (called DarkFPGA) proposed in [12] as a reference design. In DarkFPGA, batch-level parallelism of $T_B = 32$ and an image-level parallelism of $T_I = 128$ are employed and it computes 4096 parallel gradients on a single Xilinx Ultrascale+ VU9P FPGA. Thus, to enable stochastic rounding in DarkFPGA, we instantiate 4096 stochastic rounding units and compare the resource usage when using our ESRU designs with 3-bit and 8-bit plateau-shape distribution-based LFSRs, as well as previous state-of-the-art SRU designs that use the basic 24-bit, 16-bit, and 8-bit LFSRs. We use Xilinx

TABLE III

COMPARISON OF OUR LOW-BIT TRAINING AND FLOATING-POINT TRAINING FOR NATURAL LANGUAGE PROCESSING WITH BERT. THE EVALUATION METRICS INCLUDE F1, PEARSON, ACCURACY, AND MATTHEWS CORRELATION, THE HIGHER VALUE IS BETTER.

Method	Precision(W/A/G/R)	MRPC	STS-B	RTE	COLA	MNLI	QQP	SST2	QNLI	Avg.
FP	32/32/32/-	89.66	89.19	66.43	57.27	84.37	91.18	92.66	91.40	82.77
Ours(Plateau)	8/8/8/3	88.81	88.80	65.70	57.04	84.35	90.66	92.65	91.14	82.39

TABLE IV

COMPARISON OF RESOURCE UTILIZATION FOR DIFFERENT LFSR DESIGNS IN OUR ESRU ON XILINX VU9P FPGA AND DARKFPGA [12] DESIGN. DARKFPGA [12] DESIGN IS THE 8-BIT FPGA TRAINING FRAMEWORK. BASIC: THE CONVENTIONAL LFSR DESIGN.

Method	LFSR Bit-width	ESRU LUTs	ESRU FF	Total LUTs in VU9P	ESRU LUTs / VU9P LUTs	LUTs Usage in DarkFPGA [12]	ESRU LUTs / DarkFPGA [12] LUTs
Naive Mapping	32bit	217,088	270,336	1,182,240	18.4%	678,716	32.0%
Naive Mapping	24bit	159,744	204,800	1,182,240	13.5%	678,716	23.5%
Naive Mapping	16bit	110,592	139,264	1,182,240	9.4%	678,716	16.3%
Naive Mapping	8bit	61,440	73,728	1,182,240	5.2%	678,716	9.0%
Ours(Plateau)	8bit	61,440	73,728	1,182,240	5.2%	678,716	9.0%
Ours(Plateau)	3bit	**16,384**	**32,768**	1,182,240	**1.4%**	678,716	**2.4%**

Vitis 2020.1 [21] to evaluate the post place-and-route resource utilization, frequency, and latency of the designs. Note that LFSR only consumes LUT and FF resources on an FPGA. As shown in Table IV, in terms of resource usage for SRU designs, compared with SRU designs with 24-bit, 16-bit, and 8-bit LFSRs, our ERSU design with 3-bit LFSR brings $9.7\times$, $6.75\times$, and $3.75\times$ LUT savings. With respect to the entire hardware training accelerator in [12], our ERSU with 3-bit LFSR only needs 2.4% extra LUT usage, while a naive design may introduce up to 32.0% LUT overhead.

VI. CONCLUSION

In this paper, we are the first to investigate hardware-efficient stochastic rounding unit (ESRU) designs by using extremely low-bit (i.e., 3-bit) random number generation for low-bit (i.e., 8-bit) DNN training. We observed that naively using low-bit representations to approximate high-precision random numbers leads to a biased distribution, causing accuracy degradation. Thus, we proposed a new method to approximate the random numbers using low-bit representation in a plateau-shape distribution. Based on the plateau-shape distributions, we designed a hardware-efficient ESRU with the optimized LFSR variants to generate these low-bit (i.e., 3-bit) random numbers. Experimental results using a wide range of DNN applications demonstrated that our 8-bit DNN training with the optimized ESRU achieved superior accuracy than state-of-the-art 8-bit training frameworks, with a negligible accuracy drop compared to the floating-point baseline training. Moreover, compared to the prior 24-bit SRU with 24-bit PRNG and 16-bit SRU with 16-bit PRNG, our 8-bit ESRU with 3-bit PRNG reduced the SRU resource usage by $9.75\times$ and $6.75\times$, respectively.

ACKNOWLEDGMENT

This work was partly supported by NSF CCF-1901378, NSF CCF-1919117 and CCF-1937500; NSERC Discovery Grant RGPIN-2019-04613, DGECR-2019-00120, Alliance Grant ALLRP-552042-2020; CFI John R. Evans Leaders Fund.

REFERENCES

[1] François Arnault et al. Revisiting lfsrs for cryptographic applications. *IEEE Transactions on Information Theory*, 57(12):8095–8113, 2011.

[2] Liang-Chieh Chen et al. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, pages 801–818, 2018.

[3] Yunji Chen et al. Dadiannao: A machine-learning supercomputer. In *2014 IEEE/ACM Int'l. Symp. on Microarchitecture*, pages 609–622, 2014.

[4] Jacob Devlin et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, June 2019.

[5] M. Everingham et al. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[6] Suyog Gupta et al. Deep learning with limited numerical precision. In *Int'l. conf. on machine learning*, pages 1737–1746. PMLR, 2015.

[7] Kaiming He et al. Deep residual learning for image recognition. In *Proceedings of CVPR*, pages 770–778, 2016.

[8] Markus Höhfeld et al. Probabilistic rounding in neural network learning with limited precision. *Neurocomputing*, 4(6):291–299, 1992.

[9] IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[10] Roth Jr et al. *Digital systems design using VHDL*. Cengage Learning, 2016.

[11] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. *NIPS*, 25, 2012.

[12] Cheng Luo et al. Towards efficient deep neural network training by fpga-based batch-level parallelism. *Journal of Semiconductors*, 41(2):022403, 2020.

[13] Mantas Mikaitis. Stochastic rounding: Algorithms and hardware accelerator. In *IJCNN*, pages 1–6. IEEE, 2021.

[14] Taesik Na et al. On-chip training of recurrent neural networks with limited numerical precision. In *IJCNN*, pages 3716–3723. IEEE, 2017.

[15] Marc Ortiz et al. Low-precision floating-point schemes for neural network training. *arXiv preprint arXiv:1804.05267*, 2018.

[16] Marc Ortiz et al. Low-precision floating-point schemes for neural network training. *CoRR*, abs/1804.05267, 2018.

[17] Alex Wang et al. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *EMNLP Workshop*, 2018.

[18] Naigang Wang et al. Training deep neural networks with 8-bit floating point numbers. *NIPS*, 31, 2018.

[19] Thomas Wolf et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[20] Shuang Wu et al. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[21] Xilinx. Vitis unified software platform. https://docs.xilinx.com/v/u/2020.1-English/ug1393-vitis-application-acceleration, 2021. Last accessed November 17, 2021.

[22] Yukuan Yang et al. Training high-performance and large-scale deep neural networks with full 8-bit integers. *Neural Networks*, 125:70–82, 2020.

[23] Kang Zhao et al. Distribution adaptive int8 quantization for training cnns. In *Proceedings of AAAI*, 2021.

[24] Shuchang Zhou et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[25] Feng Zhu et al. Towards unified int8 training for convolutional neural network. In *Proceedings of CVPR*, pages 1969–1979, 2020.