LNS-Madam: Low-Precision Training in Logarithmic Number System using Multiplicative Weight Update

Jiawei Zhao, Steve Dai, Rangharajan Venkatesan, Brian Zimmer, Mustafa Ali, Ming-Yu Liu, Brucek Khailany, William J. Dally, Anima Anandkumar

Abstract—Representing deep neural networks (DNNs) in low-precision is a promising approach to enable efficient acceleration and memory reduction. Previous methods that train DNNs in low-precision typically keep a copy of weights in high-precision during the weight updates. Directly training with low-precision weights leads to accuracy degradation due to complex interactions between the low-precision number systems and the learning algorithms. To address this issue, we develop a co-designed low-precision training framework, termed LNS-Madam, in which we jointly design a logarithmic number system (LNS) and a multiplicative weight update algorithm (Madam). We prove that LNS-Madam results in low quantization error during weight updates, leading to stable performance even if the precision is limited. We further propose a hardware design of LNS-Madam that resolves practical challenges in implementing an efficient datapath for LNS computations. Our implementation effectively reduces energy overhead incurred by LNS-to-integer conversion and partial sum accumulation. Experimental results show that LNS-Madam achieves comparable accuracy to full-precision counterparts with only 8 bits on popular computer vision and natural language tasks. Compared to FP32 and FP8, LNS-Madam reduces the energy consumption by over 90% and 55%, respectively.

1 Introduction

Deep neural networks (DNNs) have shown impressive performance in many applications, including image classification and language processing. However, training and deploying DNNs typically incurs significant computation and energy costs. Traditionally, values in neural networks are represented using floating-point (32-bit) numbers, which incurs a large arithmetic and memory footprint, and hence significant energy consumption. However, recent studies suggest that high-precision number formats are redundant, and models can be quantized in low-precision with little loss in accuracy [1], [2]. Low-precision numbers only require low-bitwidth computational units, leading to better computational efficiency and less required memory bandwidth and capacity.

While low-precision training methods generally reduce computational costs, energy efficiency can be further improved by choosing a logarithmic number system (LNS) for representing numbers. LNS achieves a higher computational efficiency by transforming expensive multiplication operations in the network layers to inexpensive additions in their logarithmic representations. In addition, it attains a wide dynamic range and can provide a good approximation of the non-uniform weight distribution in neural networks. Thus LNS is an excellent candidate for training DNNs in low-precision [3], [4], [5].

Although previous studies demonstrate that it is feasible to train networks in low-precision using LNS, these approaches have not yet shown promising results on larger datasets and state-of-the-art models [5], [6]. Standard LNS fixes the base of the logarithm, termed log-base, to be precisely two. However, a more flexible log-base is needed since the numbers in DNNs require different quantization gaps during training [7]. A flexible log-base can introduce additional hardware overhead due to expensive conversion operations between the logarithmic and integer (linear) formats. This motivates us to design a LNS that has a flexible choice of the log-base while maximizing the efficiency of LNS-to-integer conversions.

Conventional low-precision training methods typically require high-precision copies of weights and gradients during weight update to maintain optimization stability. In fact, most recent studies even use a full-precision (FP32) copy of weights [5], [6]. This introduces additional energy costs and expensive FP32 arithmetic, which becomes prohibitive especially in energy-constrained edge devices.

The high-precision requirement for weight updates is due to complex interactions between learning algorithms and number systems, which has usually been ignored in previous studies. For example, as illustrated in Fig. 1, updates generated by stochastic gradient descent (SGD) are disregarded more frequently by LNS when the weights become larger. This is because the quantization gap grows exponentially as the weights become larger in LNS, which suggests that conventional learning algorithms may not be suitable for LNS. Hence, in previous studies, high-precision weight copies are required to avoid numerical instabilities [5], [7].

To directly update the weights in low-precision, we employ a learning algorithm tailored to LNS. Recently,

- *J. Zhao and A. Anandkumar are with Caltech. E-mail: jiawei@caltech.edu*
- *S. Dai, R. Venkatesan, B. Zimmer, M. Liu, B. Khailany, B. Dally, A. Anandkumar are with NVIDIA.*
- *M. Ali is with Purdue University.*

Manuscript received.

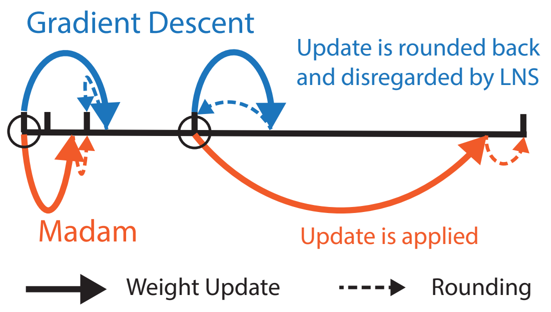


Fig. 1. Illustration for updating weights using Gradient Descent (GD) and Madam under logarithmic representation. Each coordinate represents a number stored in LNS. Assume the weights at two circles receive the same gradient. The updates generated from GD are disregarded as the weights move larger, whereas the updates generated by Madam are adjusted with the weights.

Bernstein et al. [8] proposed the Madam optimizer based on multiplicative updates, which is equivalent to updating weights additively in the logarithmic space. As illustrated in Fig. 1, Madam generates larger magnitudes of the updates when the weights become larger, making it more suitable for a logarithmic weight representation. However, Bernstein et al. [8] still employ full-precision training with Madam without considering low-precision LNS.

In this work, we propose a co-designed low-precision training framework called LNS-Madam in which we adopt LNS (with a more flexible log-base) for representing DNNs and apply a modified Madam (tailored to LNS) to train these networks. LNS-Madam reduces the precision requirements for all components of the training, including *forward and backward propagation, as well as weight updates*.

Our contributions are summarized as follows:

1) We design a multi-base LNS where the log-base can be fractional powers of two. The multi-base LNS accommodates the precision and range requirements of the training dynamics while being hardware-friendly. In addition, we propose an approximation for the addition arithmetic in LNS to further improve its energy efficiency.
2) We propose an efficient hardware implementation of LNS-Madam that addresses challenges in designing an efficient datapath for LNS computations, including accumulation and conversion between logarithmic and integer formats. We leverage this implementation to study the energy benefits of training in LNS.
3) To achieve low-precision weight updates in LNS, we replace standard SGD or Adam optimizers with our proposed optimizer based on Madam, which directly updates the weights in the LNS. Through theoretical analysis and empirical evaluations, we show that the proposed Madam optimizer achieves significantly lower quantization error in LNS.
4) In our experiments, LNS-Madam achieves full-precision accuracy with 8 bit representations on popular computer vision and natural language tasks while reducing energy consumption by over 90%. The energy efficiency results for training different models with various number formats are summarized in Fig. 2.

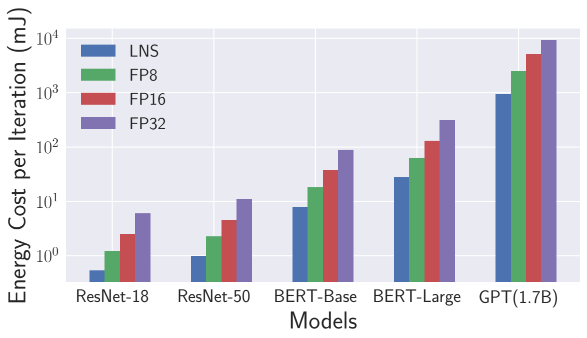


Fig. 2. Energy efficiency for training different models with various number formats. The per-iteration energy consumption (mJ) is listed.

2 Hardware-friendly Multi-Base Logarithmic Number System

In this section, we introduce our multi-base logarithmic number system (LNS), including the corresponding number representation and arithmetic operations.

We start with the mathematical formulation that we will use throughout this paper. We assume that the DNN $F(\cdot, W)$ is composed of L layers with learnable weights W and input activations X across the layers. $\mathcal{L}(W)$ denotes the training loss. The forward propagation is defined as: $X_l = f_l(X_{l-1}, W_l)$, where $l \in [1, L]$ denotes layer l. $\nabla_{X_l} = \frac{\partial \mathcal{L}(W)}{\partial X_l}$ and $\nabla_{W_l} = \frac{\partial \mathcal{L}(W)}{\partial W_l}$ denote gradients with respect to input activations and weights, respectively, at layer l. For a number system, we define $\mathcal{B}$ as the bitwidth, x as a number, and x^q as the number in quantized format.

2.1 Multi-base Logarithmic Representation

Unlike prior work that uses exactly 2 as the base of the logarithmic representation, we propose a multi-base logarithmic representation that allows the base to be two with a fractional exponent, such that:

$$x = \text{sign} \times 2^{\tilde{x}/\gamma} \quad \tilde{x} = 0, 1, 2, ..., 2^{\mathcal{B}-1} - 1,$$

where $\tilde{x}$ is an integer and γ is the base factor that controls the fractional exponent of the base. γ controls the quantization gap, which is the distance between successive representable values within the number system. Previous work has already demonstrated that logarithmic quantized neural networks achieve better performance when relaxing γ from 1 to 2 [5]. We find that further relaxation can help adapt to different models and datasets. Therefore we generalize the base factor setting, enabling more flexibility in controlling the quantization gap in order to more accurately approximate the training dynamics. In addition, we specially restrict γ to be powers of 2 for hardware efficiency, as described later.

2.2 Arithmetic Operations in LNS

One of the benefits of using LNS stems from the low computational cost of its arithmetic operations. We use dot product operations as an example since they are prevalent during training. Consider two vectors $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n$ that are represented by their integer exponents $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ in LNS. A dot product operation between them can be represented as follows:

$$\begin{aligned}\boldsymbol{a}^T\boldsymbol{b} &= \sum_{i=1}^{n} \mathrm{sign}_i \times 2^{\tilde{\boldsymbol{a}_i}/\gamma} \times 2^{\tilde{\boldsymbol{b}_i}/\gamma} \\ &= \sum_{i=1}^{n} \mathrm{sign}_i \times 2^{(\tilde{\boldsymbol{a}_i}+\tilde{\boldsymbol{b}_i})/\gamma} \\ &= \sum_{i=1}^{n} \mathrm{sign}_i \times 2^{\tilde{p_i}/\gamma},\end{aligned} \tag{1}$$

where $\mathrm{sign}_i = sign(\boldsymbol{a}_i) \oplus sign(\boldsymbol{b}_i)$. In this dot product operation, each element-wise multiplication is computed as an addition between integer exponents, which significantly reduces the computational cost by requiring adders instead of multipliers.

While the multiplications are easy to compute in LNS, the accumulation is difficult to compute efficiently as it requires first converting from logarithmic to integer format and then performing the addition operation. The conversion between these formats is generally expensive as it requires computing $2^{\tilde{p_i}/\gamma}$ using polynomial expansion. To overcome this challenge, we decompose the exponent $2^{\tilde{p_i}/\gamma}$ into a quotient component $\tilde{p_i}_q$ and and a remainder component $\tilde{p_i}_r$ as follows:

$$2^{\tilde{p_i}/\gamma} = 2^{\tilde{p_i}_q + \tilde{p_i}_r/\gamma} = 2^{\tilde{p_i}_q} \cdot 2^{\tilde{p_i}_r/\gamma}. \tag{2}$$

With this decomposition, converting from LNS to integer requires only a table lookup for $2^{\tilde{p_i}_r/\gamma}$ followed by a shift for $2^{\tilde{p_i}_q}$. For the table lookup, we simply maintain γ constants $2^{i/\gamma}\ \forall i \in \{0, 1, ..., \gamma - 1\}$ and select the constant based on the remainder $\tilde{p_i}_r$. Note that because γ is restricted to be power of 2, the remainder can be efficiently extracted from the least-significant bits (LSB) of the exponent while the quotient can be extracted from the most-significant bits (MSB). Typically the lookup table (LUT) requires $2^{\mathcal{B}}$ entries for storing all possible values.

2.3 Conversion Approximation

In addition to the exact conversion technique discussed above, we can further reduce the cost of the LNS-to-integer conversion using a hybrid approximation method. Our method is based on Mitchell approximation [9]: $2^{\tilde{x}/\gamma} \approx (1 + \tilde{x}/\gamma)$, where the logarithmic format can be efficiently approximated to the integer format when $\tilde{x}/\gamma$ is small. Specifically, we further split the remainder into a LSB and a MSB component. The value of the LSB is approximated using Mitchell approximation, and the value of the MSB is performed with table lookup. This helps reduce the size of the LUT. We present a detailed description of our approximation method in the Appendix B. In addition, since the approximation serves as an additional non-linear operation in neural networks, we find the approximated training does not damage accuracy in practice.

3 Quantized Forward and Backward Propagation on LNS

In this section, we introduce how to apply multi-base LNS to quantized training, as illustrated in Fig. 3.

To realize reduced precision for values and arithmetic during training, we define a logarithmic quantization function $\mathrm{Q}_{\log} : \mathbb{R} \to \mathbb{R}$, which quantizes a real number into a sign and an integer exponent using a limited number of bits. $\mathrm{Q}_{\log}$ is defined as follows:

$$\mathrm{Q}_{\log}(x) = sign(x) \times s \times 2^{(\tilde{x}/\gamma)}, \tag{3}$$

where $\tilde{x} = clamp(\ round(\ \log_2(|x|/s) \times \gamma), 0, 2^{\mathcal{B}-1} - 1)$, and $s \in \mathbb{R}$ denotes a scale factor. $\mathrm{Q}_{\log}$ first brings scaled numbers $|x|/s$ into their logarithmic space, magnifies them by the base factor γ and then performs rounding and clamping functions to convert them into desired integer exponents $\tilde{x}$. The scale factor s usually is shared within a group of numbers, and its value is assigned to match the maximum number within the group.

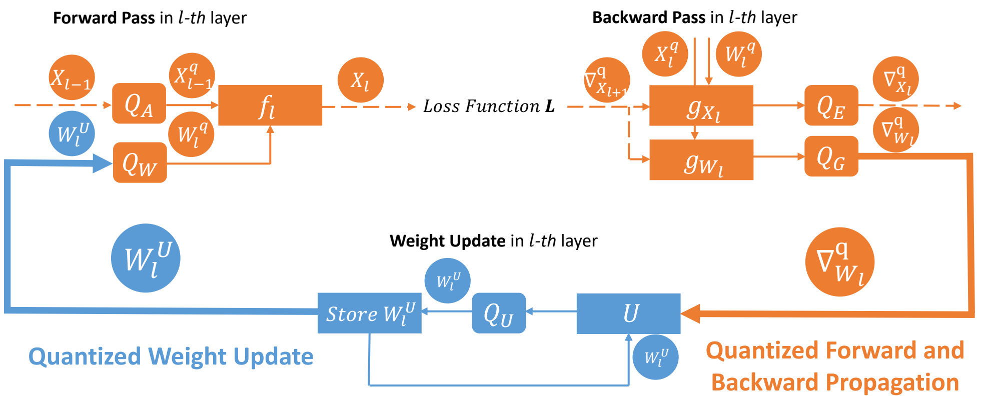
We apply quantization-aware training (QAT) for quantizing weights and activations during forward propagation. Each quantizer is associated with a STE to allow the gradients to directly flow through the non-differentiable quantizer during backward pass [10]. Because QAT views each quantization function as an additional non-linear operation in the networks, the deterministic quantization error introduced by any quantizer in the forward pass is implicitly reduced through training. We define weight quantization function as $\mathrm{Q_W}$ and activation quantization function as $\mathrm{Q_A}$ for each layer during forward propagation, where $W_l^q = \mathrm{Q_W}(W_l)$ and $X_l^q = \mathrm{Q_A}\left(f_l\left(X_{l-1}^q, W_l^q\right)\right)$.

In order to accelerate training in addition to inference, gradients also need to be quantized into low-precision numbers. As shown by recent studies, the distribution of gradients resembles a Gaussian or Log-Normal distribution [11], [12]. This suggests that logarithmic representation may be more suitable than fixed-point representations when quantizing gradients to attain hardware efficiency. We quantize the activation gradients using quantization function $\mathrm{Q_E}$: $\nabla_{X_l}^q = \mathrm{Q_E}\left(\nabla_{X_l}\right)$. We also quantize the weight gradients using quantization function $\mathrm{Q_G}$: $\nabla_{W_l}^q = \mathrm{Q_G}\left(\nabla_{W_l}\right)$. In this work, we aim to reduce the precision requirement for both weight gradients and activation gradients in the backward pass.

4 Multiplicative Weight Update Algorithm for LNS

Although logarithmic quantized training significantly improves training efficiency, its overall efficiency continues to be hampered by the high precision requirement of weight updates. We note that quantized weight update is orthogonal to quantized training due to the difference in their objectives. Quantized training tries to maintain the fidelity of weight gradients while accelerating forward and backward propagation. This provides accurate gradient information for the weight update. On the other hand, after receiving quantized weight gradients, quantized weight update aims to reduce gaps between updated weights and their (rounded) quantized counterparts. Fig. 3 distinguishes the two parts by different colors.

Previous works generally assume that the weight update is computed over a full-precision weight space. In other words, a full-precision copy of weights is maintained [5], [7] and very little rounding follows weight update. However, this offsets the efficiency benefits of quantized training and requires expensive floating-point arithmetic not available especially in cheap energy-constrained edge devices. Therefore, in this work, we consider quantized weight update in LNS, where the weights are updated over a discrete



Forward Pass in l-th layer
Backward Pass in l-th layer
Loss Function L
Weight Update in l-th layer
Store W_l^U
Quantized Weight Update
Quantized Forward and Backward Propagation

Fig. 3. Illustration of LNS-Madam. Quantized training includes quantizing weights W and activations X in forward propagation, and weight gradients ∇_W and activation gradients ∇_X in backward propagation. g_X and g_W denote the functions to compute gradients. Quantized weight update applies a quantization function Q_U over weights after any learning algorithm U updates them. The quantized weights W^U are the actual numbers stored in the system.

logarithmic space instead of a full-precision one. We aim to minimize the rounding error given that weights are represented in LNS.

4.1 Quantized Weight Update

To better understand this problem, we first define a generalized form of a weight update as: $W_{t+1} = U\left(W_t, \nabla_{W_t}\right)$, where U represents any learning algorithm. For example, gradient descent (GD) algorithm takes $U_{GD} = W_t - \eta \nabla_{W_t}$, where η is learning rate.

Because the weights need to be represented in a quantized format in LNS, it is necessary to consider the effect of logarithmic quantization during weight update. We define *logarithmic quantized weight update* as follows:

$$W_{t+1}^U = \mathrm{Q}_{\log}\left(U\left(W_t, \nabla_{W_t}\right)\right). \tag{4}$$

In this case, W_{t+1}^U can be directly stored in a logarithmic format without using floating-point data type. For simplicity, we assume weight gradients ∇_{W_t} are exact as quantized training is orthogonal to this problem. Switching to the approximated gradient estimates will not affect our theoretical results.

4.2 Quantization Error Analysis

Because the logarithmic quantization requires representing values in a discrete logarithmic scale, quantized weight update inevitably introduces a mismatch between the quantized weights and their full-precision counterparts. To preserve the reliability of the optimization, we aim to reduce the quantization error (i.e., the mismatch). For the following, we take a theoretical perspective to discuss how different learning algorithms affect the quantization error under LNS. Detailed assumptions and proofs can be found in Appendix A.

Due to the logarithmic structure, we focus on minimizing a quantizaion error $r_t = \|\log_2 |W_{t+1}^U| - \log_2 |W_{t+1}|\|^2$, which measures the L2 norm of the difference between the weights and their quantized counterparts in logarithmic space. Because r_t quantify the relative difference between $|W_{t+1}^U|$ and $|W_{t+1}|$, minimizing r_t is largely similar to minimizing the relative quantization error $\|(W_{t+1} - W_{t+1}^U)/W_{t+1}\|^2$.

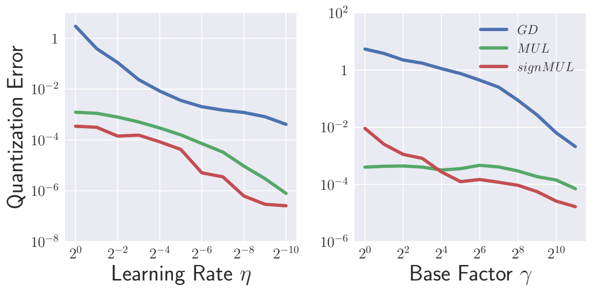


Fig. 4. Quantization error from different learning algorithms on ImageNet. The errors are averaged over all iterations in the first epoch. The results suggest that multiplicative algorithms introduce significantly lower errors compared to the gradient descent, which are also in line with our theoretical results.

We assume a simplified logarithmic quantization where the scale factor and the clamping function are ignored. This ensures our focus is on the effect of the quantization gap determined by γ instead of the dynamic range. We also replace the deterministic rounding with a stochastic counterpart SR where $\mathbb{E}\,\mathrm{SR}(x) = x$ for any real number. Although SR helps us establish the theoretical results, in practice SR requires random generators that induce additional costs, and thus are not suitable for energy-efficient training.

Given everything we need, we use gradient descent as an example to discuss why traditional learning algorithms are not suited for LNS-based quantized weight updates. The theorem is stated as follows:

Theorem 1. *The quantization error $r_{t,GD}$ introduced by logarithmic quantized gradient descent at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,GD} \leq \frac{\sqrt{d}}{\gamma} \|\log_2 \left(|W_t| - \eta_1 \nabla_{W_t}\right)\|, \tag{5}$$

where d is the dimension of W and η_1 is the learning rate of U_{GD}.

Theorem 1 suggests that $r_{t,GD}$ is magnified when the magnitudes of weights become larger. This is because the updates $\eta_1 \nabla_{W_t}$ generated by GD are not proportional to the magnitudes of weights. $\eta_1 \nabla_{W_t}$ can be orders of magnitude smaller than the quantization gaps as weights become larger, and thus these updates often are disregarded by quantization function $\mathrm{Q}_{\log}$. We intuitively illustrate this problem in Fig. 1.

To ensure the updates are proportional to the weights, a straightforward way is to update the weights multiplicatively. Because the weights are represented in LNS, we further consider a special multiplicative learning algorithm tailored to LNS, which updates the weights directly over their logarithmic space:

$$U_{MUL} = sign(W_t) \odot 2^{\tilde{W}_t - \eta \nabla_{W_t} \odot sign(W_t)} \tag{6}$$

where $\tilde{W}_t = \log_2 |W_t|$ are the exponents of the magnitude of weights, and $\odot$ denotes element-wise multiplication. U_{MUL} makes sure the magnitude of each element $W_t(k)$ of the weights decreases when the sign $sign(W_t(k))$ and $\nabla_{W_t(k)}$ agree and increases otherwise. The quantization error with regards to U_{MUL} is stated as follows:

Theorem 2. *The quantization error $r_{t,MUL}$ introduced by logarithmic quantized multiplicative weight update at iteration t can be bounded in expectation, as:*

$$\mathbb{E}\, r_{t,MUL} \le \frac{\sqrt{d}\,\eta_2}{\gamma}\, \|\nabla_{W_t}\|, \tag{7}$$

where d is the dimension of W and η_2 is the learning rate of U_{MUL}.

Theorem 2 indicates that $r_{t,MUL}$ does not depend on the magnitudes of weights, and thus the quantization error is not magnified when the weights become larger. This is in stark contrast to the quantization error from gradient descent shown in Equation 5. The comparison is illustrated in Fig. 1.

Interestingly, we find that the quantization error $r_{t,MUL}$ can be further simplified by regularizing the information of gradients for the learning algorithm U_{MUL}:

Lemma 1. *Assume the multiplicative learning algorithm U_{MUL} only receives the sign information of gradients where $U_{MUL} = \tilde{W}_t - \eta_2\, sign(\nabla_{W_t}) \odot sign(W_t)$. The upper bound on quantization error $r_{t,MUL}$ becomes:*

$$\mathbb{E}\, r_{t,MUL} \le \frac{d\,\eta_2}{\gamma}. \tag{8}$$

The result in Lemma 1 suggests that $r_{t,MUL}$ can be independent of both weights and gradients when only taking sign information of gradients during weight update. We denote this special learning algorithm as $U_{signMUL}$. $U_{signMUL}$ is a multiplicative version of signSGD, which has been studied widely [12], [13].

To verify our theoretical results, we empirically measure the quantization errors for the three aforementioned learning algorithms over a range of η and γ. As shown in Fig. 4, the empirical findings are in line with our theoretical results. Although all learning algorithms introduce less errors when η and γ become smaller, the multiplicative algorithms introduce significantly lower errors compared to the gradient descent.

In addition to reducing the quantization error in the quantized weight update, $U_{signMUL}$ must also have the ability to minimize the loss function $\mathcal{L}(W)$. Interestingly, we notice that $U_{signMUL}$ resembles a recently proposed learning algorithm Madam, where Bernstein et al. [8] proves that Madam optimizes the weights in a descent direction.

Algorithm 1 *Madam optimizer on LNS*

```
Require: Base-2 weight exponents $\tilde{W}$, where $\tilde{W} = \log_2(W)$, learning rate $\eta$, momentum $\beta$
Initialize $g_2 \leftarrow 0$
repeat
    $g \leftarrow \mathrm{StochasticGradient}()$
    $g_2 \leftarrow (1-\beta)g^2 + \beta g_2$
    $g^* \leftarrow g/\sqrt{g_2}$
    $\tilde{W} \leftarrow \tilde{W} - \eta\, g^* \odot sign(W)$
until converged
```

Madam updates the weights multiplicatively using normalized gradients:

$$\begin{aligned} U_{Madam} &= W_t \odot e^{-\eta\, sign(W_t) \odot g_t^*} \\ g_t^* &= g_t / \sqrt{g_{2_t}} \end{aligned} \tag{9}$$

where g_t represents the gradient vector ∇_{W_t}, and g_t^* denote a normalized gradient, which is the fraction between g_t and the square root of its second moment estimate $\sqrt{g_{2_t}}$. Bernstein et al. [8] demonstrates that Madam achieves state-of-the-art accuracy over multiple tasks with a relatively fixed learning rate η. They also theoretically prove the descent property of Madam that ensures its convergence. Although Bernstein et al. [8] further shows the possibility of applying Madam over a discrete logarithmic weight space, they still employ full-precision training without considering low-precision LNS.

To ensure low-precision weight updates in LNS, we apply a modified version of the Madam optimizer to enable fast convergence while preserving low quantization error. The modified Madam directly optimizes the weights over their base-2 logarithmic space using the gradient normalization technique described in Equation 9. Details of our optimizer are shown in Algorithm 1. Because our Madam optimizer directly updates base-2 exponents of weights in LNS, there is no need for integer-to-LNS conversion during weight update when the weights are already in LNS, further reducing the energy cost.

5 Hardware Implementation

We extend a previously optimized DNN accelerator [14] to support LNS-based DNN computations. Fig. 5 shows the micro-architecture of the PE which performs dot-product operations. Each PE consists of set of vector MAC units fed by the buffers that store weights, input activations, and output gradients. Additionally, the accumulation collectors store and accumulate partial sums which are passed to the PPU for post-processing (e.g., quantization scaling, non-linear activation functions) if necessary.

Fig. 6 shows the LNS-based datapath inside the LNS-Madam Vector MAC Unit. Here we model exact LNS-to-integer conversion without any approximation. With a vector size of 32 and input bitwidths of 8, the datapath processes 32 7-bit exponent values at each of its exponent inputs (e_a and e_b) and 32 1-bit sign values (s_a and s_b) at each of its sign inputs to produce a 24-bit partial sum. First, the LNS datapath performs the dot-product multiplication by adding the exponents and XOR-ing the sign bits. The output of the product computation requires an additional bit to account for the carry-out of the adder. At

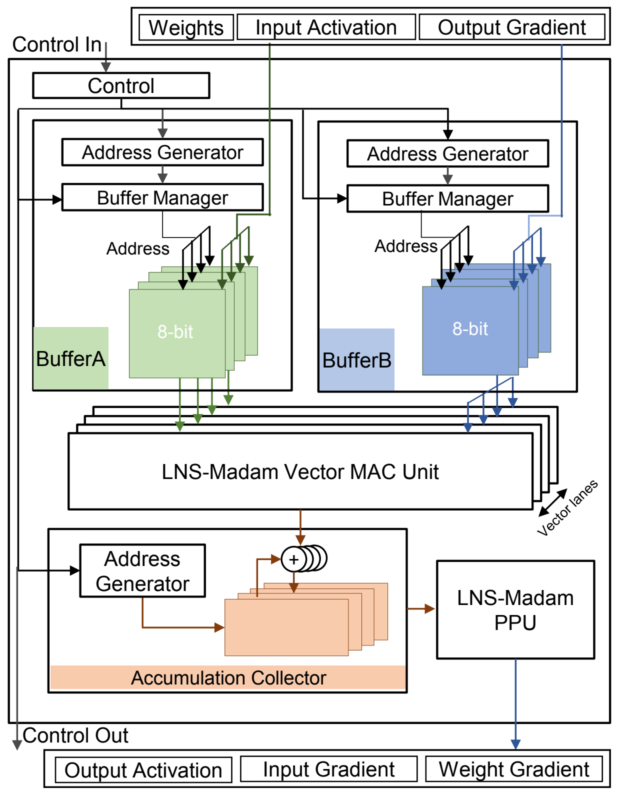


Fig. 5. LNS-Madam processing element (PE).

this point, each exponent is split into a quotient component (e_q) and a remainder component (e_r) based on the LSB/MSB property mentioned in Section 2.2. Second, the datapath performs shifting by the quotient to implement the quotient component in Equation 2. Depending on the sign bit, the corresponding signed shifted value is selected and passed to the corresponding adder tree based on the remainder select signal. Third, the result of shifted values are reduced through the set of adder trees and registered. At last, the results of the adder trees are multiplied with corresponding remainder constants (described in Section 2.2) from a LUT and accumulated into the final partial sum, represented in integer (linear) format. This partial sum needs to be converted back into logarithmic format and written back to the global buffer for subsequent LNS-based computations.

Additional microarchitectural details of the PE are listed in Table 5. Notably, our accelerator uses a multi-level dataflow called output-stationary local-A-stationary [14] to optimize reuse across different operands. Inputs from buffer A are read out once every 16 cycles and stored in a register for temporal reuse. Inputs from buffer B are read once every cycle and reused across the 32 lanes spatially. Partial sums are temporally accumulated in a 16-entry latch array collector before sending the completed sum to the post-processing unit. The two buffers in the PE store different data depending on whether output activation, input gradient, or weight gradient is being computed. For example, weights and input activations are stored in `BufferA` and `BufferB` respectively during forward propagation to compute the output activations. On the other hand, input activations and output gradients are stored in the respective buffers during backward propagation to compute the weight gradient. Table 2 outlines how we map various tensors in DNN computation to buffers in our hardware during different computation passes. Note that weight updates are performed outside of the PEs through the global buffer.

TABLE 1
Microarchitectural details of LNS-Madam PE

Dataflow	Multi-level
Vector size / # Vector lanes	32
Weight/activation precision	8-bit
Gradient precision	8-bit
# Remainder Bins	8
Accumulation precision	24-bit
Accumulation collector size	1.5 KB
BufferA size	128 KB
BufferB size	8 KB

TABLE 2
Mapping of tensors to buffers in PE during different computation passes[1]

Pass	**BufferA**	**BufferB**
Forward	Weight	Input Activation
Backward (Input)	Weight	Output Gradient
Backward (Weight)	Input Activation	Output Gradient

[1] Backward pass consists of backward computation for input gradient, denoted `Backward(Input)`, and backward computation for weight gradient, denoted `Backward(Weight)`.

6 Experiments

In this section, we evaluate both the accuracy and energy efficiency of using LNS-Madam to train state-of-the-art models on large-scale datasets.

6.1 Model Accuracy

To evaluate accuracy, we simulate LNS-Madam using a PyTorch-based neural network quantization library that implements a set of common neural network layers (e.g., convolution, fully-connected) for training and inference in both full and quantized modes [15]. The baseline library supports integer quantization in a fixed-point number system, and we further extend it to support LNS. The library also provides utilities for scaling values to the representable integer range of the specific number format. With this library, a typical quantized layer consists of a conventional layer implemented in floating-point preceded by a weight quantizer and an input quantizer that converts the weights and inputs of the layer to the desired quantized format. For the backward pass, after the gradients pass through the STE in each quantizer, they will be quantized by their quantizers as well.

We benchmark LNS-Madam on various tasks including ResNet models on CIFAR-10 and ImageNet, and BERT-base and BERT-large language models on SQuAD and GLUE. Specifically, we train ResNet models from scratch on CIFAR-10 and ImageNet, and fine-tune pre-trained BERT models on SQuAD and GLUE. Detailed descriptions of datasets and models can be found in the appendix.

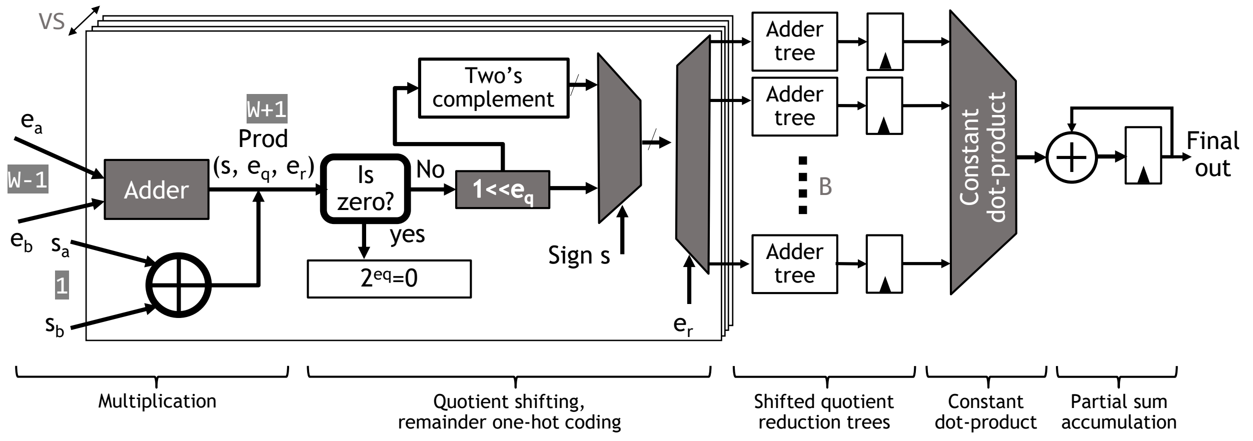


Fig. 6. LNS-Madam Vector MAC Unit – Performs dot-products of inputs represented in LNS and produces partial sum outputs in integer format. Bitwidths of different signals are highlighted. VS stands for vector size; W stands for bitwidth of input values; B refers to base factor and number of remainder bins.

TABLE 3
Base Factor Selection on ImageNet[1,2]

γ	Dynamic Range	Forward	Backward
1	(0,127)	NaN	NaN
2	(0,63.5)	75.81	75.79
4	(0,31.8)	75.96	76.07
8	(0,15.9)	75.88	76.23
16	(0,7.9)	76.32	63.67
32	(0,4.0)	68.15	20.71

[1] Bitwidth is 8-bit across settings. Quant Forward or Quant Backward denotes the settings where either forward propagation or backward propagation is quantized while leaving the rest of computation in full-precision.
[2] The results of test accuracy (%) are listed.

6.1.1 Parameter Settings

We fix the bitwidth to be 8-bit for both forward and backward propagation, which includes the bitwidth for weights, activations, activation gradients, and weight gradients. We note that 8-bit weight gradients are lower than previous studies that use 16-bit or even 32-bit weight gradients [5], [6].

To find an appropriate base factor γ under the 8-bit setting, we vary γ to find the appropriate dynamic ranges for forward and backward qantization. The dynamic range in LNS is $(0,(2^{\mathcal{B}-1}-1)/\gamma)$, which is controlled by both bitwidth and base factor.

As shown in Table 3, we fix the bitwidth as 8-bit and vary the base factor γ to find the appropriate dynamic ranges for forward and backward quantization. According to the results, we find the base factor of 8 with the dynamic range $(0,15.9)$ that uniformly works across $\mathrm{Q_W}$,$\mathrm{Q_A}$,$\mathrm{Q_E}$, and $\mathrm{Q_G}$.

In order to maintain optimization stability, the bitwidth of the weight updates require to be larger than the bitwidth of the weights. When the bitwidth of $\mathrm{Q_U}$ is larger than 8-bit, we increase its base factor to match the desired dynamic range $(0,15.9)$.

We empirically search the best learning rate η for our Madam optimizer from 2^{-4} to 2^{-10}, and we find $\eta=2^{-7}$ works best uniformly across tasks, which suggests the learning rate for Madam is robust.

TABLE 4
Benchmarking LNS-Madam on various datasets and models[1]

Dataset	Model	LNS-Madam[2]	FP8[2]	FP32
CIFAR-10	ResNet-18	93.41	93.12	93.51
ImageNet	ResNet-50	76.14	75.83	76.38
SQuAD	BERT-base	88.13	88.07	88.36
SQuAD	BERT-large	90.75	90.54	90.80
GLUE	BERT-base	88.89	88.73	88.92
GLUE	BERT-large	89.24	88.91	89.35

[1] The results of test accuracy (%) are listed.
[2] Forward and backward propagation are in 8-bit, and the weight update is in 16-bit.

TABLE 5
Comparing LNS-Madam with recent low-precision training methods on 8-bit training[1]

	Data format	16-bit	32-bit
LNS-Madam	LNS	**76.14**	76.23
BHQ [16]	INT	74.89	**76.35**
Unified INT8 [17]	INT	74.73	76.27
FP8 [2]	FP	71.46	71.53

[1] Evaluate ResNet-50 on ImageNet. Forward and backward propagation are in 8-bit. Test accuracy (%) evaluated under 32-bit and 16-bit weight update are presented.

TABLE 6
Comparing LNS-Madam and BHQ over a range of bitwidth[1]

	4-bit	5-bit	6-bit	7-bit	8-bit
LNS-Madam	**74.23**	**75.89**	74.41	**76.16**	76.23
BHQ [16]	74.04	75.70	**76.21**	76.14	**76.35**

[1] Bitwidth of activation gradients varies from 4-bit to 8-bit. The results of test accuracy (%) are listed.

6.1.2 Comparisons

Given the settings above, we compare LNS-Madam with FP8 and FP32. For FP8 and FP32, the standard optimizers are applied for tasks by default. We use a tuned SGD optimizer for CIFAR-10 and ImageNet datasets, and a tuned AdamW optimizer for SQuAD and GLUE datasets. For all settings, we use per-channel scaling for ResNet and per-

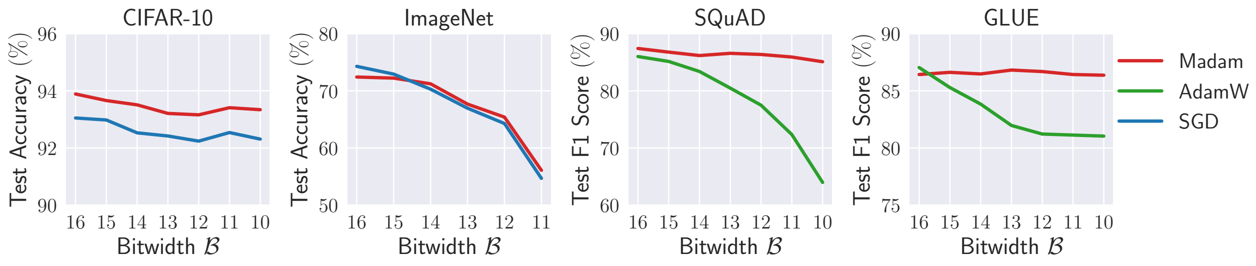


Fig. 7. Comparing Madam with SGD and Adam optimizers under the logarithmic quantized weight update (defined in Equation 4). The bitwidth of the weight update Q_U is varied from 16-bit to 10-bit.

feature scaling for BERT, both of which are commonly used scaling techniques. The clamping function is performed by matching the largest value within each group of numbers.

To demonstrate LNS-Madam is better than popular number systems, we compare it with both FP8 and FP32, where our FP8 contains 4-bit exponent and 3-bit mantissa. As shown in Table 4, LNS-Madam yields better performance than FP8, and it even achieves performance comparable to the full-precision counterpart.

In addition, we compare LNS-Madam with recent methods on low-precision training. For all methods, we fix forward and backward propagation in 8-bit while varying the weight update precision from 32-bit to 16-bit. As shown in Table 5, LNS-Madam achieves the best accuracy under 16-bit weight update, which demonstrates its effectiveness under the quantized setting. FP8 [2] also achieves negligible degradation after switching to 16-bit, as it applies stochastic rounding over weight update process. Since BHQ [16] achieves the best accuracy under 32-bit, we also compare it with LNS-Madam over a range of bitwidth settings in Table 6.

We also compare Madam with the default optimizers SGD and AdamW under logarithmic quantized weight update, as defined in Equation 4. All optimizers use the same learning rates as above.

As shown in Fig. 7, we vary the bitwidth of the quantized weight update Q_U from 16-bit to 10-bit to test their performance over a wide range. The results suggest compared to other optimizers, *Madam always maintains higher accuracy when precision is severely limited*. Notably, for BERT model on SQuAD and GLUE benchmarks, Madam is 20% better than Adam with respect to F-1 score, when the weight update is in 10-bit. We observe large degradation for both Madam and SGD on ImageNet training, and we believe this is because the weights in some layers inevitably require higher precision settings. We leave it as future work to explore LNS-Madam under a customized precision setting.

TABLE 7
Design tools used for LNS-Madam hardware experiments

HLS Compiler	Mentor Graphics Catapult HLS
Verilog simulator	Synopsys VCS
Logic synthesis	Synopsys Design Compiler
Place-and-route	Synopsys ICC2
Power Analysis	Synopsys PT-PX

6.2 Energy Efficiency

We leverage the hardware implementation described in Section 5 to evaluate the energy efficiency of LNS-Madam. We code the hardware model in C++ and Verilog and synthesize it to a combined cycle-accurate RTL using a commercial high-level synthesis tool [18]. Once the RTL is generated, a standard logic synthesis flow is used to obtain the gate-level netlist that is then simulated with representative inputs. To extract energy consumption, we supply the gate-level simulation results from post-synthesis to a standard power analysis tool. We then use an analytical model to compute the total energy consumption for different workloads. We perform our analysis in a sub-16nm state-of-the-art process technology at 0.6V targeting a frequency of 1.05 GHz. Table 6.1.2 summarizes the design tools used in the evaluation.

TABLE 8
Energy efficiency for different models and number formats[1]

Model	LNS	FP8	FP16	FP32
ResNet-18	0.54	1.22	2.50	5.99
ResNet-50	0.99	2.25	4.59	11.03
BERT-Base	7.99	18.23	37.21	89.35
BERT-Large	27.85	63.58	129.74	311.58

[1] The per-iteration energy consumption in mJ are listed.

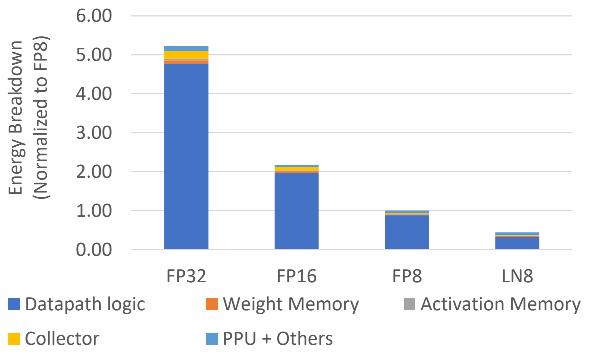


Fig. 8. Energy breakdown of the PE shown in Fig. 6 for different dataformats.

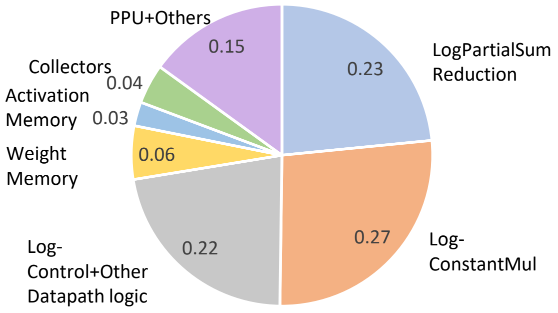


Fig. 9. LNS PE energy breakdown showing different components of datapath

TABLE 9
Comparing LNS-Madam with recent LNS-based designs

	LNS-Madam	Arnab et al. [19]	Miyashita et al. [5]	Lee et al. [20]	Vogel et al. [7]
Support inference or fine-tuning?	✓	✓	✓	✓	✓
Support training from scratch?	✓	✓	✗	✗	✗
Weight update precision	<16-bit	32-bit	32-bit	32-bit	32-bit
Efficient log-to-linear conversion support	✓	✓	✓	✓	✗
Large-scale evaluation	✓	✗	✓	✓	✓

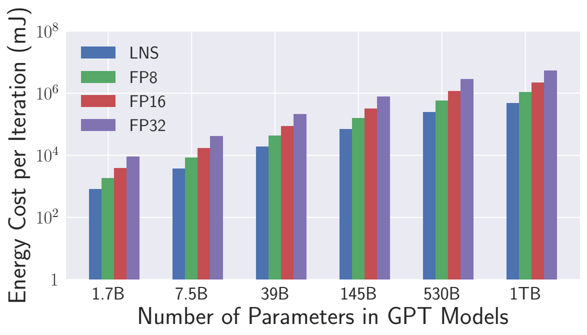


Fig. 10. Energy efficiency over a range of GPT models from 1 billion to 1 trillion parameters. The models are scaled by a throughput efficient method proposed by Narayanan et al. [21].

In our experiment, the LNS-Madam hardware is designed with bitwidth $\mathcal{B} = 8$ and base factor $\gamma = 8$ for both forward and backward computations. In addition to experimenting with the LNS-based datapath shown in Fig. 6, we also consider FP8, FP16, and FP32 datapath baselines for comparison.

Table 8 presents the energy efficiency per iteration of one forward pass and one backward pass of training. Because different number systems share the same training iterations, per-iteration energy results imply the energy comparison over the entire training. We also present the energy breakdown of the whole PE in Figure 8. As shown in the figure, FP arithmetic is extremely expensive, contributing a large fraction to the total energy consumption of PEs. The proposed LNS datapath offers significant reduction in the logic complexity, leading to 2.2X, 4.6X, and 11X energy efficiency improvements over FP8, FP16, and FP32 implementations, respectively. We also provide a detailed energy breakdown showing different components of the LNS PE in Fig. 9. In addition, Fig. 10 shows the energy efficiency over a range of GPT models from 1 billion to 1 trillion parameters.

7 Related Works

7.1 Low-precision training

To achieve good accuracy at reduced precision, quantization-aware training (QAT) is commonly applied to directly train the quantized model using straight-through estimators [10], [22], [23], [24], [25]. To accelerate the training phase, several studies suggest quantizing the gradients during backward propagation [2], [6], [26]. To maintain the fidelity of the gradient accumulation, some low-precision training methods assume a full-precision copy for weights during the weight update [16], [26]. Other studies reduce the precision for the weight update by using high-precision gradient accumulator [27], stochastic rounding [2], [28] or additionally quantizing the residual part of weights [29], [30]. Cambricon-Q accelerates the weight update from a hardware perspective by avoiding costly data transferring in weight update [31]. However, they mostly apply SGD or Adam during the weight update without considering the relationship between the precision of the weights and the underlying learning algorithms.

7.2 Logarithmic number system

Previous works demonstrate the effectiveness of using logarithmic representation for DNNs [5], [20], [32], [33]. Furthermore, some studies suggest using multiple levels of log-base to reduce the quantization error [5], [7]. However, few of them address the additional computational cost induced by this multi-base design nor scale the system to state-of-the-art neural networks for both training and inference. From the perspective of hardware design, a few studies focus on improving the efficiency of LNS by utilizing the significant cost reduction of multiplications [32], [33], [34], [35], [36]. We compare LNS-Madam with recent LNS-based designs on Table 9.

7.3 Multiplicative weight update

Multiplicative algorithms, such as exponentiated gradient algorithm and Hedge algorithm in AdaBoost framework [37], [38], have been well studied in the field of machine learning. In general, multiplicative updates are applied to problems where the optimization domain's geometry is described by relative entropy, such as probability distribution [37]. Recently, [8] proposes an optimizer Madam that focuses on optimization domains described by any relative distance measure instead of only relative entropy. Madam shows great performance in training large-scale neural networks. However, Madam requires full-precision training without considering its connection to LNS-based low-precision training.

8 Conclusions

In this work, we propose a co-designed low-precision training framework LNS-Madam that jointly considers the logarithmic number system and the multiplicative weight update algorithm. Experimental results show that LNS-Madam achieves comparable accuracy to full-precision counterparts even when forward and backward propagation, and weight updates are all in low-precision. To support the training framework in practice, we design a hardware implementation of LNS-Madam to efficiently perform the necessary LNS computations for DNN training. Based on our energy analysis, LNS-Madam reduces energy consumption by over 90% compared to a floating-point baseline.

An important application of our low-precision training framework is learning neural networks over energy-constrained edge devices. This is fundamental for intelligent edge devices to easily adapt to changing and non-stationary environments by learning on-device and on-the-fly. By enabling highly energy-efficient training, our work carries the promising opportunity for using LNS-based hardware to conduct environmental-friendly deep learning research in the near future.

References

[1] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015.

[2] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Neural Information Processing Systems*, 2018.

[3] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[4] T. M. Bartol, Jr., C. Bromer, J. Kinney, M. A. Chirillo, J. N. Bourne, K. M. Harris, and T. J. Sejnowski, "Nanoconnectomic upper bound on the variability of synaptic plasticity," *eLife*, 2015.

[5] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," 2016.

[6] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1796–1807. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/13b919438259814cd5be8cb45877d577-Paper.pdf

[7] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient hardware acceleration of cnns using logarithmic data representation with arbitrary log-base," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3240765.3240803

[8] J. Bernstein, J. Zhao, M. Meister, M.-Y. Liu, A. Anandkumar, and Y. Yue, "Learning compositional functions via multiplicative weight updates," *arXiv preprint arXiv:2006.14560*, 2020.

[9] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[10] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[11] B. Chmiel, L. Ben-Uri, M. Shkolnik, E. Hoffer, R. Banner, and D. Soudry, "Neural gradients are near-lognormal: improved quantized and sparse training," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=EoFNy62JGd

[12] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signSGD with majority vote is communication efficient and fault tolerant," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BJxhijAcY7

[13] ——, "signsgd with majority vote is communication efficient and fault tolerant," 2019.

[14] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, Y. Zhang, B. Zimmer, W. J. Dally, J. Emer, S. W. Keckler, and B. Khailany, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2019, pp. 1–8.

[15] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[16] J. Chen, Y. Gai, Z. Yao, M. W. Mahoney, and J. E. Gonzalez, "A statistical framework for low-bitwidth training of deep neural networks," *arXiv preprint arXiv:2010.14298*, 2020.

[17] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards Unified INT8 Training for Convolutional Neural Network," 2020, pp. 1969–1979. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/html/Zhu_Towards_Unified_INT8_Training_for_Convolutional_Neural_Network_CVPR_2020_paper.html

[18] M. McFarland, A. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 301–318, Feb. 1990.

[19] A. Sanyal, P. A. Beerel, and K. M. Chugg, "Neural network training with approximate logarithmic computations," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3122–3126, ISSN: 2379-190X.

[20] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 5900–5904.

[21] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters," 2021.

[22] 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," *arXiv preprint arXiv:1606.06160*, 2016.

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[24] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[25] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[26] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 5145–5153. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/e82c4b19b8151ddc25d4d93baf7b908f-Paper.pdf

[27] C. Sakr and N. Shanbhag, "Per-tensor fixed-point quantization of the back-propagation algorithm," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rkxaNjA9Ym

[28] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HJGXzmspb

[29] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," *Advances in neural information processing systems*, vol. 32, pp. 4900–4909, 2019.

[30] C. D. Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré, "High-accuracy low-precision training," 2018.

[31] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang, L. Li, Z. Xu, and T. Chen, "Cambricon-Q: A Hybrid Architecture for Efficient Training," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, Jun. 2021, pp. 706–719. [Online]. Available: https://ieeexplore.ieee.org/document/9499944/

[32] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018.

[33] J. Johnson, "Efficient, arbitrarily high precision hardware logarithmic arithmetic for linear algebra," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, 2020, pp. 25–32.

[34] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[35] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate integer and floating-point dividers with near-zero error bias," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[36] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran, "Realm: reduced-error approximate log-based integer multiplier," in *2020*

Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 1366–1371.

[37] J. Kivinen and M. K. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *Information and Computation*, 1997.

[38] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, 1997.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*, 2016.

[40] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

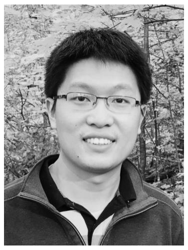
[41] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition*, 2009.

[43] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. [Online]. Available: https://www.aclweb.org/anthology/D16-1264

[44] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. [Online]. Available: https://www.aclweb.org/anthology/W18-5446

Jiawei Zhao received the B.S. degree in Computer Science from NUAA, Nanjing, China, in 2019. He is currently pursuing the Ph.D. degree at Caltech, under the supervision of Prof. Anima Anandkumar. His research interest lies in machine learning, specifically in deep learning and optimization. He is working on the development of novel and efficient algorithms for solving deep learning problems, such as distributed or low-precision training.

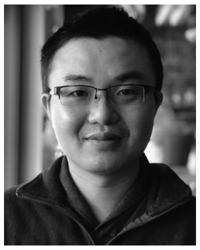


Steve Dai is currently a Research Scientist as part of the ASIC & VLSI Research Group at NVIDIA. His research interests include energy-efficient DL acceleration, high-level design methodologies, and ML-assisted EDA. Dai received the B.S. degree in electrical engineering from the University of California at Los Angeles in 2011, the M.S. degree in electrical engineering from Stanford University in 2013, and the Ph.D. degree in electrical and computer engineering from Cornell University in 2019.

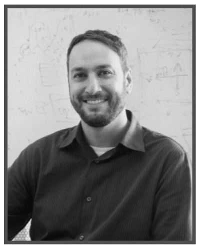
Rangharajan Venkatesan is a Senior Research Scientist with NVIDIA. His research interests include machine learning accelerators, low-power VLSI design, and SoC design methodologies. He has served as a member of the technical program committees of several leading IEEE conferences including International Solid-State Circuits Conference,International Symposium on Micro architecture, Design Automation Conference, and International Symposium on Low Power Electronics and Design. Venkatesan received the B.Tech. degree in electronics and communication engineering from the Indian Institute of Technology in 2009 and the Ph.D.degree in electrical and computer engineering from Purdue University in 2014.

Brian Zimmer received the B.S. degree in electrical engineering from the University of California at Davis, Davis, CA, USA, in 2010, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, in 2012 and 2015, respectively. He is currently a Senior Research Scientist with the Circuits Research Group, NVIDIA, Inc., Santa Clara, CA. His research interests include soft error resilience, energy-efficient digital design, low-voltage static random-access memory (SRAM) design, machine learning accelerators, productive design methodologies, and variation tolerance.

Mustafa Ali received the B.Sc. (Hons.) and M.Sc. degrees in electrical engineering from MTC, Cairo, Egypt, in 2011 and 2016, respectively. He is currently pursuing the Ph.D. degree with Purdue University, under the guidance of Prof. Roy. He was honored the Duty Medal for excellent performance during his studies. He worked on flexible electronics applications using TFTs in his M.Sc. degree from 2014 to 2016. In Addition, he worked as a TA and RA at MTC from 2013 to 2017. He was also a Hardware and Embedded Systems Engineer at Integreight, Inc., from 2012 to 2017. He joined the Nano-Electronics Research Lab (NRL), Purdue University, in Spring 2018. His research interests include accelerating brain-inspired computing and machine learning. He is also interested in in-memory computing based on CMOS and post-CMOS devices.



Ming-Yu Liu received the PhD degree from the Department of Electrical and Computer Engineering, University of Maryland, College Park, Maryland, in 2012. He is currently a Distinguished Research Scientist and a Manager with NVIDIA Research, Santa Clara, CA, USA. Before joining NVIDIA in 2016, he was a principal research scientist with Mitsubishi Electric Research Labs. His object pose estimation system was awarded one of hundred most innovative technology products by the R&D magazine in 2014. In CVPR 2018, he won the first place in both the Domain Adaptation for Semantic Segmentation Competition in the WAD challenge and the Optical Flow Competition in the Robust Vision Challenge. His research interests include generative models for image generation and understanding. His goal is to enable machines superhuman-like imagination capabilities.



Brucek Khailany joined NVIDIA in 2009 and is the Director of the ASIC and VLSI Research group. He leads research into innovative design methodologies for IC development, ML and GPU assisted EDA, and energy efficient ML accelerators. Over 10 years at NVIDIA, he has contributed to many projects in research and product groups spanning computer architecture and VLSI design. Previously, Dr. Khailany was a Co-Founder and Principal Architect at Stream Processors, Inc where he led R&D related to parallel processor architectures. At Stanford University, he led the VLSI implementation of the Imagine processor, which introduced the concepts of stream processing and partitioned register organizations. He received his PhD in Electrical Engineering from Stanford University and BSE degrees in Electrical and Computer Engineering from the University of Michigan. He is a Senior Member of the IEEE.

William J. Dally is the Chief Scientist and Senior Vice President of research at NVIDIA Corporation, Santa Clara, CA, USA, and an Adjunct Professor and former Chair of computer science at Stanford University, Stanford, CA, USA. His research interests include domain-specific accelerators, parallel computer architectures, interconnection networks, and high-speed signaling circuits. Dally received a Ph.D. degree in computer science in 1986 from the California Institute of Technology, Pasadena, CA, USA. He is a Member of the National Academy of Engineering, and a Fellow of IEEE, ACM, and the American Academy of Arts and Sciences.

Anima Anandkumar was a Principal Scientist with Amazon Web Services. She is currently a Bren Professor with the California Institute of Technology (Caltech), Pasadena, CA, USA, and the Director of ML Research with NVIDIA. She is part of the World Economic Forum's Expert Network. She is passionate about designing principled AI algorithms and applying them in interdisciplinary applications. Her research focus is on unsupervised AI, optimization, and tensor methods. Dr. Anandkumar received several honors, such as the Alfred P. Sloan Fellowship, the NSF Career Award, the Young Investigator Awards from DoD, and Faculty Fellowships from Microsoft, Google, Facebook, and Adobe.