TraiNDSim: A Simulation Framework for Comprehensive Performance Evaluation of Neuromorphic Devices for On-Chip Training

Donghyeok Heo[1], Hyeonsu Bang[2], Jong Hwan Ko[3]
[1] Department of AI Systems Engineering, Sungkyunkwan University, Suwon, South Korea
[2] Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, South Korea
[3] College of Information and Communication Engineering, Sungkyunkwan University, Suwon, South Korea
{gjehdgur5392, bhs1996, jhko}@skku.edu

ABSTRACT

The advancement of neuromorphic devices (NDs) for processing deep neural networks has narrowed the accuracy gap with software-trained models. To accurately assess ND performance, reliable simulation frameworks for on-chip training are crucial. However, existing frameworks encounter difficulties accurately reflecting the characteristics of NDs in training simulations. Consequently, we introduce TraiNDSim, a novel framework that comprehensively evaluates the performance of NDs to address these difficulties. Specifically, we propose an advanced conductance normalization strategy called layer-wise normalization, which limits the weight range by taking the initial weight distribution into account. Additionally, our framework integrates three conductance models, notably refining one of the conventional models to depend solely on nonlinearity. Moreover, it features a bi-directional weight representation method with a unique conductance compensation technique. Our comprehensive analysis using TraiNDSim demonstrates its effectiveness in accurately reflecting the impact of ND parameters on training, promising more precise device performance evaluations. Our framework is available at https://github.com/donghyeokheo/TraiNDSim.

1 INTRODUCTION

With the advancement of deep neural networks (DNNs), convolutional neural networks (CNNs) have demonstrated excellent performance in the field of image processing. The operation of CNNs entails multiply and accumulate (MAC) operations on pairs of matrices, weights and input feature maps (IFMs). However, the traditional von Neumann architecture leads to significant energy consumption and decreased computing speed as a result of substantial data movement during MAC operations. To address these challenges, a processing-in-memory (PIM) approach has been introduced.

PIM architecture performs MAC operations within the memory array where cells are connected at the intersection points of each word-line and bit-line. This crossbar structure efficiently leverages Ohm and Kirchhoff's laws for MAC operations, significantly minimizing data movement and eliminating summation and multiplication units [8, 15]. Non-volatile memories such as RRAM [27], PCM [11], FeFET [9], and ECRAM [25] are integrated into this structure, functioning as neuromorphic devices (NDs) or memristors.

Weight updates during on-chip training are executed by adjusting the ND's conductance value, guided by long-term potentiation/depression (LTP/LTD) characteristics. Conductance is increased by applying potentiated voltage pulses, vice versa. The amount of voltage pulses is determined by the gradient derived via backpropagation (BP) [14, 23]. Two strategies simulate this process: gradient-based update method (GUM) and pulse-based update method (PUM). Moreover, the LTP/LTD characteristics are represented through conductance modeling formulas, incorporating the non-ideal aspects of NDs as fitting parameters. These non-idealities include nonlinearity, on/off ratio, and maximum conductance level. Moreover, distortion in the conductance values can arise due to cycle-to-cycle (C2C) and device-to-device (D2D) variations.

In the community focusing on NDs, a recent method of evaluating device performance involves analyzing the results from simulation-driven on-chip training of DNNs. Three notable frameworks, namely CrossSim [1], TxSim [22], and NeuroSim [4, 17], were dedicated to facilitating on-chip training simulation. However, these frameworks exhibit inconsistencies in the training process, meaning that the performance of the device is not accurately reflected in the simulation. CrossSim [1] and TxSim [22] used the flawed conductance update method, which overlooks the impact of discrete characteristics of conductance. NeuroSim [17] employed the WAGE quantization technique [26] that neglects the influence of nonlinearity in determining the final changes in conductance.

We identified inconsistencies in the previous works and introduce our TraiNDSim, a simulation framework that aims to deliver accurate performance evaluations of NDs. For conductance updates, TraiNDSim uses PUM improved with layer-wise normalization, a conductance normalization technique proposed in this work. Through the precise weight update facilitated by layer-wise normalization, the implementation of this technique leads to a scenario where the image classification accuracy of a DNN model trained with the on-chip training simulation of TraiNDSim closely follows those of a DNN model trained through software. Furthermore, TraiNDSim integrates three conductance modeling formulas that are typically used in the device community, notably by refining one of the conventional models to depend solely on nonlinearity. We also implement a compensation technique for conductance clipping through the application of additional voltage pulses, which is only suitable for the bi-directional method of weight representation.

2 CHARACTERISTICS OF NEUROMORPHIC DEVICES AND THEIR IMPACTS

NDs exhibit deviations from the ideal behavior, including nonlinearity, on/off ratio (defined by G_{max} and G_{min} for the highest and lowest conductance, respectively), maximum conductance level, and C2C and D2D variations. Nonlinearity affects the regularity of the conductance change, and on/off ratio represents the range of

DAC '24, June 23-27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3656535

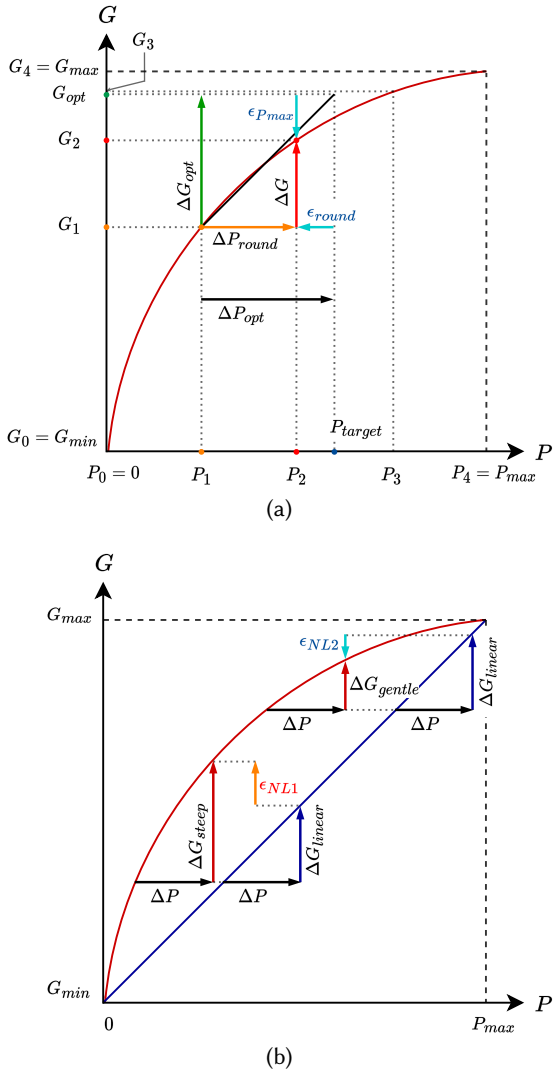

Figure 1. Influences of (a) maximum conductance level and (b) nonlinearity in conductance update procedure

the conductance value, equivalent to the fraction of G_{max} to G_{min}. Moreover, maximum conductance level sets the number of discrete conductance states, termed maximum pulse number (P_{max}). It influences the minimal scale of conductance change corresponding to a single voltage pulse application, thereby affecting the spectrum of attainable conductance values. The performance of ideal ND is achieved by reducing nonlinearity, increasing on/off ratio, and enhancing the number of conductance states. Also, C2C variations introduce immediate conductance distortions on application of voltage pulses [16], while D2D variations influence nonlinearity of NDs. Both variations are often modeled as Gaussian distributions [5].

During the conductance update process, the amount of conductance change is mainly influenced by the maximum conductance level and nonlinearity. Voltage pulses, typically derived assuming a linear LTP/LTD response in the ND, are applied to an inherently nonlinear ND, resulting in a distortion of the expected conductance change. Fig. 1 illustrates how the maximum conductance level and nonlinearity affect the conductance update in NDs. In Fig. 1a, it is assumed that ND has a conductance state value of G_1 before the conductance update. Here, ΔG_{opt}, the optimal value for updating the conductance, is calculated by BP. Ideally, the voltage pulses, ΔP_{opt}, derived from a ΔG_{opt} would be imposed on the ND:

$$\Delta P_{opt} = \frac{P_{max}}{G_{max} - G_{min}} \Delta G_{opt}. \quad (1)$$

Given the ND's discrete conductance states (G_0 to G_4), ΔP_{opt} is rounded to ΔP_{round} for practical application, causing a transition from G_1 to G_2. Fig. 1b shows how nonlinearity affects the conductance change when equal voltage pulses are applied to the ND. For a linear LTP/LTD response, the conductance is updated uniformly to ΔG_{linear}. However, with a nonlinear ND, the conductance update deviates from ΔG_{linear} to ΔG_{steep} or ΔG_{gentle}, depending on the slope of each conductance state.

3 TECHNIQUES OF EXISTING FRAMEWORKS FOR ON-CHIP TRAINING SIMULATION

For DNN simulation using NDs, various simulation frameworks have emerged. In general, these can be categorized into frameworks focused on inference tasks [13, 29] and those handling on-chip training simulation [1, 4, 6, 17, 18, 21, 22]. The partitioning extends to on-chip training simulation frameworks, where weight values are processed digitally [6, 18] or analogically [1, 4, 17, 21, 22]. Our work, which deals with analog on-chip training simulation to improve precision in weight value processing, is related to frameworks such as CrossSim [1], TxSim [22], NeuroSim [4, 17], and AIHWKIT [21]. However, AIHWKIT [21] did not adopt the fitting parameters and their corresponding conductance modeling formulas, which are the characteristics [2, 9] described in Section 2, restricting the examination to CrossSim [1], TxSim [22], and NeuroSim [4, 17]. In this section, we discuss techniques of these simulation frameworks.

3.1 Conductance Update Methods

GUM, adopted in CrossSim [1] and TxSim [22], computes the conductance change directly from the gradient. Although conductance can theoretically be continuous, maintaining stability throughout the range of its value is almost impossible [20]. Due to its inherent instability, it is important to consider its discrete characteristics. However, GUM overlooks this aspect during the conductance update. In contrast, PUM, employed in NeuroSim [4, 17], calculates voltage pulses required for conductance change based on the gradient, and applies these pulses to the ND to update the conductance value. Thus, PUM ensures the conductance update within the discrete states. Although DNN+NeuroSim [17] used PUM, its utilization of WAGE quantization [26] compromises the training process by ignoring the influence of nonlinearity, detailed in Section 4.1.

3.2 Conductance Modeling Formulas

In the field of ND research, conductance modeling is essential, with three primary formulas prevalent to represent trajectories of the conductance value, depending on fabrication techniques and materials. These models are classified into asymmetric and

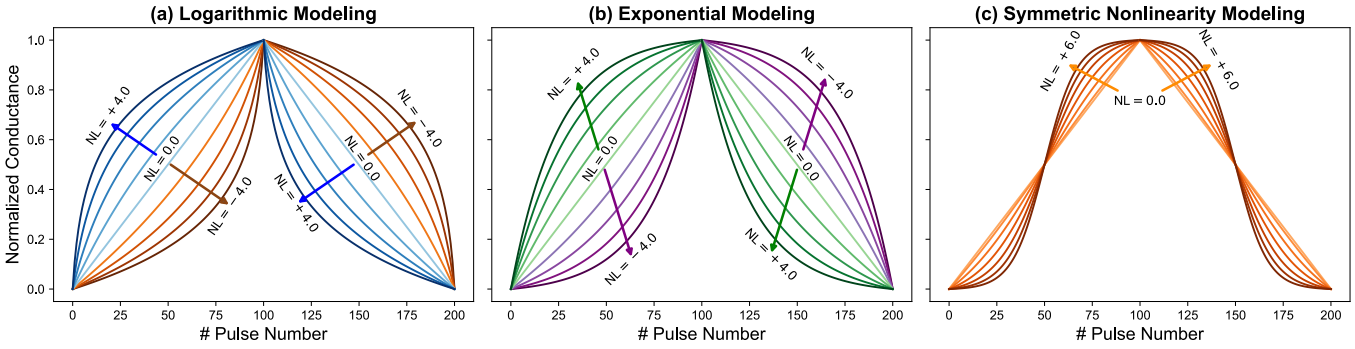

Figure 2. Trajectories (LTP/LTD) depicted through conductance modeling formulas

symmetric nonlinearity groups [1]. Within the former category, the logarithmic [10, 19] and exponential [3] models are prominent. The mathematical representations utilized for conductance modeling are presented below in the context of PUM.

- Logarithmic modeling formula:

$$\mathbf{G_{LTP}}(\mathbf{P}) = G_{min} + \mathbf{C_1} \cdot \ln\left(\frac{e^{\mathbf{NL}} - 1}{P_{max}}\mathbf{P} + 1\right), \tag{2}$$

$$\mathbf{C_1} = \frac{G_{max} - G_{min}}{\mathbf{NL}}. \tag{3}$$

- Exponential modeling formula:

$$\mathbf{G_{LTP}}(\mathbf{P}) = G_{min} + \mathbf{C_2}\left(1 - e^{-\frac{\mathbf{NL}}{P_{max}}\mathbf{P}}\right), \tag{4}$$

$$\mathbf{C_2} = \frac{G_{max} - G_{min}}{1 - e^{-\mathbf{NL}}}. \tag{5}$$

- Symmetric nonlinearity modeling formula:

$$\mathbf{G_{LTP}}(\mathbf{P}) = G_{min} + \mathbf{C_3}\left(\mathbf{D}(\mathbf{P}) - 1\right), \tag{6}$$

$$\mathbf{C_3} = \frac{G_{max} - G_{min}}{e^{\mathbf{NL}} - 1}, \quad \mathbf{D}(\mathbf{P}) = \frac{e^{\mathbf{NL}} + 1}{1 + e^{-\mathbf{NL}\left(\frac{2}{P_{max}}\mathbf{P} - 1\right)}}. \tag{7}$$

In the above equations, **NL** quantifies nonlinearity, while **P** denotes the pulse number for each conductance state. In the foundational methodology proposed by Querlioz *et al.* [19] for logarithmic modeling, two parameters are used: step size and nonlinearity. This work induces logarithmic modeling to a single-parameter model with nonlinearity, which will be further detailed in Section 4.3. The exponential modeling was implemented in all three frameworks, CrossSim [1], TxSim [22], and NeuroSim [4, 17], whereas the symmetric nonlinearity modeling was employed only in CrossSim [1]. These concepts are visualized in Fig. 2.

3.3 Weight Representation Methods

Two widely used methods for the representation of weight values are the uni-directional and bi-directional methods, utilizing single and dual conductances, respectively. The uni-directional method uses a fixed offset, called reference conductance, to represent negative weights by subtracting it from the actual conductances. This offset is typically set at the midpoint between G_{max} and G_{min}. The uni-directional method is expressed as follows:

$$\mathbf{W_{uni}} = \gamma(\mathbf{G} - G_{ref}), \quad G_{ref} = \frac{G_{max} - G_{min}}{2}, \tag{8}$$

where the term γ denotes the normalization scale.

The bi-directional method expresses weight values by the differences between conductances for positive weights and those for negative weights. Employing dual conductances for weight representation expands the spectrum of attainable states for the weight. This method can be mathematically presented as follows:

$$\mathbf{W_{bi}} = \gamma(\mathbf{G_p} - \mathbf{G_n}). \tag{9}$$

3.4 Conductance Normalization Technique

Conductance normalization is essential to map the weight value to the conductance during training. Typically, fixed normalization is used, scaling the conductance by the range between G_{max} and G_{min} to fit weight values in the range of $[-1, 1]$:

$$W_{uni} = \frac{2}{G_{max} - G_{min}}(G - G_{ref}). \tag{10}$$

3.5 Comparison between Existing Frameworks and TraiNDSim

Table 1 presents a comparative analysis of CrossSim [1], TxSim [22], NeuroSim [4, 17], and our framework, TraiNDSim, focusing on the available techniques for on-chip training, network models, and datasets. As mentioned earlier, existing frameworks exhibit inconsistencies with the characteristics of ND in training simulation. CrossSim [1] and TxSim [22] used GUM, which overlooks the impact of the discrete characteristics of conductance. DNN+NeuroSim [17], while using PUM, employed WAGE quantization [26], which leads to ignoring the impact of the nonlinearity of ND. TraiNDSim rectifies these inconsistencies and encompasses all widely known techniques. We also introduce layer-wise normalization to overcome the limitations of fixed normalization.

4 MODELING FRAMEWORK OF TRAINDSIM

Contrary to the approach of using the GUM as in CrossSim [1] and TxSim [22], TraiNDSim employs the PUM to accurately reflect the impact of discrete characteristics of conductance. Although DNN+NeuroSim [17] utilizes PUM, it does not consider the effect of nonlinearity when determining final conductance changes. Consequently, TraiNDSim uses PUM and refines the defectively implemented training techniques identified in DNN+NeuroSim [17]. However, rectifying the flawed training process alone does not

Table 1. Comparison of on-chip training options in different frameworks

	CrossSim [1]	TxSim [22]	NeuroSim [4, 17]	**TraiNDSim**
Networks and Dataset	MLP / MNIST	**MLP, CNN / MNIST, CIFAR-10, CIFAR-100**		
Conductance Update Method	GUM	GUM	PUM + WAGE	**PUM**
Conductance Modeling Formulas	Exp, Sym	Exp	Exp	**Log, Exp, Sym**
Weight Representation Methods	Uni	Uni	Uni	**Uni, Bi**
Conductance Normalization	Fixed*	Not use§	Fixed	**Fixed, Layer-wise†**

*The conductance is normalized by on/off ratio. §As an alternative to normalization, the conductance update process is executed by factoring the gradient with a coefficient called $Scale_{Layer-k}$ [22]. †This is a conductance normalization technique proposed in this work.

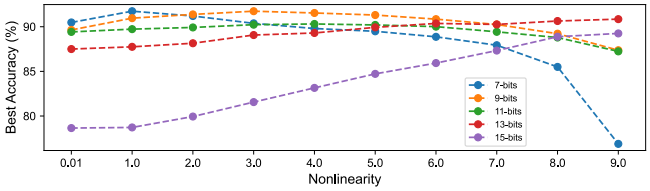

Figure 3. Simulation results of DNN+NeuroSim [17] with fixed GQ bits

ensure successful training due to the limitations of the fixed normalization. Therefore, we introduce layer-wise normalization to enhance the training process. Additionally, TraiNDSim features a unique logarithmic modeling option, defined only by nonlinearity. This approach mitigates the inherent complexity associated with logarithmic modeling, thus reducing the potential for error in constructing conductance modeling. Furthermore, TraiNDSim incorporates a compensation technique for conductance clipping, which is applicable exclusively to the bi-directional method. This technique typically imposes the additional change directly on the conductance, whereas our framework utilizes supplementary voltage pulses in congruence with the conductance change.

4.1 DNN+NeuroSim's WAGE Issue and TraiNDSim's PUM-Only Approach

DNN+NeuroSim [17] introduces inconsistency with actual device behavior by using WAGE quantization [26], which equates conductance changes of nonlinearly characterized conductance with those of linearly characterized conductance. It quantizes weights and gradients based on maximum conductance level, with quantization bits representing the number of conductance states. During the training process, conductance changes are made using the gradient quantization (GQ) bits as 2^{1-bits_g}, which are not derived from BP gradients. With G_{max} and G_{min} normalized to 1 and -1 by fixed normalization, the conductance change is expressed as follows:

$$2^{1-bits_g} = \frac{2}{2^{bits_g}} = \frac{1-(-1)}{P_{max}} = \frac{G_{max}-G_{min}}{P_{max}}. \tag{11}$$

According to Eq. (1), the final term of Eq. (11) implies that NeuroSim [17] updates the conductance assuming linear LTP/LTD. To examine this characteristic of DNN+NeuroSim [17], we trained VGG-8 [24] using CIFAR-10 [12], varying nonlinearity and GQ bits. The results in Fig. 3 show consistent accuracies in most GQ bits, unaffected by changes in nonlinearity.

Recognizing the flawed conductance update process of WAGE quantization [26], TraiNDSim strategically opts out of this approach. Therefore, it exclusively utilizes PUM to ensure a more precise simulation. This strategic choice stems from the understanding that accurately capturing the nonlinear dynamics of conductance changes is crucial for on-chip training. On the other hand, relying solely on PUM led to a decrease in classification accuracy, which is a consequence of accurately mirroring the authentic on-chip training process. The root cause of this accuracy reduction is attributed to the fixed normalization, a challenge that we overcome with our proposed layer-wise normalization technique.

4.2 Layer-wise Normalization

The fixed normalization standardizes the weight range across all layers to a uniform scale $[-1, 1]$. Consequently, this uniformity fixes the minimum change in weights and conductances across all layers at the same level. As previously mentioned, the minimum change in the conductance is determined by the maximum conductance level. In situations where the maximum conductance level is not sufficiently high, the relatively large minimum change, in comparison to the layer's weight range, hinders fine adjustments in layers that require delicate modulation. This scenario increases the potential of deviating from the optimal weight distribution range. Therefore, during the weight update, the potential of incorporating the optimal change is reduced, which can result in a less precise weight update and a subsequent decrease in training accuracy.

Fig. 4 shows the weight distributions across four layers in different instances of the VGG-8 model [24]. Among these four scenarios, an instance shows the weight distribution of a network initialized using Kaiming uniform initialization [7], which sets the initial weights inversely proportional to the size of the IFM per layer. In the VGG-8 model [24], it leads to a narrowed weight distribution range as the layers deepen. Another scenario shows the weight distribution of a network post-trained under baseline conditions—using optimal software without the conductance update process. This baseline scenario reveals the weight distribution closely reflecting initial patterns, suggesting that the optimal weight distribution of each layer generally resembles its initial distribution.

To resolve the issue of the fixed normalization, we introduce layer-wise normalization. This technique sets the range of normalized conductances based on the distribution of initialized weights, defining unique fluctuation boundaries for each layer's weights. Therefore, this approach increases the potential of achieving the optimally trained weight distribution. As the normalized conductance

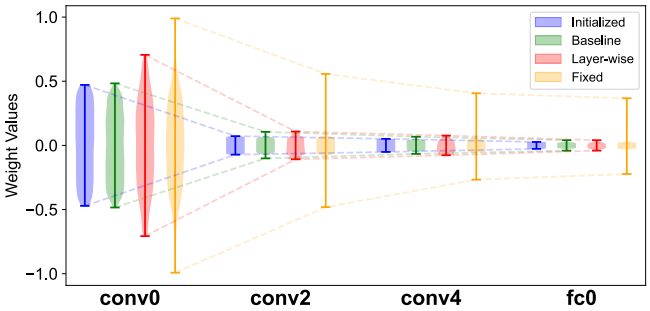

Figure 4. Distributions of weights across four layers in the initialized VGG-8 [24], the network trained under baseline conditions, and the network utilizing fixed and layer-wise normalization, all implemented on the CIFAR-10 dataset [12]

range narrows, the minimum conductance change reduces, enabling a more accurate reflection of ideal weight changes. Typically, the initialized weight range is smaller than $[-1, 1]$, making the weight range with the layer-wise normalization smaller than that with the fixed normalization, leading to a more accurate representation of weight changes. However, if the normalized conductance range becomes too small, it may not match the weight distribution of the baseline network, requiring additional scaling. Taking these aspects into account, the normalization process involves multiplying fixed normalization results by the maximum value of initialized weights and a distribution scale to adjust the normalized conductance range:

$$W_{uni} = \frac{2 \cdot dist\_scale \times init\_W_{max}}{G_{max} - G_{min}}(G - G_{ref}), \tag{12}$$

where the terms $dist\_scale$ and $init\_W_{max}$ represent the distribution scale and the maximum initial weight, respectively. In the case of the network using the layer-wise normalization in Fig. 4, the distribution scale was set to 1.5 for the training process. As a result of training, the weight distributions in most layers closely resemble the baseline weight distributions.

4.3 Derivation of Logarithmic Modeling Formula

This section presents the derivation of the logarithmic modeling formula into single-parameter modeling, introduced by Querlioz *et al.* [19]. Their model interprets the conductance shift in response to a single voltage pulse, based on the conductance state value:

$$\Delta G_{LTP} = \alpha \cdot \exp\left(-NL\frac{G_{LTP}(P) - G_{min}}{G_{max} - G_{min}}\right), \tag{13}$$

$$\Delta G = G(P+1) - G(P). \tag{14}$$

Here, α represents the step size, a key fitting parameter. Kim *et al.* [10] modified this equation by approximating the discrete pulse number P to a continuous form, converting it into a differential equation, and subsequently solving it:

$$G_{LTP}(P) = G_{min} + C_1 \cdot \ln\left(\frac{\alpha}{C_1}P + 1\right), \tag{15}$$

where C_1 is defined in Eq. (3). The derived logarithmic modeling formula is expressed in terms of step size and nonlinearity. These parameters are acquired from the experimental conductance data. However, there are cases where the extracted step size may not correspond appropriately with the data. Under such circumstances, the conductance value obtained from this modeling deviates from the original data, hindering accurate conductance modeling.

In this work, we discover the relationship between step size and nonlinearity. By making the step size a function of nonlinearity, we revise the logarithmic modeling formula to depend solely on nonlinearity. This modification enables the conductance to be modeled exclusively with nonlinearity, leading to error-free conductance modeling. We derive this relationship considering that the conductance reaches G_{max} at the maximum pulse number for the LTP curve. Substituting the maximal pulse number into P in Eq. (15), we get a following equation:

$$G_{max} = G_{min} + C_1 \cdot \ln\left(\frac{\alpha}{C_1}P_{max} + 1\right). \tag{16}$$

By referring to Eq. (3) and rearranging this equation to depict the step size in terms of the nonlinearity, we arrive at the following:

$$\alpha = \frac{G_{max} - G_{min}}{NL \cdot P_{max}}\left(e^{NL} - 1\right) = \frac{C_1}{P_{max}}\left(e^{NL} - 1\right). \tag{17}$$

Table 2. Best accuracies across various networks and datasets observed within the baseline, layer-wise normalization, and fixed normalization strategies

Dataset	MNIST		CIFAR-10		CIFAR-100	
Network	**VGG-8**	**MLP**	**VGG-8**	**MLP**	**VGG-8**	**MLP**
Baseline (%)	99.49	97.42	89.03	58.62	67.27	30.23
Layer-wise (%)	99.31	97.27	88.01	56.86	48.87	24.15
Fixed (%)	98.69	96.43	72.61	51.97	32.36	19.33

4.4 Conductance Compensation Technique in Bi-directional Method

During the conductance update, clipping occurs when conductance values exceed set limits. The uni-directional method, using a single conductance, lacks compensation for clipping. Conversely, the bi-directional method uses dual conductances, allowing compensation for clipping by adjusting the opposite conductance, thus preserving the optimal weight value. Since the conductance change is governed by the amount of voltage pulses applied, TraiNSim applies additional voltage pulses in alignment with the conductance change rather than directly imposing additional conductance change.

5 EXPERIMENTS AND DISCUSSION

In this section, we evaluate the effectiveness of layer-wise normalization and perform a comparative analysis of the simulation results of TraiNDSim and DNN+NeuroSim [17]. Throughout our experiments, we configured the training to conclude at the 50th epoch, with a batch size of 200. The sum of squared errors was employed as the loss function, and stochastic gradient descent with momentum served as the optimizer.

Table 2 compares the training outcomes for multi-layer perceptron (MLP) and VGG-8 [24] on MNIST [28], CIFAR-10, and CIFAR-100 [12] datasets across three scenarios: optimal software training

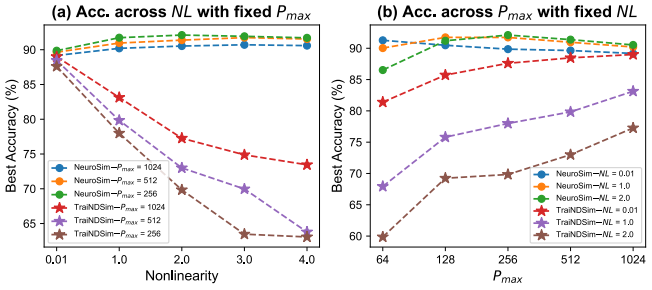

Figure 5. Test accuracies from on-chip training simulations of VGG-8 [24] on CIFAR-10 [12] using TraiNDSim and DNN+NeuroSim [17] under various nonlinearity and maximum conductance levels

(baseline), and on-chip training with fixed or layer-wise normalization. These experiments, using ideal ND parameters (nonlinearity of 0.01, maximum pulse number of 1024, on/off ratio of 15.5/0.5), revealed minor accuracy differences between baseline and fixed normalization in VGG-8 on MNIST. However, compared to baseline, fixed normalization exhibited lower performance for CIFAR-10 and CIFAR-100, while layer-wise normalization demonstrated accuracies that were more closely aligned with baseline.

Fig. 5 shows the simulation results of TraiNDSim and NeuroSim [17] under various nonlinearity and maximum conductance levels, with the aim of evaluating their effectiveness in reflecting the non-idealities of ND. The investigation focused mainly on the effects of nonlinearity and maximum conductance level with a stable on/off ratio of 50/1, excluding the impact of C2C and D2D variations. Section 4.1, DNN+NeuroSim [17] exhibits a limited ability to accurately reflect the impacts of non-ideal characteristics. This limitation is evident from the minimal changes in test accuracies across various scenarios. In contrast, TraiNDSim precisely captures these impacts in all cases, evidenced by higher test accuracy with lower nonlinearity and higher maximum conductance level. Moreover, the clear differentiation among test accuracies in TraiNDSim simulations offers a more precise assessment of the ND performance.

6 CONCLUSION

This work addresses training inadequacies in CrossSim [1], TxSim [22], and NeuroSim [4, 17], proposing the layer-wise normalization for the more accurate weight update, which restricts the weight range by considering the distribution of initialized weights. Our framework integrates several conductance modeling formulas, notably refining the logarithmic modeling to depend solely on nonlinearity. We also have incorporated the bi-directional method of weight representation and implemented the conductance compensation technique. Using TraiNDSim, the comprehensive analysis of the simulation results is conducted to evaluate the effectiveness of the layer-wise normalization. This analysis reveals that the layer-wise normalization exhibits superior effectiveness compared to the fixed normalization. Furthermore, the simulation results from TraiNDSim are compared with those obtained from DNN+NeuroSim [17]. The results of these experiments imply that TraiNDSim provides a more accurate evaluation of the device performance.

ACKNOWLEDGMENTS

This work was partly supported by the National Research Foundation (NRF) grants (RS-2023-00251438, 2022R1A4A3032913), the Institute of Information and Communication Technology Planning & Evaluation (IITP) grants (IITP-2019-0-00421, IITP-2023-2020-0-01821, IITP-2021-0-02068, IITP-2021-0-02052), and the Ministry of Trade, Industry & Energy (MOTIE) grant (RS-2023-00235718).

REFERENCES

[1] S. Agarwal et al. 2016. Resistive memory device requirements for a neural algorithm accelerator. In *IEEE IJCNN*. 929–938.
[2] J. Chen et al. 2019. LiSiO X-based analog memristive synapse for neuromorphic computing. *IEEE Electron Device Letters* 40, 4 (2019), 542–545.
[3] P. Y. Chen et al. 2015. Mitigating effects of non-ideal synaptic device characteristics for on-chip learning. In *IEEE/ACM ICCAD*. 194–199.
[4] P. Y. Chen et al. 2018. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE TCAD* 37, 12 (2018), 3067–3080.
[5] M. H. DeGroot et al. 2012. *Probability and statistics*. Pearson Education.
[6] X. Fei et al. 2020. XB-SIM: A simulation framework for modeling and exploration of ReRAM-based CNN acceleration design. *Tsinghua Science and Technology* 26, 3 (2020), 322–334.
[7] K. He et al. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE ICCV*. 1026–1034.
[8] M. Hu et al. 2018. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials* 30, 9 (2018), 1705914.
[9] M. Jerry et al. 2017. Ferroelectric FET analog synapse for acceleration of deep neural network training. In *IEEE IEDM*. 6–2.
[10] C. H. Kim et al. 2018. Emerging memory technologies for neuromorphic computing. *Nanotechnology* 30, 3 (2018), 032001.
[11] W. Kim et al. 2019. Confined PCM-based analog synaptic devices offering low resistance-drift and 1000 programmable states for deep learning. In *IEEE Symposium on VLSI Technology*. T66–T67.
[12] A. Krizhevsky et al. 2009. Learning multiple layers of features from tiny images. (2009).
[13] C. Lammie et al. 2022. MemTorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing* 485 (2022), 124–133.
[14] Y. LeCun et al. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[15] C. Li et al. 2018. Analogue signal and image processing with large memristor crossbars. *Nature electronics* 1, 1 (2018), 52–59.
[16] W. Q. Pan et al. 2020. Strategies to improve the accuracy of memristor-based convolutional neural networks. *IEEE T-ED* 67, 3 (2020), 895–901.
[17] X. Peng et al. 2020. DNN+ NeuroSim V2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE TCAD* 40, 11 (2020), 2306–2319.
[18] N. L. Prabhu et al. 2022. Neuromorphic In-Memory RRAM NAND/NOR Circuit Performance Analysis in a CNN Training Framework on the Edge for Low Power IoT. *IEEE Access* 10 (2022), 125112–125135.
[19] D. Querlioz et al. 2011. Learning with memristive devices: How should we model their behavior?. In *IEEE/ACM NANOARCH*. 150–156.
[20] M. Rao et al. 2023. Thousands of conductance levels in memristors integrated on CMOS. *Nature* 615, 7954 (2023), 823–829.
[21] M. J. Rasch et al. 2021. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. In *IEEE AICAS*. 1–4.
[22] S. Roy et al. 2021. Txsim: Modeling training of deep neural networks on resistive crossbar systems. *IEEE Transactions on VLSI Systems* 29, 4 (2021), 730–738.
[23] D. E. Rumelhart et al. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.
[24] K. Simonyan et al. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[25] J. Tang et al. 2018. ECRAM as scalable synaptic cell for high-speed, low-power neuromorphic computing. In *IEEE IEDM*. 13–1.
[26] S. Wu et al. 2018. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680* (2018).
[27] W. Wu et al. 2018. A methodology to improve linearity of analog RRAM for neuromorphic computing. In *IEEE symposium on VLSI technology*. 103–104.
[28] L. Yann et al. 2010. *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/
[29] Z. Zhu et al. 2020. MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. In *GLSVLSI*. 83–88.