International Journal of Computer Applications (0975 - 8887)
Volume 186 - No.49, November 2024

Genetic Algorithm-based Training Method for Memristor Neural Networks

Wei Zhang
School of Instrument Science and Opto-Electronics Engineering
Beijing Information Science and Technology University
Beijing
China

Qingtian Zhang
School of Integrated Circuits
Tsinghua University
Beijing
China

Huaqiang Wu
School of Integrated Circuits
Tsinghua University
Beijing
China

ABSTRACT

In recent years, deep learning and large models have significantly advanced artificial intelligence applications in areas such as natural language processing and computer vision. However, as model scales grow, the demand for computing power increases, revealing the limitations of traditional von Neumann architectures. Memristor-based in-memory computing offers a promising alternative, yet neural networks deployed on memristor arrays suffer from accuracy loss due to device non-idealities. To address this, authors introduce a novel genetic algorithm (GA)-based training methodology specifically designed for memristor arrays to enhance neural network performance. authors detail the framework and strategic operations of this approach, supported by empirical validation using a series of lightweight models and demonstrate substantial accuracy improvements. Additionally, authors explore the impact of various hyperparameter settings on model precision. Overall, this approach significantly enhances the accuracy of lightweight neural networks on memristor arrays, with important implications for edge computing environments.

General Terms

Algorithms, Optimization, Image Classification

Keywords

memristor neural networks, in-memory computing, genetic algorithm

1. INTRODUCTION

Deep learning and large-scale pre-trained models have emerged as pivotal technologies within the realm of contemporary artificial intelligence, significantly propelling innovation and advancement across various sectors. By emulating the operational principles of neurons in the human brain, deep neural networks are capable of processing and comprehend highly intricate data patterns, exhibiting exceptional performance in diverse application contexts such as natural language processing, computer vision, and speech recognition [20]. For instance, extensive models like BERT and GPT, which are founded on the Transformer architecture, are able to acquire rich linguistic representations during the unsupervised pre-training phase, attributable to their extensive parameter counts and multi-layered configurations. Subsequently, these models can be fine-tuned to accommodate a range of downstream tasks, demonstrating capabilities that exceed those of traditional methodologies. Nevertheless, the efficacy of these large models is accompanied by a substantial requirement for computational resources. The training of such models necessitates high-performance computing systems and considerable training durations, which not only escalate the costs associated with computational resources but also impose heightened demands on energy consumption.

In the traditional von Neumann architecture, the computation and storage units are designed independently, which often leads to significant performance bottlenecks when dealing with large data sets or complex computational tasks. As a result, the development of neural networks based on this architecture is limited by size and power consumption [34]. In contrast, biological nervous systems do not suffer from these limitations. To address this issue, the researchers proposed an integrated architecture for storage and computation based on non-von Neumann computational methods.

The memristor, introduced by Leon Chua in 1971 [4], and experimentally validated by Hewlett-Packard in 2008 [22], is well-suited for this architecture and offers potential to replace von Neumann systems, particularly in neural networks and big data applications [24]. This architecture opens the door to more intelligent edge and mobile devices. In recent years, considerable advancement has been made in the research and development of various types of neural networks using memristor crossbars, including single- and multi-layer perceptrons [25, 28, 12], spiking neural network [13] and convolutional neural network(CNN) [29], and have demonstrated efficacy in a range of applications, including image recognition [12], image classification [18] and target detection [23].

However, non-idealities like conductance drift, read disturbances, wire parasitics, and device degradation pose challenges, causing inaccuracies and malfunctions [6, 1, 8]. These non-ideal factors significantly degrade the computational accuracy of compute-in-memory systems, thereby imposing substantial limitations on their

applications. Therefore, targeted approaches are required to improve the computational accuracy in compute-in-memory chips.
Current methods address these issues through architectural design, training techniques, algorithm optimization, multi-model integration, and energy efficiency improvements.
Recent developments in mixed-precision in-memory computing architectures have demonstrated significant potential for enhancing the performance and efficiency of neural networks. [11] introduced a mixed-precision architecture that combines a von Neumann machine with a computational memory unit, facilitating iterative refinement of solution accuracy. Additionally, in situ training has been investigated as a strategy to mitigate hardware non-idealities in memristor-based neural networks. [26] illustrated the efficacy of directly mapping convolutional kernels to memristor crossbars within 1T1R arrays, while [3] formulated a training methodology that accounts for inter-device variations to bolster network resilience.
Various approaches have been proposed to enhance the accuracy and robustness of memristor-based neural networks. [9] employed committee machines to improve inference accuracy through ensemble averaging, and [17] introduced methods to alleviate the impacts of conductance state asymmetry and cycle-to-cycle variations. Further contributions by [32] and [16] involved the development of stochastic and adaptive learning techniques and committee machine frameworks, respectively, to address inaccuracies stemming from memristor nonlinearity. [10] also proposed a non-ideal perception training methodology that enhances energy efficiency while preserving accuracy.
In addressing memristor non-idealities during training, [5] devised a backpropagation and gradient accumulation algorithm that achieved high accuracy on the MNIST and CIFAR-10 datasets. [30] introduced the concept of "memristor activity difference energy minimization," which utilizes hardware discrepancies to optimize network parameters.
Recent research has also concentrated on efficiency and precision. [27] developed a mixed-precision training method that improves the efficiency of vector-matrix multiplication while maintaining high-precision weights, resulting in notable accuracy with the LeNet model. [33] presented a generalized algorithmic framework, STELLAR, which addresses multi-scale error sources within the memristor storage-computation paradigm, providing techniques to reduce energy consumption and enhance learning accuracy.
Despite some success, these approaches are limited by their dependence on specific hardware configurations, computational complexity, and the diverse nature of non-idealities, which require further refinement. In light of these considerations, this paper proposes a methodology for training a neural network for memristors utilizing a genetic algorithm.
Genetic algorithm, known for their global search capabilities, are effective in optimizing neural networks and can be executed via matrix-vector multiplication in memristor arrays [31]. Hence, a novel training method for memristor neural networks, named the genetic algorithm training method, is proposed. This method integrates the fitness function of genetic algorithms with the conventional training process of memristor neural networks. Special selection and mutation strategies tailored for memristor neural networks are employed, enabling the training method to leverage the comprehensive search capabilities of genetic algorithms while remaining well-suited to the unique properties of memristor-based architectures. The genetic training method has demonstrated promising results when applied to ProtoNet [21], MobileNet-v2 [19], and MicroNet [14].

The remainder of this paper is organized as follows: Section II reviews current methods for mitigating memristor non-idealities, Section III presents the proposed algorithm framework and implementation, Section IV demonstrates the algorithm's effectiveness through experiments on ProtoNet, MobileNet-V2, and MicroNet, and the final section concludes the paper.

2. METHODS

The article introduces a new method employing genetic algorithms to achieve robust weight parameters, thereby enhancing model accuracy. This approach tackles non-idealities present in memristor systems by optimizing quantized models. In a software simulation, artificial noise mimics hardware non-idealities' impact, with Gaussian-distributed noise added to weight matrices during convolutions [2]. Figure 1 depicts GA integration for memristor array weight deployment, applying selection, crossover, and mutation. Post-GA, weights undergo neural network backpropagation adjustments before deployment. Figure 2 details the method's implementation.
During initialization, a quantized model with added noise is prepared, setting the model's inference accuracy as the baseline. Parameters including population size, number of iterations, and genetic operations are initialized. Each population member, representing a one-dimensional variable of all weight parameters, is initialized. Different members represent varying weight parameters. Two initialization strategies are provided: 1) "origin", where all members match the baseline model's weights; 2) "large", where one member aligns with the baseline, and others randomly generate weights within the ranges of two reference models (e.g., quantized models with 5% and 10% noise).
The genetic algorithm utilized in this study primarily uses the open-source PyGAD library for PyTorch. In practice, key methods are tailored to meet specific requirements, including adjustments to the fitness function, selection method, and mutation method.
Fitness Function: The detailed flow is illustrated in Figure 3. In genetic algorithms, the one-dimensional variable of all model weights is called a solution. Initially, each solution's parameters are loaded into the model, followed by quantization training. This updates the model parameters through neural network backpropagation. Next, an inference operation is conducted, and the inference accuracy serves as the solution's fitness value. If a solution achieves the best inference accuracy in both the current and historical generations, the model parameters are recorded and updated.
Selection: The PyGAD library offers various selection techniques, and this article focuses on the roulette selection method. This method first calculates the probability of selecting each individual based on their fitness value. In PyGAD, this probability is calculated by dividing an individual's fitness value by the sum of all individuals' fitness values. However, experiments showed that differences in fitness values can be minimal, potentially obscuring distinctions between individuals. To refine this differentiation, the approach applies a sigmoid function on the ratio of an individual's fitness value to the total fitness, thereby enhancing the selection precision.
Crossover: The PyGAD library offers a variety of crossover methods, all of which were successfully implemented in the experiments. Therefore, the built-in methodologies of the PyGAD library were chosen to be utilized. In the experiments detailed in this paper, the two-point crossover method was employed.
Mutation: Experimental results indicated that the mutation process significantly affects outcomes. This paper discards the mutation method provided by the PyGAD library in favor of two new

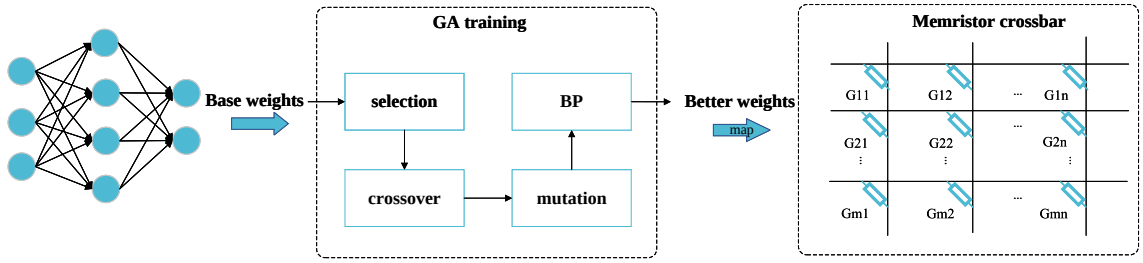

Fig. 1: **GA Training Method and algorithm flowchart.** The position of the GA training method. The network model's weights are input as they are deployed onto the memristor array. Optimal weights, obtained via GA training, are then deployed into the array.

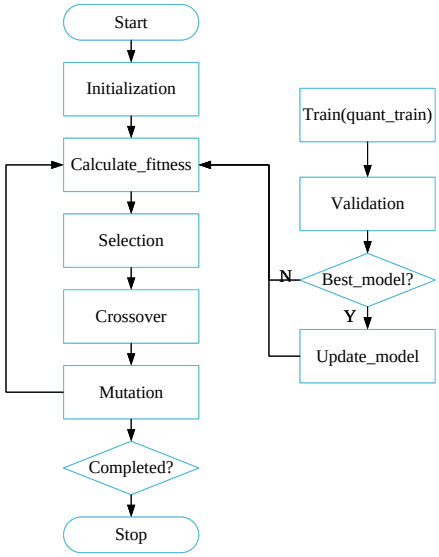

Fig. 2: Flowchart of GA Training Method.

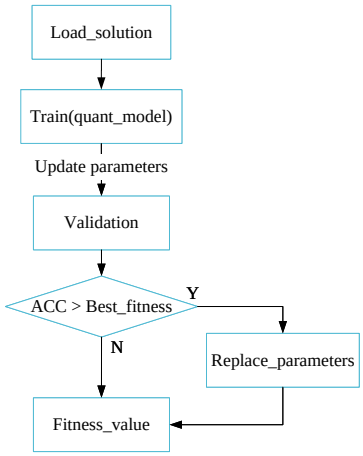

Fig. 3: Flow of fitness function.

methods. The first is a random noise addition strategy, involving adding 5% noise to individual genes. The second method, the random max-min strategy, uses two reference models, such as quantization models with 5% and 10% noise. In this method, the gene at the mutation position is replaced with a random number within the max-min range of the two models at that gene's layer. Algorithm 1 details this process.

After calculating the fitness values and applying the selection, crossover, and mutation strategies, a new population is generated. This process is repeated until the termination condition is met, which, in this study, is defined by a specified number of iterations.

Algorithm 1 Mutation Function

Input: Population of current generation, MutationRate, reference model1 M1, reference model2 M2
Output: Population after mutation
for each individual in population **do**
  for each gene in individual **do**
    R ← RANDOM()
    if $R < MutationRate$ **then**
      obtain the maximum weight G1Max of the layer where the gene is located in M1;
      obtain the minimum weight G1Min of the layer where the gene is located in M1;
      obtain the maximum weight G2Max of the layer where the gene is located in M2;
      obtain the minimum weight G2Min of the layer where the gene is located in M2;
      gene ← RANDOM(MIN(G1Min, G2Min), MAX(G1Max, G2Max));
    end if
  end for
end for
return population

3. EXPERIMENTS AND RESULTS

The methodology and all experiments presented in this paper are implemented in a software environment. In order to simulate the impact of the non-idealities of the hardware system on model training, a simulation approach is employed that introduces noise to the model weights. Given the considerable number of model parameters and floating-point data, which are not readily deployable directly into a memristor array, the methodology proposed in this paper is tailored to quantized models. Due to the large number of model parameters and floating-point data unsuitable for direct

Table 1. : Experiments and Results

TASK	BASELINE	GAT
ProtoNet-MNIST	73.93%	81.99%
ProtoNet-FashionMNIST	78.35%	83.81%
MobileNetV2-Cifar-10	77.91%	83.18%
MicroNet-Cifar-100	73.90%	76.19%

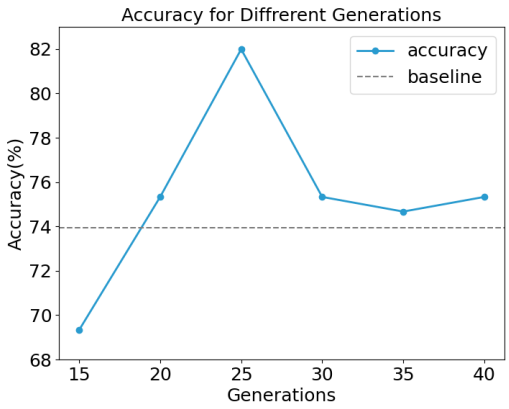

Fig. 4: **Model accuracy at different generations.** At this case, population size is set to 10, the crossover probability is set to 0.8, and the mutation probability is set to 0.01.

deployment in memristor arrays, this paper's methodology is designed for quantized models. Experiments will be conducted on the ProtoNet model with the MNIST dataset and FashionMNIST dataset, MobileNet with CIFAR-10, and MicroNet with CIFAR-100, with the results summarized in Table 1. In the table, BASELINE denotes the accuracy of the reference quantization model, while GAT denotes the accuracy achieved using the GA training method. For instance, the impact of different hyperparameter configurations on ProtoNet's accuracy with MNIST will be evaluated. To validate ProtoNet's effectiveness, experiments on MNIST will extend the reference quantization model [21] by incorporating noisy weights and employing the GA training method. The inference accuracy of the noise-added quantized model serves as a benchmark for the GA training method.

During training, five categories of data were randomly selected from the MNIST dataset, with 20 additional random samples per category. Five samples from each category were assigned to the support set, and the remaining 15 to the query set. The model's weights were then quantized to 2 bits, and had 5% noise added. Evaluated on a test set of 10 categories, the model achieved an inference accuracy of 73.93% after quantization and additive noise training.

In the GA training, the same quantization noise and dataset processing methods as the benchmark model were used. The main parameters of the genetic algorithm were set as follows: 25 iterations, a population size of 10, roulette wheel selection, two-point crossover with a probability of 0.8, and a random max-min mutation strategy with a variance probability of 0.01. Applying this training method to the MNIST test set achieved over 80% accuracy, representing approximately an 8% improvement over the benchmark model.

Additionally, this paper examines how changes in parameter configurations affect model precision. In genetic algorithms, the number of iterations and population size are critical parameters influencing training outcomes. While theoretically increasing these parameters should lead to better searches, experimental observations (Figure 4) show that model accuracy peaks at a certain number of iterations and may decline with further increases. Thus, determining the optimal number of iterations is essential. Regarding population size, experimental results suggest that a larger size correlates with higher model accuracy, as shown in Figure 5.

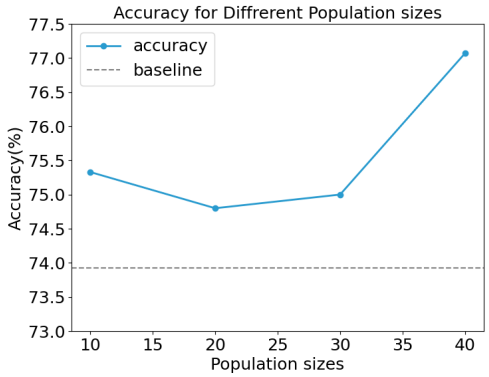

Fig. 5: **Model accuracy at different population sizes.** At this case, the number of iterations is set to 30, the crossover probability is set to 0.8, and the mutation probability is set to 0.01.

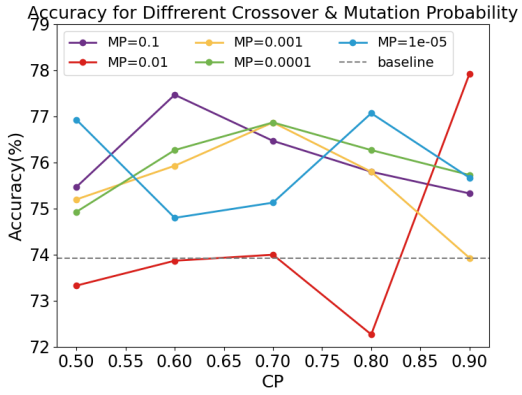

Fig. 6: **Model accuracy at different crossover and mutation probability.** MP represents mutation probability, and CP represents crossover probability. At this case, the number of iterations is set to 25, and a population size is set to 10.

It is also crucial to consider the settings for selection probability and crossover probability in genetic algorithms. Figure 6 illustrates the impact of varying these parameters on model accuracy. Here, CP and MP denote crossover probability and mutation probability, respectively. Experimental results show that a crossover probability of 0.6 generally leads to lower accuracy, with 0.7 or higher being more suitable. Among five mutation probabilities tested, 0.001 is found to be the most appropriate.

It is important to recognize that different tasks have unique characteristics, requiring task-specific optimization of hyperparameters. The effects of various population initialization methods and the impact of genetic strategies on the model are illustrated in Figure 7.

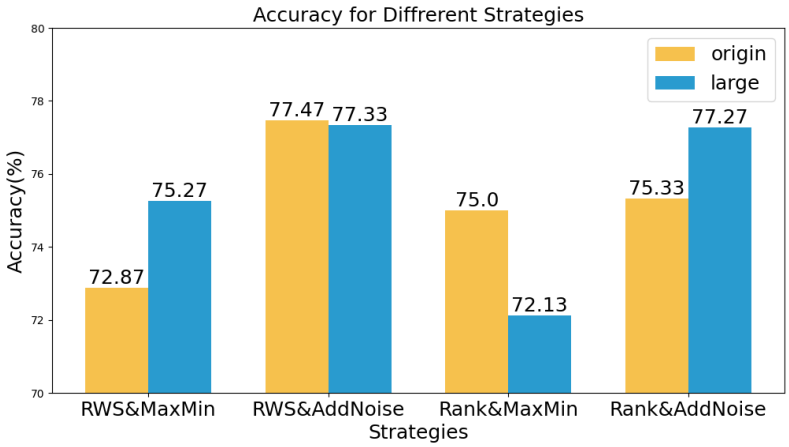

Fig. 7: **Model accuracy at different strategies.** Origin and Large represent two population initialization methods. Origin indicates that all individuals are identical and set to the weight parameters of the baseline model, while Large retains one individual as the reference model's parameters and initializes the rest with added noise based on the reference model. The horizontal axis represents combinations of two selection strategies and two mutation strategies. Here, RWS stands for the roulette wheel selection method, Rank for the ranking selection method, MaxMin for the maximum-minimum mutation method, and AddNoise for the random noise addition mutation method.

Figure 7 illustrates the impact of different strategies on model accuracy under two population initialization methods. Here, RWS denotes the roulette selection method, Rank denotes the ranked selection method, and MaxMin denotes the randomized max-min variation strategy. The AddNoise strategy refers to a random noise addition mutation strategy. The results indicate that the roulette selection strategy and the random noise mutation strategy are more effective than the other strategies, regardless of the initialization method. However, when considering the initialization method alone, there does not seem to be a clear pattern.

This study further investigates the efficacy of ProtoNet utilizing the FashionMNIST dataset. During training, five categories of data were randomly selected from the FashionMNIST dataset, with 20 additional random samples per category. Five samples from each category were assigned to the support set, and the remaining 15 to the query set. The model's weights were then quantized to 3 bits, and had 20% noise added. Evaluated on a test set of 10 categories, the model achieved an inference accuracy of 78.35% after quantization and additive noise training. In the GA training, the same quantization noise and dataset processing methods as the benchmark model were used. The main parameters of the genetic algorithm were set as follows: 25 iterations, a population size of 10, roulette wheel selection, two-point crossover with a probability of 0.8, and a random max-min mutation strategy with a variance probability of 0.01. Applying this training method to the FashionMNIST test set achieved over 83.81% accuracy, representing approximately an 5% improvement over the benchmark model.

To validate the effectiveness of the MobileNetV2 model, experiments will be conducted on the CIFAR-10 dataset. This experiment uses a quantization model obtained through quantization-aware training with an adaptive core set selection method, as described in reference [7]. During training, a 50% subset of the data is selected for 2-bit quantization at each iteration, with 5% noise added to the weights. The model's inference accuracy on the full test set is 77.91%.

Using the GA training method, the same quantization plus noise approach, and dataset settings were applied. Key parameters for the genetic algorithm included 5 iterations, a population of 10 individuals, ranked selection, two-point crossover with a probability of 0.8, and a random max-min variance strategy was employed with a variance probability of 0.001. Applying this training method to the CIFAR-10 test set yielded an accuracy of 83.18%, representing a 5% improvement over the baseline.

The experiment for the MicroNet model is based on the quantized model proposed by Arturo Marban et al., which uses an entropy-constrained training triangularization method, as described in reference [15]. This method involves creating a super network by scaling the pre-trained model's size, followed by simultaneous pruning (using entropy constraints) and quantization (assigning ternary values layer by layer) during training, resulting in a sparse ternary network. Using this ternary network, inference was performed on the CIFAR-100 test set with 5% noise added to the weights, achieving an accuracy of 73.90%.

For the GA training method, the same model architecture and quantization with noise addition were used. The genetic algorithm parameters were set as follows: five iterations, a population size of 10, a ranking selection strategy, a two-point crossover strategy with a probability of 0.8, and a random max-min mutation strategy with a variance probability of 0.001. Applying this training method to the CIFAR-100 test set resulted in an accuracy of 76.19%, representing a 2% improvement over the baseline.

4. CONCLUSIONS

In conclusion, enhancing the intelligence of edge devices necessitates the implementation of more sophisticated network models. Memristors have emerged as pivotal components for this purpose, owing to their low power consumption and compact dimensions. Nevertheless, the efficacy of current research is constrained by reliance on particular hardware configurations, computational complexity, and various non-ideal characteristics, all of which warrant further refinement. This paper introduces a robust solution to the prevalent issue of accuracy degradation in memristor neural networks due to hardware imperfections. By implementing genetic al-

gorithms training method, the adverse effects of non-ideal device characteristics on neural network performance were effectively countered. Comprehensive evaluations were performed on various models, including ProtoNet, MobileNetV2, and MicroNet, which demonstrated notable enhancements in accuracy. In particular, ProtoNet exhibited an accuracy increase of nearly 8% on the MNIST dataset and approximately 5% on the FashionMNIST dataset. MobileNetV2 experienced an approximate 5% improvement in accuracy on the CIFAR-10 dataset, whereas MicroNet achieved an accuracy enhancement of about 2% on the CIFAR-100 dataset. Additionally, the analysis of the impact of various hyperparameters provides further insights into optimizing performance. The results affirm that the proposed approach not only enhances the accuracy of memristor-based neural networks but also facilitates their practical deployment on edge devices, thus contributing valuable insights for ongoing research in this area.

5. ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (Grant No. U2341221).

6. REFERENCES

[1] Stefano Ambrogio, M Gallot, Katherine Spoon, Hsinyu Tsai, Charles Mackin, M Wesson, Sanjay Kariyappa, Pritish Narayanan, C-C Liu, A Kumar, et al. Reducing the impact of phase-change memory conductance drift on the inference of large-scale hardware neural networks. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 6–1. IEEE, 2019.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Guillem Boquet, Edwar Macias, Antoni Morell, Javier Serrano, Enrique Miranda, and Jose Lopez Vicario. Offline training for memristor-based neural networks. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1547–1551. IEEE, 2021.

[4] Leon Chua. Memristor-the missing circuit element. *IEEE Transactions on circuit theory*, 18(5):507–519, 1971.

[5] Shuai Dong, Yihong Chen, Zhen Fan, Kaihui Chen, Minghui Qin, Min Zeng, Xubing Lu, Guofu Zhou, Xingsen Gao, and Jun-Ming Liu. A backpropagation with gradient accumulation algorithm capable of tolerating memristor non-idealities for training memristive neural networks. *Neurocomputing*, 494:89–103, 2022.

[6] Anteneh Gebregiorgis, Abhairaj Singh, Sumit Diware, Rajendra Bishnoi, and Said Hamdioui. Dealing with non-idealities in memristor based computation-in-memory designs. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE, 2022.

[7] Xijie Huang, Zechun Liu, Shih-Yang Liu, and Kwang-Ting Cheng. Efficient quantization-aware training with adaptive coreset selection. *arXiv preprint arXiv:2306.07215*, 2023.

[8] YeonJoo Jeong, Mohammed A Zidan, and Wei D Lu. Parasitic effect analysis in memristor-array-based neuromorphic systems. *IEEE Transactions on Nanotechnology*, 17(1):184–193, 2017.

[9] Dovydas Joksas, Pedro Freitas, Zheng Chai, Wing H Ng, Mark Buckwell, C Li, WD Zhang, Q Xia, AJ Kenyon, and A Mehonic. Committee machines—a universal method to deal with non-idealities in memristor-based neural networks. *Nature communications*, 11(1):4273, 2020.

[10] Dovydas Joksas, Erwei Wang, Nikolaos Barmpatsalos, Wing H Ng, Anthony J Kenyon, George A Constantinides, and Adnan Mehonic. Nonideality-aware training for accurate and robust low-power memristive neural networks. *Advanced Science*, 9(17):2105784, 2022.

[11] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. Mixed-precision in-memory computing. *Nature Electronics*, 1(4):246–253, 2018.

[12] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications*, 9(1):2385, 2018.

[13] Junrui Li, Zhekang Dong, Li Luo, Shukai Duan, and Lidan Wang. A novel versatile window function for memristor model with application in spiking neural network. *Neurocomputing*, 405:239–246, 2020.

[14] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Lu Yuan, Zicheng Liu, Lei Zhang, and Nuno Vasconcelos. Micronet: Improving image recognition with extremely low flops. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 468–477, 2021.

[15] Arturo Marban, Daniel Becking, Simon Wiedemann, and Wojciech Samek. Learning sparse & ternary neural networks with entropy-constrained trained ternarization (ec2t). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 722–723, 2020.

[16] Adnan Mehonic, Dovydas Joksas, Nikolaos Barmpatsalos, Wing H Ng, Anthony J Kenyon, Erwei Wang, and George Constantinides. Mitigating non-idealities of memristive-based artificial neural networks-an algorithmic approach. In *2022 6th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, pages 399–401. IEEE, 2022.

[17] Wen-Qian Pan, Jia Chen, Rui Kuang, Yi Li, Yu-Hui He, Gui-Rong Feng, Nian Duan, Ting-Chang Chang, and Xiang-Shui Miao. Strategies to improve the accuracy of memristor-based convolutional neural networks. *IEEE Transactions on Electron Devices*, 67(3):895–901, 2020.

[18] Huanhuan Ran, Shiping Wen, Shiqin Wang, Yuting Cao, Pan Zhou, and Tingwen Huang. Memristor-based edge computing of shufflenetv2 for image classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(8):1701–1710, 2020.

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[20] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE access*, 7:53040–53065, 2019.

[21] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[22] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

[23] Junwei Sun, Xiao Xiao, Peng Liu, and Yanfeng Wang. Multiple target recognition and position identification circuit based on memristor. *AEU-International Journal of Electronics and Communications*, 151:154223, 2022.

[24] Kaixuan Sun, Jingsheng Chen, and Xiaobing Yan. The future of memristors: Materials engineering and neural networks. *Advanced Functional Materials*, 31(8):2006773, 2021.

[25] Yaoyuan Wang, Shuang Wu, Lei Tian, and Luping Shi. Ssm: a high-performance scheme for in situ training of imprecise memristor neural networks. *Neurocomputing*, 407:270–280, 2020.

[26] Zhongrui Wang, Can Li, Peng Lin, Mingyi Rao, Yongyang Nie, Wenhao Song, Qinru Qiu, Yunning Li, Peng Yan, John Paul Strachan, et al. In situ training of feed-forward and recurrent convolutional memristor networks. *Nature Machine Intelligence*, 1(9):434–442, 2019.

[27] Yuting Wu, Qiwen Wang, Ziyu Wang, Xinxin Wang, Buvna Ayyagari, Siddarth Krishnan, Michael Chudzik, and Wei D Lu. Bulk-switching memristor-based compute-in-memory module for deep neural network training. *Advanced Materials*, 35(46):2305465, 2023.

[28] Le Yang, Zhigang Zeng, and Xinming Shi. A memristor-based neural network circuit with synchronous weight adjustment. *Neurocomputing*, 363:114–124, 2019.

[29] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.

[30] Su-in Yi, Jack D Kendall, R Stanley Williams, and Suhas Kumar. Activity-difference training of deep neural networks using memristor crossbars. *Nature Electronics*, 6(1):45–51, 2023.

[31] Yongbin Yu, Jiehong Mo, Quanxin Deng, Chen Zhou, Biao Li, Xiangxiang Wang, Nijing Yang, Qian Tang, and Xiao Feng. Memristor parallel computing for a matrix-friendly genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 26(5):901–910, 2022.

[32] Wei Zhang, Lunshuai Pan, Xuelong Yan, Guangchao Zhao, Hong Chen, Xingli Wang, Beng Kang Tay, Gaokuo Zhong, Jiangyu Li, and Mingqiang Huang. Hardware-friendly stochastic and adaptive learning in memristor convolutional neural networks. *Advanced Intelligent Systems*, 3(9):2100041, 2021.

[33] Wenbin Zhang, Peng Yao, Bin Gao, Qi Liu, Dong Wu, Qingtian Zhang, Yuankun Li, Qi Qin, Jiaming Li, Zhenhua Zhu, et al. Edge learning using a fully integrated neuro-inspired memristor chip. *Science*, 381(6663):1205–1211, 2023.

[34] Xingqi Zou, Sheng Xu, Xiaoming Chen, Liang Yan, and Yinhe Han. Breaking the von neumann bottleneck: architecture-level processing-in-memory technology. *Science China Information Sciences*, 64(6):160404, 2021.