Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 2797 – 2807

doi:10.3906/elk-1911-77

Improving the efficiency of DNN hardware accelerator by replacing digital feature extractor with an imprecise neuromorphic hardware

Majid MOHAMMADIRAD, Omid SOJODISHIJANI*
Department of Computer Engineering, Faculty of Computer and
Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

Received: 14.11.2019 • **Accepted/Published Online:** 14.05.2020 • **Final Version:** 25.09.2020

Abstract: Mixed-signal in-memory computation can drastically improve the efficiency of the hardware implementing machine learning (ML) algorithms by (i) removing the need to fetch neural network parameters from internal or external memory and (ii) performing a large number of multiply-accumulate operations in parallel. However, this boost in efficiency comes with some disadvantages. Among them, the inability to precisely program nonvolatile memory devices (NVM) with neural network parameters and sensitivity to noise prevent the mixed-signal hardware to perform a precise and deterministic computation. Unfortunately, these hardware-specific errors can get magnified while propagating along with the layers of the deep neural network. In this paper, we show that the inability to implement parameters of the already trained network with enough precision can completely stop the network from performing any meaningful operation. However, even at this level of degradation, the feature extractor section of the network still extracts enough information from which an acceptable level of performance can be achieved by just retraining the last classification layers of the network. Our results suggest that instead of just blindly trying to implement software algorithms in hardware as precisely as possible, it might be more efficient to implement neural networks with imperfect devices and circuits and let the network itself compensate for these imprecise computations by only retraining few layers.

Key words: Artificial neural networks, memristor, in-memory computation, convolutional neural networks, imprecise computation, fault tolerance

1. Introduction

Deep neural networks (DNN) are gaining more and more attention every day, and nowadays can be found in so many applications such as object detection [1, 2] autonomous driving [3], speech recognition [4],a gaming [5], image recognition and segmentation [6], etc. However, most of these applications depend on the computing power of cloud centers mainly because of the high computational complexity of machine learning algorithms. Since using the 250 watts graphics processing unit (GPU) on the IoT device to deploy practical-size AI at the edge is not a feasible solution [7], there has been a new wave of efforts in developing better and more efficient hardware accelerators for DNN. Although some improvements can be achieved through the complementary algorithms like weight quantization [8] or network pruning [9], but most dramatic improvements are still needed to bring power consumption down to the level of IoT devices. Among the possible solution, neuromorphic computing systems developed to accelerate the execution of DNN algorithms have demonstrated very promising performances in terms of power and speed in recent years [10–13]. These systems have recognized the memory bottleneck of

*Correspondence: o_sojoodi@qiau.ac.ir

von Neumann architecture in digital hardware and have tried to solve it through the concept called in-memory computation. In this concept, instead of moving the network parameters back and forth between the external memory and processing units, the network parameters are stored inside the memory of nonvolatile memory devices in analog form and the same physical devices are used to perform the computations of the network. Although any type of nonvolatile memory devices can be used for this purpose, floating-gate transistors and memristors are the most popular ones mainly because of their good analog behaviors and properties [12, 13].

Despite all the advantages, neuromorphic computing systems have their own drawbacks. The most important problem is the resolution of weights that can be stored in nonvolatile memory devices. Although in digital implementation reaching any weight precision is pretty much straightforward, the accuracy of weights stored in analog nonvolatile devices can barely reach 6 to 8 bits [14]. Reducing weight precision can degrade the functional performance of the network and impact the practicality of the neuromorphic systems in real-world applications.

In this paper, we try to understand the effect of having imprecise weights on the performance of neuromorphic hardware accelerating the execution of an already trained DNN. We show here that there is a point in the accuracy of deep neural network parameters up to which the network can tolerate faulty or imprecise network parameters. By passing this point, the performance of the network drops sharply almost to the untrained network. Unfortunately, this key switching point is still within the range of programming accuracies of nonvolatile memory devices demonstrated to date [10–12] meaning that just importing the weights of the trained network into the neuromorphic system may not lead into any acceptable performance especially when dealing with deep networks. Although there is always hope in the development of more robust and precisely programmable devices in the future, the reader should understand that its chance is very slim due to the fact that the switching behavior of these nonvolatile memory devices is highly stochastic [10].

Several methods can be considered to overcome this problem. For example, by using multiple memory devices rather than one (or two in the case of differential implementation) to implement each individual weight, weight precision can be increased to any desired level. However, this will waste a huge area and remove most of the advantages that could have been achieved with in-memory computation. In situ training is another method of dealing with imprecise or faulty hardware in which the network is retrained based on the information extracted from the hardware hoping that retraining can take care of the effects of the imprecise hardware automatically [11]. Although it sounds promising, this method needs (i) full characterization of the hardware and (ii) full retraining of the whole network, which can be computationally expensive and impractical at large scale.

Our proposed method for a more practical system is a mixed-signal solution in which each network is divided into two sections: (i) feature extractor and (ii) knowledge extractor or detector. While the first early layers of any deep neural networks are responsible for extracting features, the last fully connected layers are using the extracted features for decision making and mapping the inputs to one of the possible outputs. Interestingly, it has been demonstrated that through the process called transfer learning [15], we can use the exact feature extractor trained on one dataset in a completely different problem without retraining. This clearly implies that the accuracy of the feature extractor part of the deep neural network is not the key deciding factor in the overall performance of the network. Using this observation, our solution proposes to implement the feature extractor part of neural network with neuromorphic hardware consisting of imprecise nonvolatile memory devices and implement the last layers with digital or more accurate analog circuits. Then, instead of retraining the whole network, we will just retrain the last layers or knowledge extractor section of the neural network using the inaccurate outputs of the feature extractor. In this way, we can take advantage of the high efficiency of

neuromorphic hardware when implementing the computationally expensive part of neural network (i.e. feature extractors) and still reach acceptable performance by performing light and fast retraining on a few last layers of the network implemented with more accurate hardware. As our simulation results show, an indistinguishable level of performance can be achieved even when the programming accuracy of the utilized nonvolatile memory devices in the neuromorphic hardware barely reaches 2–3 bits.

The authors would like to acknowledge their awareness of the necessity of expanding the current study to larger and deeper neural networks trained with more sophisticated training algorithms (like using data augmentation) which can generate better classification performance than those presented in this paper. However, these network architectures are not included in this study due to the lack of access to the cluster of GPUs which is required for training large networks. Moreover, the goal of this paper is not to surpass the highest classification performance reported for this dataset. Instead, our focus here is to show that for any given neural network, there is a possibility of reaching very high hardware efficiency by performing the significant portion of computations of the inference phase of neural networks using imprecise in-memory computing hardware and still achieve good performance. We believe that the proposed method is valid for other network architectures and applications not covered in this paper because solving more complicated problems like classifying ImageNet dataset requires more powerful networks like ResNet [16] or Yolo [17] which still would have high tolerance to imprecise computation due to their large capacity (comping from the large number of parameters they have.)

The rest of the paper is organized as follows: In Section 2 we describe the structures and characteristics of the networks we have studied their sensitivity to imprecise weights in this paper. We will also explain how we have modeled the imprecise weights of neuromorphic hardware for software simulation. The results of the sensitivity study on multi-layer, fully-connected and conventional networks to noisy weights are presented in Section 3 using MNIST and CIFAR10 datasets. Here we also show how these degraded results can be improved by retraining a last layer of the network. Further discussions that can be found in Section 4 conclude the paper.

2. Network architectures considered for this study

In order to validate our proposed solution, two different network architectures, i.e. fully-connected and convolutional networks, are considered as shown in Figure 1. Each of these networks has several intermediate or hidden layers, allowing them to be categorized as deep neural networks. The exact specifications of each of these architectures are explained below.

2.1. Fully-connected neural network

The fully-connected network shown in Figure 1a is used for the classification of the MNIST dataset so it will always have 28×28 inputs and 10 outputs. All layers except the last layer of the network use the tansig activation function. For any original fully-connected network with any number of hidden layers, we will assume that the last layer of the network is the knowledge extractor or detector layer and the remaining layers of the network belong to the feature extractor part of the network. In this way, after training the original network, we will separate the feature extractor part, convert precise trained parameters into imprecise weights similar to how they will be programmed in neuromorphic hardware (i.e. by adding noise to the weights), and then connect altered feature extractor to the new knowledge extractor network. This new knowledge extractor network can be the detector used in the original network, or can be a new fully-connected network with one or more hidden layers. Now that the network is created, only the detector part of the new network will be trained or retrained (if we are initializing the network with the parameters of the original network) to improve the classification

performance of the new network as much as possible. For this part, our original network can have one or two hidden layers and the added feature extractor can have zero or one hidden layer. For each of the hidden layers of the original network, or the added knowledge extractor, we considered different numbers of hidden neurons so we would be able to see the effect of network capacity with the ability of the retrained network to tolerate faults and imprecise computation.

2.2. Convolutional neural network

It is a known fact that fully connected layers are much less sensitive to imprecise weights because of the huge redundancy which exists in these layers. One the other hand, when moving toward deep convolutional networks, we expect the sensitivity of the network to imprecise weight increases because of two reasons. First of all, these layers might have less capacity due to their smaller number of parameters. Secondly, computation errors can propagate and get magnified as the signal passes through the layers of the deep network. However, we want to show that the reader should not jump into the conclusion right away because convolutional layers, usually extract local features compared to fully-connected layers which extract global features. As a result, we will show that the poorly implemented feature extractor, extracts enough information with which we can still squeeze an acceptable level of performance from the network.

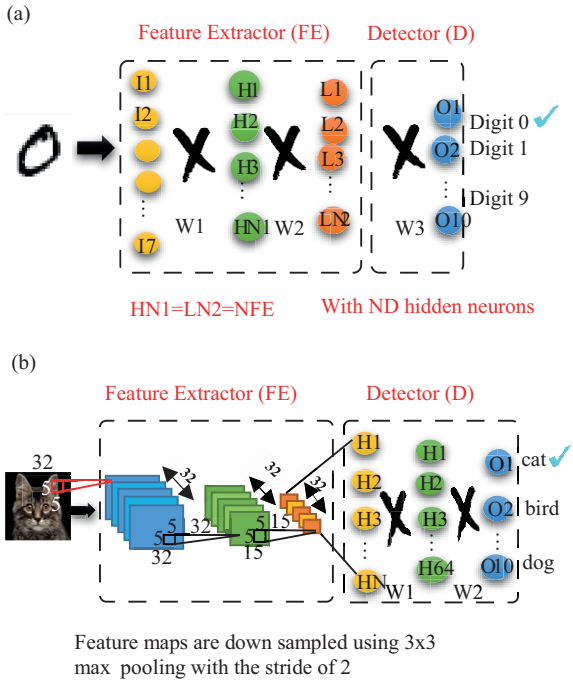


Figure 1. Internal architectures of the network studied in this paper. (a) Multilayer fully-connected network. (b) Convolutional neural network. In the fully-connected network, N_{FE} is the number of neurons in the hidden layers of the feature extractor portion of the network and N_D shows the number of neurons in the detector section (when there is a hidden layer). Networks showed in (a) and (b) are trained on MNIST and CIFAR10 datasets, respectively.

In order to demonstrate the effect of the proposed method on convolutional networks, we considered the simple network architecture shown in Figure 1b which consists of three convolutional and two fully connected layers. In our study, we will break the network in the convolutional section representing the feature extractor and

the fully-connected section playing the role of the detector. The original network was trained on the CIFAR10 dataset first and after degrading the feature extractor by adding noise to its parameters, we just retrained the knowledge extractor layers of the original network (i.e. fully connected layers). the results of this study are presented in Section 3.

2.3. Mixed-signal in-memory computation

If we look closely to the types of computations used in most deep neural networks, we can see that the most computationally expensive layers are those which their operation can be expressed as a simple operation of vector-by-matrix multiplication (VMM). while fully-connected layers are literally performing the VMM operation, with some shuffling of inputs and weights, the convolution operation can also be expressed by the VMM operation. Despite its simplicity, hardware implementation of these VMM operations is very challenging because of two reasons: (i) they have high volume of computation (i.e. multiplication and addition) and (ii) they have huge number of parameters that need to be transferred usually form an external memory into the processor. Mixed-signal circuits based on in-memory computing architectures are considered as a promising solution to overcome both of these problems by performing the computations in analog directly within the same memory that stores synaptic weights. Figure 2 explains this concept with more details. Figure 2a shows the simple VMM computation that can be found in almost all neural network architectures. If we look into just a single neuron, its output can be written as a dot product between its inputs and their corresponding weights. To implement this dot product operation using in-memory computing system such as the one shown in Figure 2b, we first need to map the synaptic weights to the conductance of the NVM devices (through analog programming). Now by representing inputs of the network as voltages and applying them to the rows of this memory module, the current of each NVM device would be equal to the multiplication of its conductance and the input voltage applied to it. Following the kirchhoff's current law, the total current on the columns of this array would be equal to the VMM operation that we were trying to compute. The main two advantages of using in-memory computation to perform VMM operation should be clear by now. First of all, we can perform N×N multiplications and additions automatically in parallel and secondly, there is no need to transfer the network parameters around. Of course there are drawbacks associated with this way of computing with the main one being the inability of precise programming of NVM devices with synaptic weights. In the following sections, we first study the sensitivity of neural networks to imprecise synaptic weights and then propose a solution to overcome this problem.

2.4. Implementation of imprecise network parameters

To simulate the network as closely as how it will eventually be implemented in the neuromorphic hardware, we looked into the literature to see how precisely today's nonvolatile memory devices can be programmed and modified the parameters of the trained network accordingly. Due to the feedback-based or write-read-verify the nature of algorithms used to program nonvolatile memory devices with analog or multibit values [9–11], the distribution of weight programming error is expected to be a Gaussian function centered around the original weight. In other words, the amplitude of the programming error is always proportional to the amplitude of the weight as shown in experiments [15]. As a result, in all of our experiments, we express the $g(\cdot)$ which is the function modeling the effect of weight programming algorithms used in neuromorphic hardware as:

$$g(w) = (1 + \alpha \times \mathcal{N}(\mu = 0, \sigma^2 = 1)) \times w, \tag{1}$$

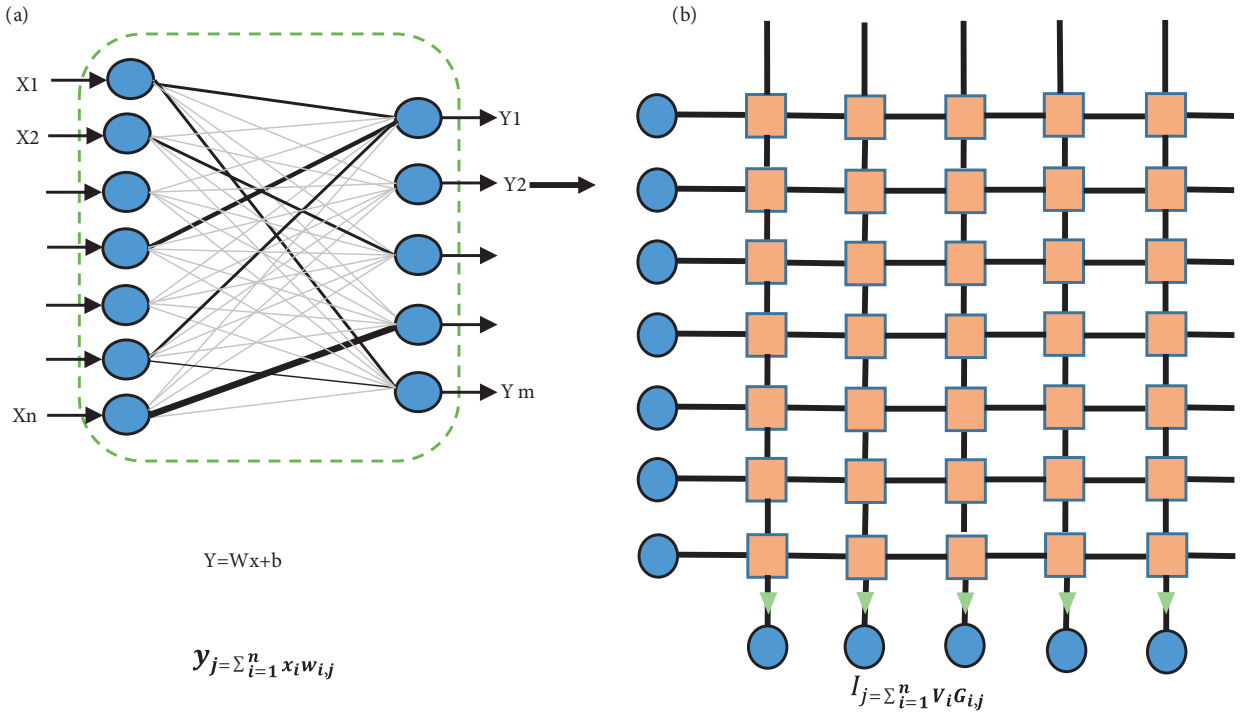


Figure 2. Implementation of VMM operation using mixed-signal in-memory computing architecture. (a) Typical VMM operation in neural networks. (b) Mapping VMM operation to NVM memory array and performing VMM operation using in-memory computation.

where $\mathcal{N}(\mu = 0, \sigma^2 = 1))$ is a normal distribution function with the mean and standard deviation of 0 and 1 respectively while α models the goodness of the analog property of the NVM devices or the programming algorithm. To keep this study as general as possible and show its potential in improving noisy networks, in our simulations the range of α is set to values much larger than those demonstrated in experiments [9–11].

3. Experimental results

3.1. Sensitivity study

Figure 3 shows the sensitivity of the considered fully-connected and convolutional neural network to the accuracy of the parameters in the feature extractor part of the networks. For this study, after training the network, the network parameters in the feature extractors were degraded according to Eq. (1) and then the classification performance of the network is remeasured. Both Figures 3a and 3b are in the fully-connected networks where in Figure 3a the network just had a single hidden layer with N_{FE} hidden neurons while Figure 3b is for the fully-connected network with two hidden layers, each with N_{NE} hidden neurons. In both networks, all the network parameters were degraded, except the weights connecting the last hidden layer to the output neurons. Surprisingly, the two-layer network with less network parameters and therefore less capacity seems to have less sensitivity to noisy weights which can be because of three reasons. Firstly, the chance of having over-fitted network is higher in the three-layer networks than two-layer ones which makes it more sensitive to noisy computation (this is also the reason of why increasing hidden neurons to more than 50 in both networks does not increase the classification rate). Secondly, the higher sensitivity of the three-layer networks can be because

of the higher depth of the network where the noise can propagate and get magnified. Finally, the activation function in these networks is a tansig function which clips the large numbers. Therefore, if the noisy input of the activation function gets into those clipping regions, the information may get lost.

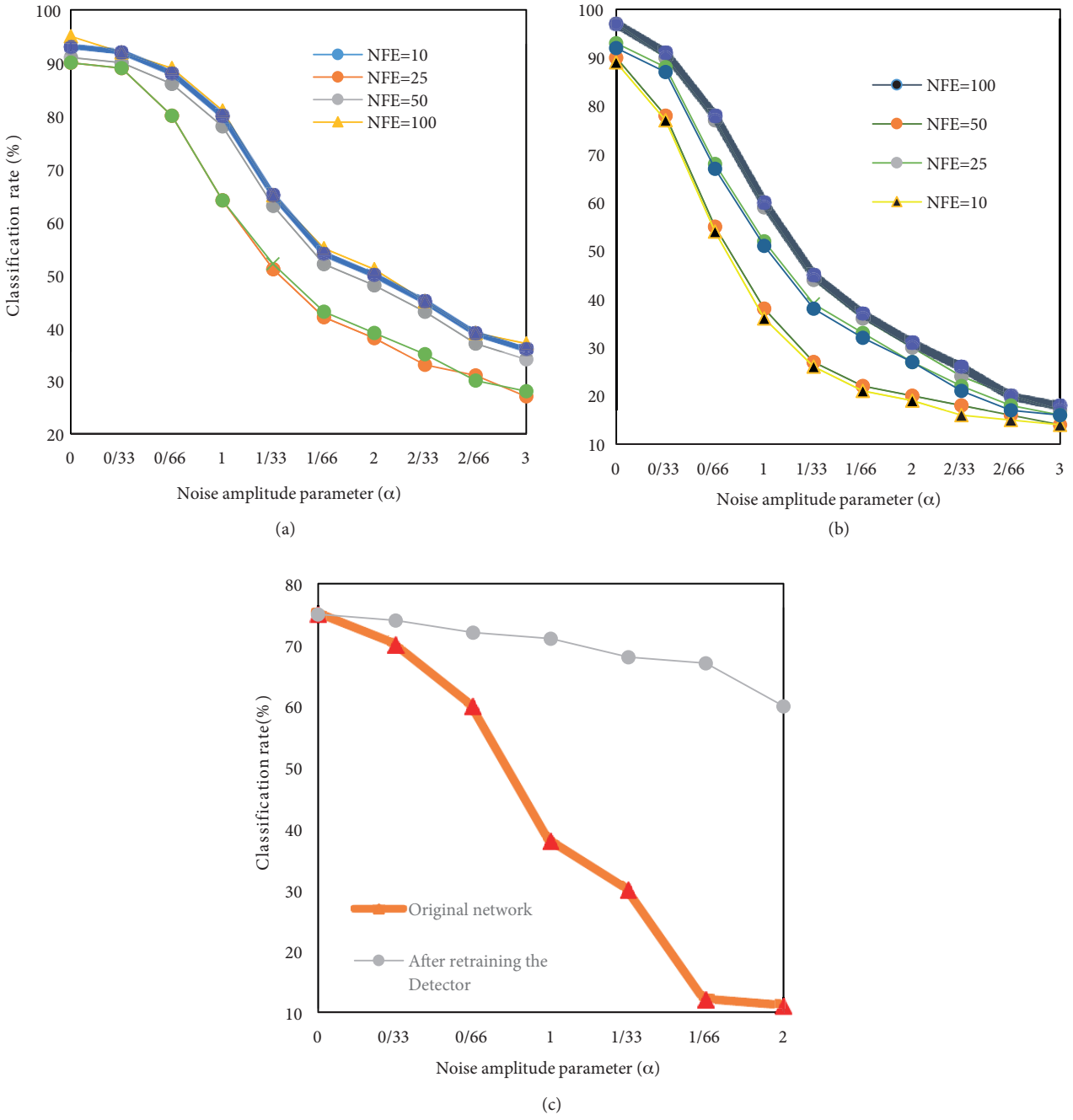


Figure 3. Results of the sensitivity study of the network shown in Figure 3. (a) Fully-connected network with a single hidden layer consisting of N_{FE} neurons. (b) Fully-connected network with a two hidden layer each with N_{FE} neurons. (c) Convolutional neural network shown in Figure 3c trained on CIFAR10 without any data augmentation.

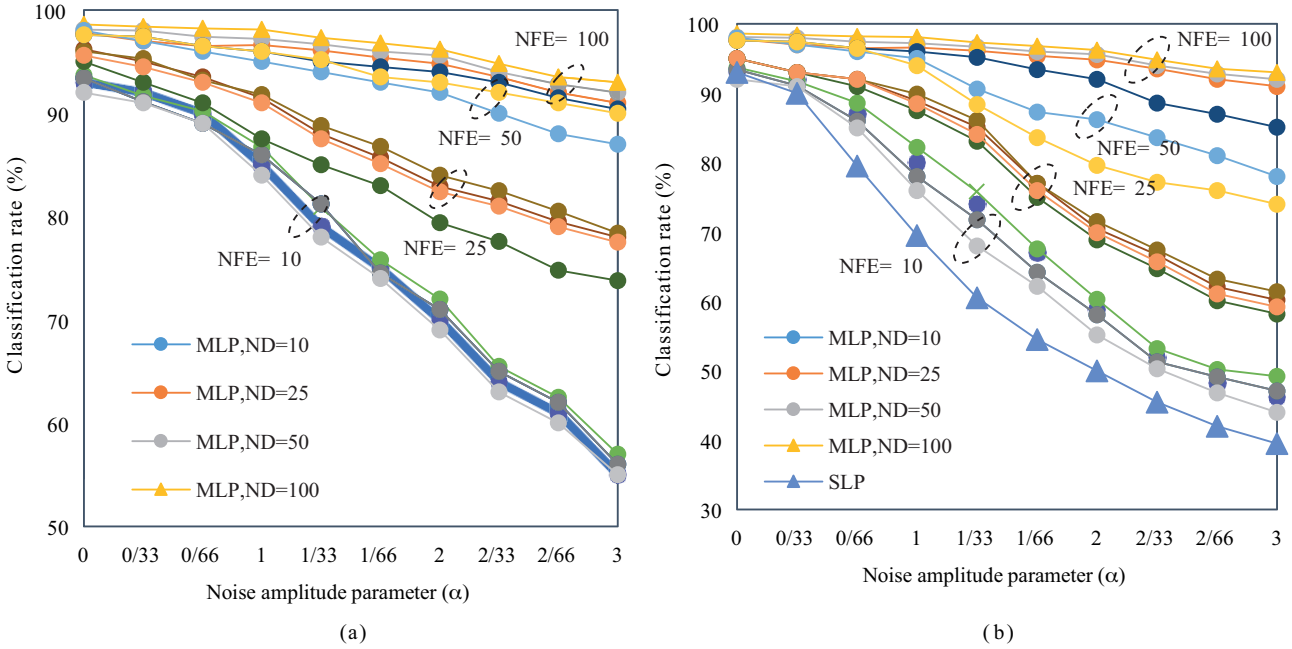
Things get more interesting as we move to the convolutional neural networks as shown in Figure 3c. Although the weights of the first three convolutional layers are degraded in this deep network, the slope of the classification rate drop is more similar to the two-layer fully connected network shown in Figure 3a meaning that the network performance does not get degraded significantly by the increase of the depth network. This is an important observation because intuitively one would expect convolutional layers to be more sensitive to noisy parameters because of the small number of parameters these layers have. As mentioned in the previous section, this robustness to imprecise computation can be due to three reasons. First, convolutional layers extract local features from the input image compared to fully-connected layers which can be classified as global feature extractors. Secondly, this network uses a ReLU activation function which does not have the limited output range like tansig. Thirdly, the max pooling layer used in this network acts as a sort of filter preventing some noisy signals from propagating within the network.

3.2. Improving the performance of degraded networks using the proposed method

Figure 4 shows the classification rate of the same degraded networks of Figure 3 when the extractor section is kept fixed, but the detector portion of the networks was retrained. As these figures show, for all the networks, we can still reach an indistinguishable classification rate even when the parameters of the feature extractor part of the network are severely degraded and are highly inaccurate. Note that $alpha = 1$ means any weight with the value of w would be mapped to a random value within the range of $[-2w,\ 4w]$ with a Gaussian distribution centered around w which is a very severe degradation. This level of degradation is much worse than what can be achieved with currently available NVM devices. Also the results of Figures 4a–4c show that while increasing the redundancy in feature extractor portion of the network can make the network more and more robust to imprecise network parameters, using a more powerful detector has a small impact on the network performance. However, this could have been expected since we have just degraded the parameters in the feature extractors. Figure 5 shows how much we were able to improve the classification rate by just retraining the last one or two layers of the already trained network and not touching the noisy feature extractor. To be more precise, Figure 5 is the result of subtracting classification performances presented in Figure 3 from the results presented in Figure 4. Unexpectedly, our proposed method has worked extremely well for the convolutional network used to classify CIFAR10 dataset which has also been our deepest network.

4. Further discussion and conclusion

Memory bottleneck is slowing down the deployment of real AI capabilities at the edge using digital hardware. To overcome this barrier, one solution is to use a so-called in-memory computation where the main computation is happening inside the array of nonvolatile memory elements that also stores the parameters of neural networks. Storing network parameters in analog inside the memory array come with the drawback of ending up having low-precision weights. In this article we studied the effect of these imprecise weights on the classification performance of deep fully-connected neural networks as well as convolutional networks and showed that it would be very hard to reach an acceptable level of performance if we just blindly import the parameters of the trained network to the nueromorphic hardware. Using more sophisticated and more complex hardware can improve the hardware quality with the cost of reducing its efficiency. On the other hand, our solution is simple: implement the feature extractor section of the network with any imprecise hardware to maximize the efficiency and just retrain the detector section of the network. In this way, we will be implementing the computationally expensive part of the network with very efficient hardware and retrain just a small part of the network which can be done pretty fast.



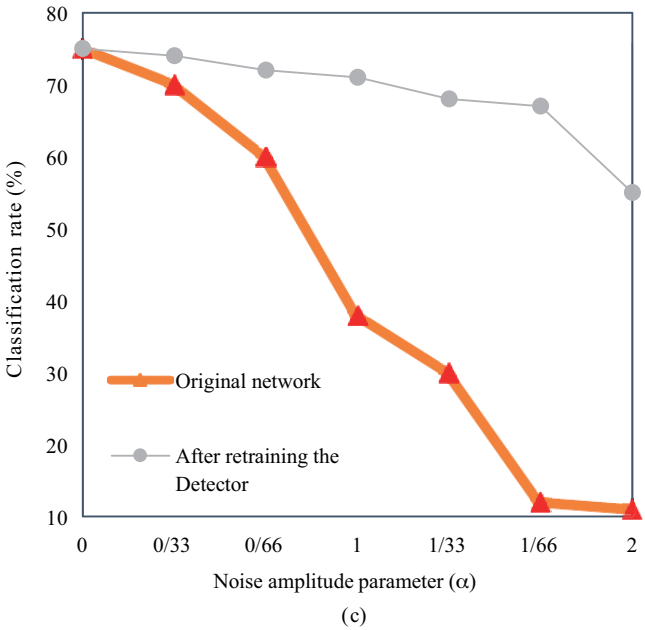


Figure 4. Improving the degraded performance of neural networks by retraining the detector part of the network. (a) Two-layer, fully-connected network. (b) Three-layer, fully-connected network. (c) Convolutional network. It is worth noting that although for most values of N_{FE} the degraded networks in Figures 4a and 4b produced meaningless outputs, here we can see that with retraining the networks with larger number of hidden neurons in their feature extractors have significantly better performances.

In this way we can reach very high hardware efficiency while keeping the retraining time at a manageable level. As the results of our study showed, using this method we could reach the indistinguishable level of performance using extremely imprecise computations.

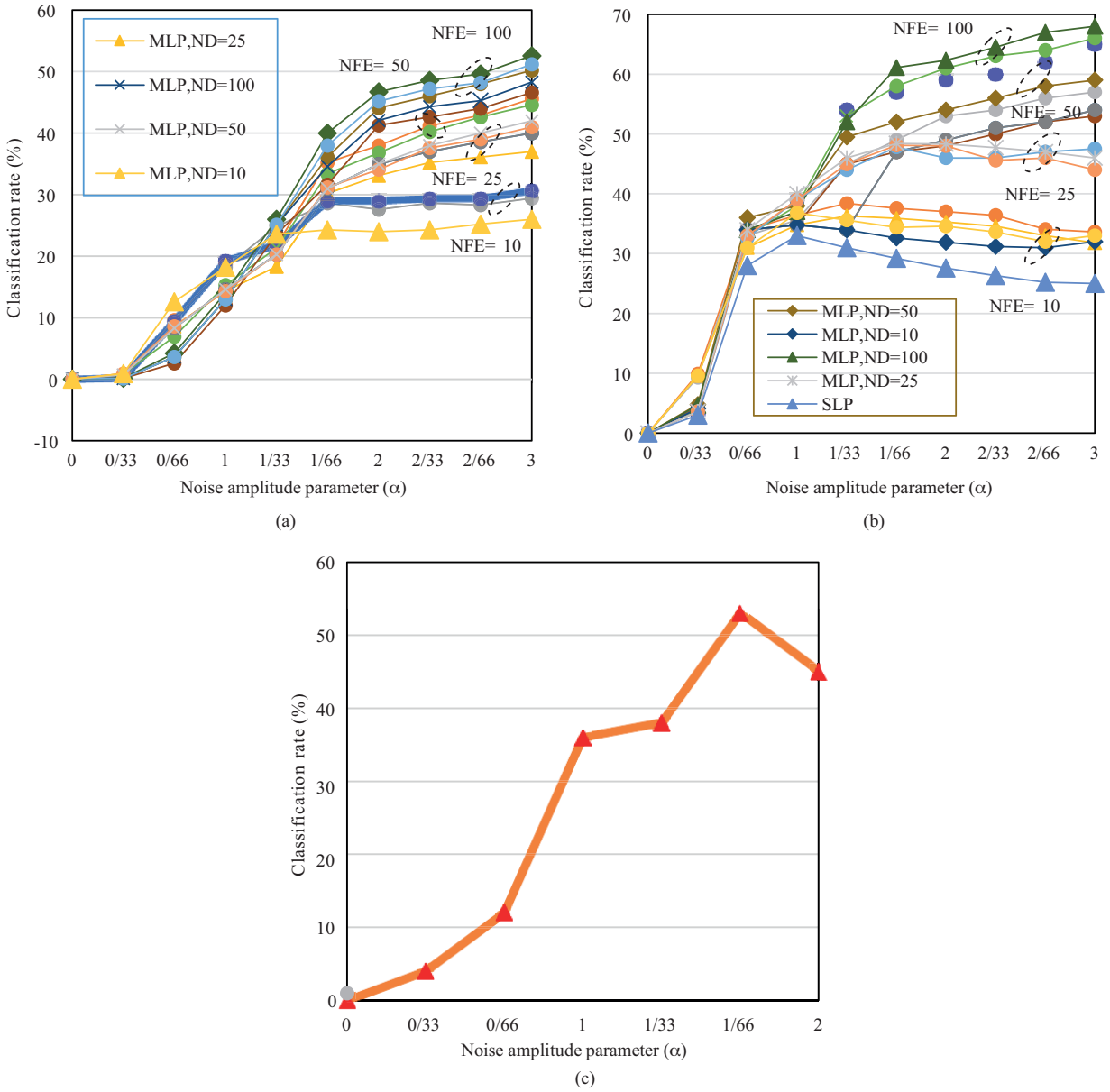


Figure 5. This figure shows how much we could increase the classification performance of the degraded networks by just retraining the detectors. (a) Two-layer fully connected network. (b) Three-layer fully-connected network. (c) Convolutional network. For the fully connected networks, the results are averaged over 100 runs but for the convolutional network, the network is simulated just once due to the lack of enough computing resources.

References

[1] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint 2017. arXiv:1704.04861.

[2] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015; 521 (2015): 1-20. doi: 10.1038/nature14539

[3] Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp et al. End to end learning for self-driving cars. arXiv preprint 2016. arXiv:1604.07316.

[4] Hannun A, Case C, Casper J, Catanzaro B, Diamos G et al. Deep speech: scaling up end-to-end speech recognition. arXiv preprint 2014. arXiv:1412.5567.

[5] Gibney E. Google AI algorithm masters ancient game of go. Nature News 2016; 529 (7587): 445. doi: 10.1038/529445a

[6] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Schölkopf B, Platt J, Hofmann T (editors). Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2012, pp.1097-1105.

[7] Hrishikesh J, Raha A, Younghyun K, Soubhagya S, Woo SL et al. Energy-efficient system design for IoT devices. In: 2016 21st Asia and South Pacific Design Automation Conference; Beijung, China; 2016. pp. 298-301.

[8] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Quantized neural networks: training neural networks with Low precision weights and activations. The Journal of Machine Learning Research 2017; 18 (1): 6869-6898.

[9] Han S, Mao H, Dally WJ. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint 2015. arXiv:1510.00149.

[10] Indiveri G, Linares-Barranco B, Hamilton TJ, Van Schaik A, Cummings RE et al. Neuromorphic silicon neuron circuits. Frontiers in Neuroscience 2011; 1: 1-20. doi: 10.3389/fnins.2011.00073

[11] Prezioso M, Merrikh-Bayat F, Hoskins BD, Adam GC, Likharev KK et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature 2015; 521 (7550): 61-64. doi: 10.1038/nature14441

[12] Merrikh-Bayat F, Prezioso M, Chakrabarti B, Nili H, Kataeva I et al. Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. Nature Communications 2018; 9 (1): 1-7. doi: 10.1038/s41467-018-04482-4

[13] Merrikh-Bayat F, Guo X, Klachko M, Prezioso M, Likharev KK et al. High-performance mixed-signal neurocomputing with nanoscale oating-gate memory cell arrays. IEEE Transactions on Neural Networks and Learning Systems 2017; 29 (10): 4782-4790. doi: 10.1109/TNNLS.2017.2778940

[14] Guo X, Merrikh-Bayat F, Bavandpour M, Klachko M, Mahmoodi MR et al. Fast, energy efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR ash memory technology. In: IEEE International Electron Devices Meeting (IEDM); New York, NY, USA; 2017. pp. 1-20. doi: 10.1109/IEDM.2017.8268341

[15] Merrikh-Bayat F, Prezioso M, Chakrabarti B, Kataeva I, Strukov D. Memristor-based perceptron classifier: Increasing complexity and coping with imperfect hardware. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD); New York, NY, USA; 2017. pp. 549-554. doi: 10.1109/ICCAD.2017.8203825

[16] Williams RS. How we found the missing memristor. In: Adamatzky A, Chan G (editors). Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM. World Scientific Nonlinear Science Series. Hackensack, NJ, USA: World Scientific Publishing Co., Inc., 2013, pp. 483-489.

[17] Radwan AG, Fouda ME. On the Mathematical Modeling of Memristor, Memcapacitor, and Meminductor. New York, NY, USA: Springer International Publishing, 2015.