Bridging the Gap: Multinary Weight Quantization and Noised Training to Combat Hardware-induced Noise for Memristor-based Neural Networks

Hongwei Tu
519030910359

Han Yan
519030910404

Jinghao Feng
519030910362

***Abstract*—Though with outstanding energy-efficiency, Memristor-based Hardware Neural Networks (MemHNNs) suffer from hardware imperfections, among which random noise accounts largely for the drop in test accuracy. To this end, this project aims at bridging the gap between the training and test phases by applying a naive simulation of hardware-induced noise, and studies how we can improve test accuracy by adopting new training strategies, including multinary weight quantization, noised training and modifications to network structures and parameters. We show that our strategies bring remarkable improvements to the baseline model provided. The source code is available at https://github.com/wolfball/MWQ-NT-MNN.**

***Index Terms*—Memristor-based Hardware Neural Network (MemHNN), Quantized Neural Network (QNN)**

Author Contribution Statement

The authors state that they all contribute equally to the project, and that the ranking is in no particular order.

Tu(33.3%): investigate recent literature, propose methods, implement the methods, analyze results, revise the manuscript.

Yan(33.3%): propose methods, implement the methods, conduct experiments, analyze results, visualize results.

Feng(33.3%): investigate recent literature, propose methods, conduct experiments, analyze results, write the manuscript.

I. Introduction

Hardware-based Neural Networks, especially Memristor-based ones, suffer from hardware non-perfections. Specifically, we focus on the case where the model is pre-trained on software, and transplanted onto memristor chips. The noise is thus introduced in the test phase. In our project, noise is naively simulated by adding random multiplicative exponential Gaussian noise to the model weights in the test phase. Under different standard deviations (std) of the noise, different performance drops are observed.

Lying at the heart of the performance issues is the discrepancy between the two phases, where the noise is present in the test phase but absent in the training phase. In order to bridge the gap between the two phases and increase the robustness against noise, we first fine-tune hyper-parameters on software, and then apply new training strategies including multinary weight quantization and noised training before weight import, mainly inspired by [1]. Weight quantization is originally required by digital hardware, but we state that it can provide some robustness against hardware-induced weight noise and propose a novel multinary quantization method. In addition, with the strategy of noised training, random multiplicative exponential Gaussian noise is added to each learnable weight before forward propagation, as a means of forcing the model to learn some robustness against noise. The idea is: (1) We maintain a full-precision network (with no quantization or noise) throughout the whole training phase; (2) in each mini-batch training round we create a Quantized Neural Network (QNN) (with noise added afterwards) counterpart of the full-precision network to perform inference and estimated gradient calculation, and update the weights of the full-precision network (instead of the QNN) via a customized update method. Ablation experiments are carried out and the results are shown in Section III. Our framework is validated on MNIST, where the source code for the baseline is directly from PyTorch.

The remainder of this report is organized as follows: Section II describes our training strategies in detail, Section III shows the results for parameter-tunings of the baseline and the performance of our framework, as well as an ablation study, and Section IV draws a conclusion.

II. Framework

Quantization is a natural requirement of digital hardware. Due to bit restrictions, engineers must make tradeoffs between accuracy and hardware cost. On the other hand, quantization may help eliminate some destructive effects of weight noise, as noise within a quantized interval is ignored. Therefore, we propose to apply multinary weight quantization in both the training and test phases. Following [1], Quantized Neural Network (QNN) architecture is adopted. A QNN layer is defined as

$$\boldsymbol{y}^l = f_s^l\left(\left[g \otimes q\left(\boldsymbol{W}^l\right)\right]\boldsymbol{x}^l + \left[g \otimes q\left(\boldsymbol{b}^l\right)\right]\right) \tag{1}$$

where $\boldsymbol{x}^l$ is the input, $\boldsymbol{b}^l$ the bias, $q(\cdot)$ denotes the weight quantization function (note that this notion applys to all learnable layers, not only FC), and g a multiplicative Gaussian noise $ln(g) \sim \mathcal{N}(0, \sigma_G^2)$, which is a naive simulation of hardware-induced noise. We assume the statistical quantities of the hardware-induced noise are known in advance or can be estimated cheaply. $\otimes$ means element-wise multiplication of a scalar to a matrix.

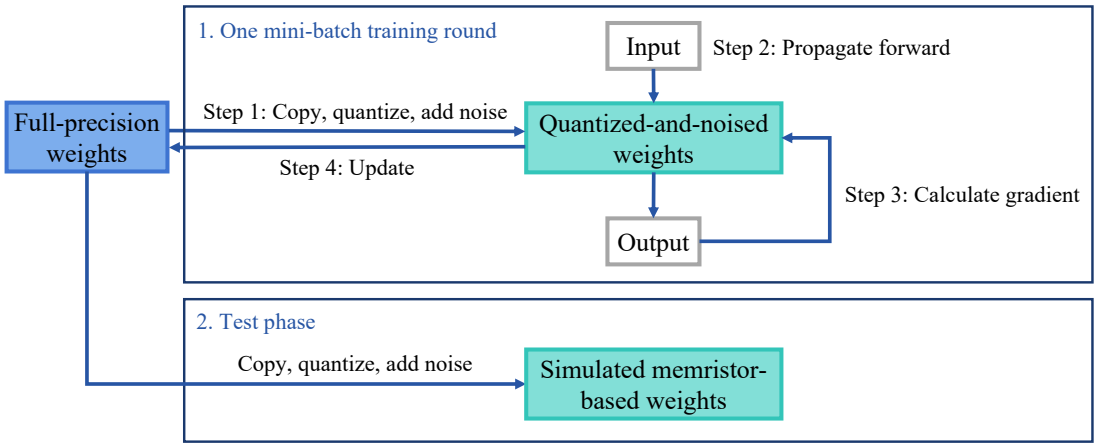


Fig. 1: Overview of our proposed framework.

For quantization, a strategy termed ternarization was introduced in [1]:

$$q\left(w^{i,l}\right)=\begin{cases}+\mu_G, & w^{i,l}>\Delta\\ 0, & \left|w^{i,l}\right|<\Delta\\ -\mu_G, & w^{i,l}<-\Delta\end{cases}, \tag{2}$$

where μ_G is the quantization level, and Δ is a positive threshold. Range tuning is performed on both μ_G and Δ.

We propose a novel multinary quantization method by extending the ternarization to multinary quantization:

$$q\left(w^{i,l}\right)=\begin{cases}0, & -\Delta<w^{i,l}<\Delta\\ +i\cdot\mu_G, & i\cdot\Delta\le w^{i,l}<(i+1)\cdot\Delta\\ -i\cdot\mu_G, & -(i+1)\cdot\Delta<w^{i,l}\le-i\cdot\Delta\\ +k\cdot\mu_G, & w^{i,l}\ge k\cdot\Delta\\ -k\cdot\mu_G, & w^{i,l}\le-k\cdot\Delta\end{cases}, \tag{3}$$

where $i\in\{1,2,\ldots,k-1\}$ is the quantization order and $n=2k+1$ in the term n-nary.

An overview of our proposed framework is shown in Fig. 1. Throughout the training phase, the software-based full-precision weights (FP-weights) are maintained, while in each mini-batch training round the FP-weights are copied, quantized and then added with noise to form the quantized-and-noised weights (QN-weights). In the forward propagation, outputs are produced using the QN-weights; in the back propagation, gradients propagate through the QN-weights, and weight updates are made to the FP-weights (instead of the QN-weights). At the end of the round, the QN-weights are discarded and will be created again in the next mini-batch training round. Furthermore, in the test phase, the same quantization and noising steps are performed on the well-trained FP-weights to simulate the real-world scenario of weight import. Since we have assumed that the statistical quantities of the noise can be estimated, the noise added in the training phase and that in the test phase conform to the same distribution.

Multinary quantization in our framework tries to compress learned knowledge into limited bits, assisting noised training in directly forcing the model to learn some robustness against noise. However, quantization asks for a new weight update strategy. While we tend to retain gradient-based back propagation and Stochastic Gradient Descent (SGD), quantization breaks the propagation chain since it is not differentiable. Still following [1] , the gradient of the quantization step is estimated with Straight-Through Estimator (STE):

$$\frac{\delta q\left(w^{i,l}\right)}{\delta w^{i,l}}=\begin{cases}1, & \left|w^{i,l}\right|\le t_c\\ 0, & \left|w^{i,l}\right|>t_c\end{cases}, \tag{4}$$

where t_c is the clipping threshold. It silences those gradients on large weights which do not affect quantized weights. Regarding the code implementation, we only need to apply a threshold mask to the gradients calculated on the QN-weights and update the FP-weights using the masked gradients multiplied by the added noise since the noise is multiplicative.

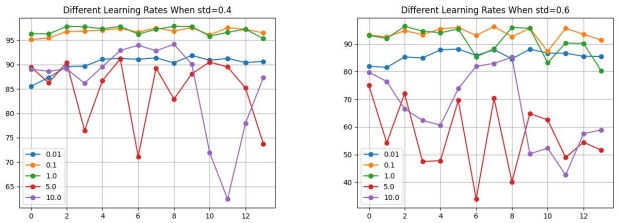
III. Experiments

In this section, we first introduce the fine-tuning results for the baseline model implemented using PyTorch on the classic dataset MNIST, with the noise adding function provided in the project slides. Then, we show results of our new training strategies, along with an ablation study. Note that in all figures the x-axis represents the number of epochs, and the y-axis represents the accuracy in percentage (%).

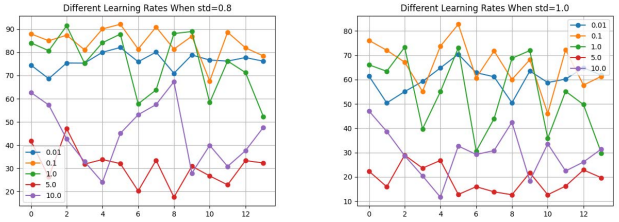
A. Baseline Tuning

For baseline, we study the effects of the learning rate, the attenuation coefficient of the learning rate, the batch size and the model architecture. The baseline architecture is based on the famous LeNet-5.

Learning rate. As shown in Fig. 2, when the standard deviation (std) of the noise is large, a large learning rate leads to apparent accuracy volatility or simply bad results, while a smaller learning rate is less affected by the noise to some extent. When the std is small, a small learning rate causes slow training where the local maxima are easily achieved, while a larger learning rate leads to volatility and instability.

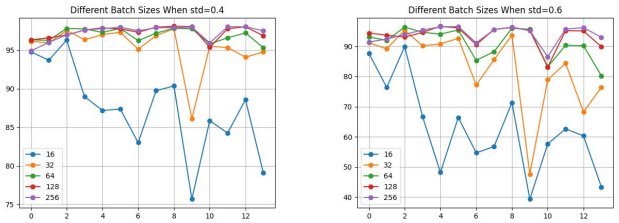


(a) std=0.4 (b) std=0.6

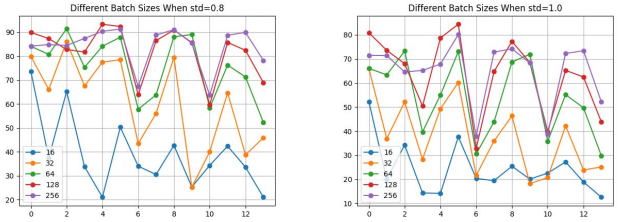


(c) std=0.8 (d) std=1.0

Fig. 2: Effects of learning rates under different noise std's.

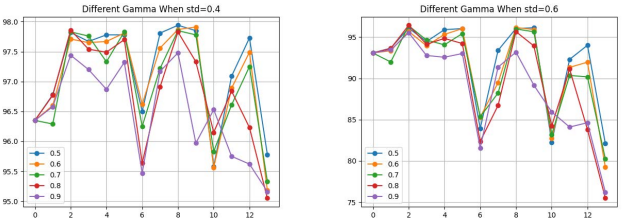


(a) std=0.4 (b) std=0.6

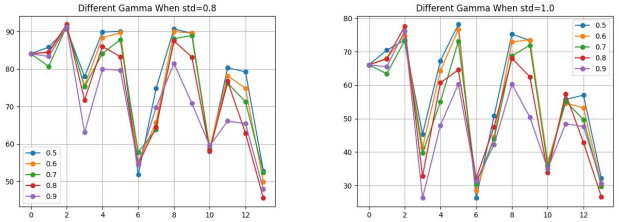


(c) std=0.8 (d) std=1.0

Fig. 4: Effects of batch sizes under different noise std's.

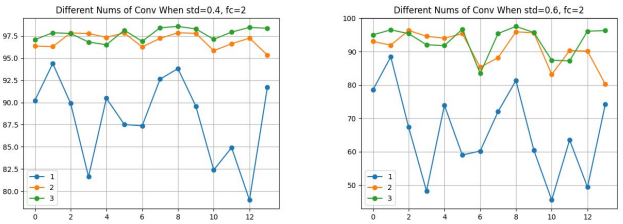


(a) std=0.4 (b) std=0.6

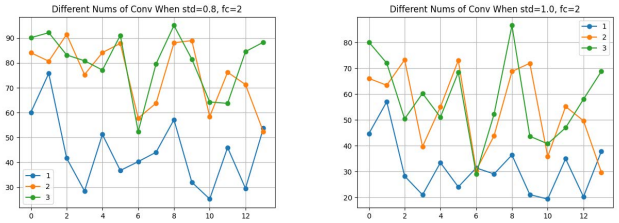


(c) std=0.8 (d) std=1.0

Fig. 3: Effects of attenuation coefficients under different noise std's.



(a) std=0.4 (b) std=0.6



(c) std=0.8 (d) std=1.0

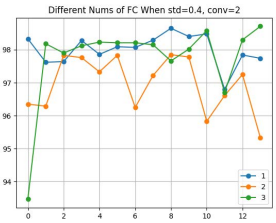
Fig. 5: Effects of numbers of conv layer under different noise std's.

Attenuation coefficient. We can see from Fig. 3 that a smaller attenuation coefficient tends to suffer less from the noise.

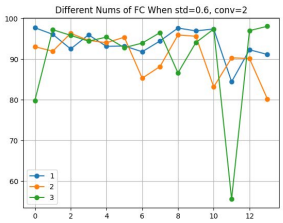
Batch size. From Fig. 4, in a certain range, a large batch size helps improve the generalization performance of the model, while a too small batch size makes training unstable and easily trapped in local maxima. Of course, the larger the std, the more instability.

Number of convolution layers. As shown in Fig. 5, more convolution layers lead to better performance. More convolution layers also bring better robustness against noise, but are more resource-intensive. We infer that a stronger ability to learn the locality of features accounts for the observed robustness.

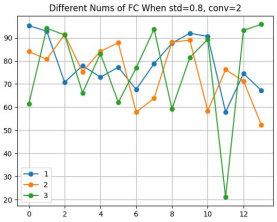
Number of FC layers. We can see from Fig. 6 that the best layer number is 3, while 1 ranks second and 2 ranks third. This is a bit counter-intuitive. Our explanation is that when the number of FC layer is 1, we remove one dropout layer and the absence of the dropout layer might have affected the stability of the network.



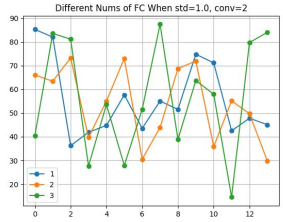
(a) std=0.4



(b) std=0.6

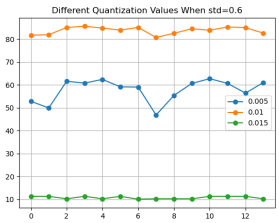


(c) std=0.8

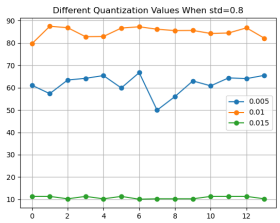


(d) std=1.0

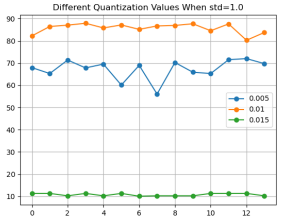
Fig. 6: Effects of numbers of FC layers under different noise std's.



(a) std=0.6



(b) std=0.8



(c) std=1.0

Fig. 7: Effects of quantization values under different noise std's.

B. Our Training Framework

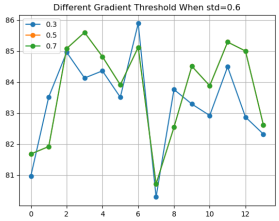
For our proposed training framework, we first study the effects of the quantization level μ_G, the gradient threshold t_c. Note that ternarization (i.e., 3-nary quantization that has an order of 1) is adopted here to reduce the number of hyper-parameters under consideration.

Quantization level. We can see from Fig. 7 that the performance of our model relies on a carefully chosen quantization level, where 0.01 is the best choice. Any value higher or lower than 0.01 would make the accuracy deteriorate. Note that we set the quantization level equal to the quantization threshold for convenience. Besides, an interesting observation is that with the noise std increasing, the test accuracy of our model tends to increase too, while the accuracy of the baseline witnesses an apparent drop when the std reaches 1.0. This suggests that our model adapts to the noise very well, especially to those with a relatively high std. Compared to the baseline, our model also shows better stability of the performance, where only slight fluctuations in the test accuracy over epochs are observed.

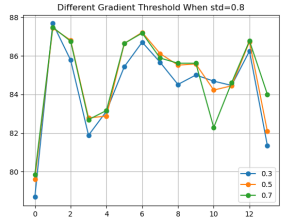
Gradient threshold. As seen in Fig. 8, the gradient threshold does not significantly affect the performance. A higher threshold only leads to a slightly larger fluctuation in the accuracy. The model is less sensitive to the gradient threshold than to the quantization level.

C. Multinary Quantization

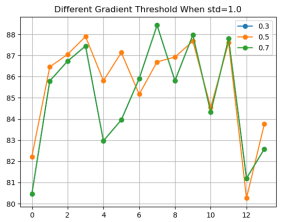
Finally, we compare the performance of our framework under different maximum quantization orders k in n-nary quantization (note that $n = 2k + 1$). As shown in Fig. 9, when the std is low, increasing the order from 1 (3-nary in the figure) to 5 (11-nary in the figure) brings accuracy rises, and



(a) std=0.6



(b) std=0.8



(c) std=1.0

Fig. 8: Effects of gradient threshold under different noise std's.

11-nary quantization achieves even better results. When the std is relatively high (e.g., 1.2) different orders show similar performance. Still, the accuracy drops with the std increasing are much smoother than the baseline, which suggests that our model does have good robustness against noise.

D. Ablation Study

In real engineering, increasing the maximum quantization orders requires chips with more bits, which leads to a potential trade-off between performance and cost. Also, noised training will be infeasible if the statistical quantities of the hardware-induced noise can not be estimated. Therefore, we carry out

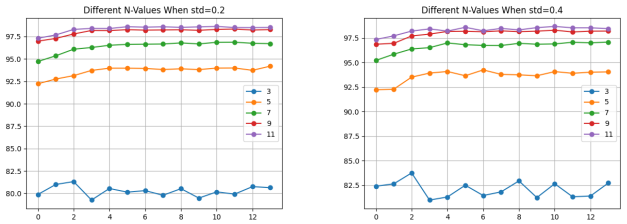
TABLE I: Test accuracy at convergence for each model type under different noise std's.

Test accuracy / Noise std / Model type	0.2	0.4	0.6	0.8	1.0
baseline	98.20	95.33	80.24	52.31	29.78
baseline + NoisedTraining	**99.07**	**98.91**	**98.48**	97.40	94.74
baseline + 9-nary	98.30	98.18	97.89	97.25	94.22
baseline + 3-nary + NoisedTraining	81.31	83.75	85.50	86.74	87.88
baseline + 5-nary + NoisedTraining	94.20	94.24	94.14	94.53	95.03
baseline + 7-nary + NoisedTraining	96.87	97.08	97.54	97.18	**95.64**
baseline + 9-nary + NoisedTraining	98.34	98.29	98.18	**97.55**	95.10
baseline + 11-nary + NoisedTraining	98.63	98.67	98.47	97.43	95.47

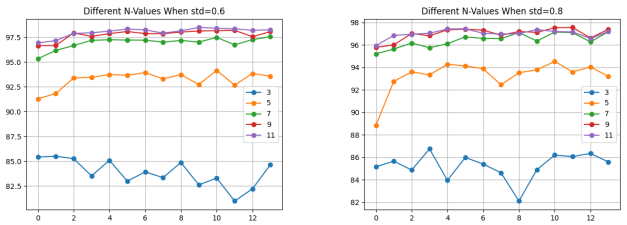
an ablation study and investigate the effect of each component in our framework. The model types under consideration in the ablation study are a combination of any one or more of the baseline, noised training and multinary quantization. The test accuracy at convergence corresponding to each model type under different noise std's is shown in TABLE I. Note that for the baseline, there is no noise added in the training phase, so we can not decide based on the training loss at which epoch to import the weights and start the test phase; a low training loss or a high training accuracy does not necessarily lead to good test results. Therefore, we take the test accuracy when the training loss converges. We can not take the best test accuracy among the epochs for comparison since in real-world scenarios it would be impossible to estimate the true test accuracy **before weight import** without approaches such as noised training, despite the fact that we can easily have access to estimated true test loss in a simulation.

One can see from TABLE I that model training is stable and robust over epochs with our proposed methods, where multinary quantization outperforms ternarization. Baseline with 9-nary quantization and no noised training achieves comparable results with some of the best model types. In stark contrast, the training process of the baseline suffers from great instability when the noise std is relatively high (0.6, 0.8, 1.0). Interestingly, when the noise std is relatively low (0.2, 0.4, 0.6), the best test accuracy is achieved by baseline + NoisedTraining. In this case, the noise std is not high enough to be the bottleneck in the model performance, and the test accuracy with multinary quantization is slightly lower than that of baseline + NoisedTraining since quantization results in the loss of information. However, as the noise std increases, it affects the performance to a greater extent, and this is when multinary quantization comes into play due to the denoising and stabilizing effect of quantization.

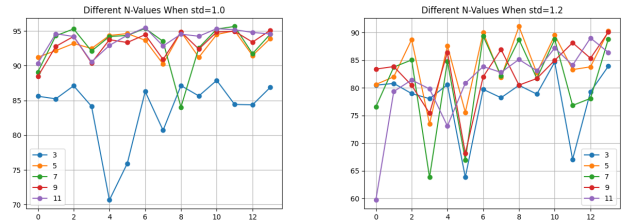
Our proposed methods, noised training and multinary quantization, generally stabilize model training and provide more robustness. Depending on the use case, these methods can be adopted individually or jointly. Especially when the statistical quantities of the hardware-induced noise are inaccessible



(a) std=0.2 (b) std=0.4



(c) std=0.6 (d) std=0.8



(e) std=1.0 (f) std=1.2

Fig. 9: Effects of n-nary quantization under different noise std's.

IV. Conclusion

In this project, to tackle the the drop in test accuracy due to hardware-induced noise, we bridge the gap between the training and test phases by applying a novel multinary weight quantization method and adopting noised training to significantly improve the test accuracy of a memristor-based neural network on the MNIST dataset. We have carried out extensive experiments, including fine-tuning the baseline, validation of

our training framework, and an ablation study. Our proposed framework produces superior results even when the noise std is relatively high (98.48% test acc when std=0.6, 97.55% test acc when std=0.8, 95.64% test acc when std=1.0).

ACKNOWLEDGMENT

We would like to thank Professor Ye and TAs for their wise guidance and patient help.

REFERENCES

[1] G. Boquet, E. Macias, A. Morell, J. Serrano, E. Miranda, and J. L. Vicario, "Offline training for memristor-based neural networks," in *2020 28th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1547–1551.