ECSE 541 Final Project: Applying the Analog Hardware Acceleration Kit for In-memory Computing Design

Hung-Yang Chang[1], Jack Hu[1], Snehal Chavan[1], Ilshat Sagitov[1]
[1]Department of Electrical and Computer Engineering, McGill University

Deep Neural Networks (DNNs) have achieved promising progress in many complex tasks, including image or speech recognition, gaming, and real-time language responses. Training of large DNNs, however, is still considered a time-consuming and computationally intensive task that demands extra high computational resources for many days. Analog based devices have shown great potential to break von Neumann's bottleneck and potentially accelerate DNN training by orders of magnitude while using much less power. Nevertheless, the non-ideality problem still remain a daunting problem when implementing analog devices in in-memory computing design [1]. Recently proposed IBM's analog hardware acceleration kit [2] provides us a window for the simulation of non-idealities of analog devices. In this paper, the IBM tool is utilized to simulate Resistive Processing Units (RPUs) to represent weights and biases value in Fully-Connected Neural Network (FCNN). MNIST dataset [3] is applied to the Analog FCNN. By performing optimization on hyperparameters such as activation functions, batch size, and learning rate, 100% testing accuracy could be achieved even with considering noise with SNR = 30 db. Moreover, computation cost and the performance is also examined. The result shows that even by applying only 3 layered neural networks (784-128-10), 95% accuracy can be achieved with only 3% decrements compared to the deeper ones.

I. Introduction

Deep Neural Networks (DNNs), as one of the AI machine learning algorithms, have achieved a promising progress in many complex tasks, including image or speech recognition, gaming, and real-time language responses. For computing phase of DNNs, multitude of matrix-to-matrix additions and multiplications are required, which can be processed using large-scale parallelization available from Graphics processing units (GPUs).

Traditionally, in computer architectures, the computation and memory units are separated. For DNN applications, data is moved back and forth between storage and computation unit too many times, creating the bottleneck for time- and energy-efficient implementations. This bottleneck is known as *von Neumann bottleneck* or *Memory Wall* [4]. Alternatively, in-memory computing architectures, well-described in [5], are designed to eliminate this bottleneck, combining computation and storage components within a single device. This architecture can be implemented utilizing RPU to represent weights and biases value because analog devices can perform multiplication and addition by the nature of ohm's law and Kirchhoff's law, respectively. This could perform the local and parallel weight storage and processing for DNN applications.

The remainder of this paper is organized as follows. In section II, we provide an overview of related works using in-memory computing principles to break von Neumann's bottleneck and what current challenges associated with such design. In section III, we explain how neural networks are constructed with Resistive Processing Unit (RPU) with IBM tool, which RPU configurations we use throughout our work and how we perform Design Space Exploration (DSE) of the networks. In section IV, we describe our simulation setup including tools and libraries, and then define the baseline, which is used as the reference for various simulations. In section V, simulation results are shown, comparing the classification capability of the baseline model with models, obtained through DSE analysis. Finally, this report is concluded in Section VI, where the results are discussed along with possible avenues for future work.

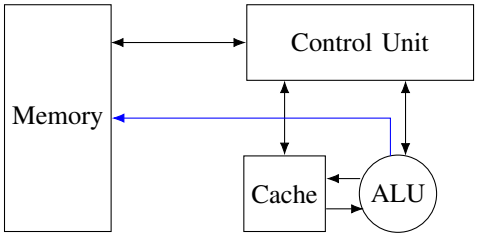

Fig. 1: Concept of in-memory computing architecture [2].

II. Related work

To break the memory wall, several research teams [1] [4] [6] [7] have demonstrated the efficient solutions using analog devices to represent weights and biases in in-memory computing. As shown in Fig. 1, computations can be done inside a memory block in single access of ALU. This technique is not realizable in conventional architectures, which experience von Neumann bottleneck. However, all aforementioned researches involved designing chips from scratch and testing the behavior of analog devices from the tape-out chips. This lack of open-source pure simulation tools impedes new designers to conduct further in-depth research. Thankfully, IBM had been constructing a brand-new Analog Hardware Acceleration Kit for exploring the capabilities of in-memory computing devices in the context of artificial intelligence since Oct. 2020 [2].

This IBM analog hardware simulation tool enables users to both simulate the non-ideality behavior of analog circuits and also to utilize PyTorch-like library to evaluate their implementations. For example, users can import libraries such

as analog.nn and analog.optim analogous to Pytorch's methodology to create neural networks and optimizers [8]. However, this simulation approach is fairly new and have not yet gained enough traction in the research community. Thus, in order to fill that gap, we shall test this toolkit by performing DSE techniques to estimate the classification accuracy of various proposed models.

The hyperparameter optimization or DSE of Neural Network (NN) is a multiobjective optimization problem with multiple constraints. It is becoming a progressively challenging and costly process which is being considered more of an art than science. There are various methods proposed for hyperparameter optimization such as random or grid search [9], random forecast [10], Gaussian processes [11]. Considering the execution time and computational complexity of the training phase, grid search is used for DSE in this work.

III. Approach

A. Neural Network and RPU

The IBM's Analog Hardware Acceleration Kit emulates in-memory analog NN of fully-connected layers. The model implements RPUs, which represent analog weights that can store value and be updated if needed locally. Here, RPU is a concept, reflecting behavior of an actual device, which could be resistive switching device [4], phase-change memory (PCM) [5], or another analog device. For detailed documentation about RPU unit, we refer readers to [1]. To implement the weight update, multiplication operations used during the update are simplified by stochastic computations. The update rule for any weight ω_{ij} is shown in Eq. (1).

$$\omega_{ij} \leftarrow \omega_{ij} + \Delta\omega \sum_{n=1}^{BL} A_i^n \wedge B_j^n \,, \tag{1}$$

where BL is the length of the stochastic bit stream that is used during the update cycle, $\Delta\omega_{min}$ is the update step size, A_i^n and B_j^n are random variables that are characterized by Bernoulli process with n representing the position in trial sequence.

During the weight update, RPU experiences Additive White Gaussian Noise (AWGN). We characterize the relation of weight signals to noise by Signal-to-noise ratio (SNR) and denote it with γ. The corresponding equation is shown in Eq. (2).

$$\gamma = \frac{1}{\sigma^2} \tag{2}$$

Since absolute values of weights are $|w| \leq 1$, the signal power is also 1. The noise power is σ^2, where σ is the standard deviation. The simulation toolkit allows to modify the value of σ. During the weight update, its step size $\Delta\omega$ is proportional to conductance of the RPU. The IBM libraries provide several update models. Among them, we apply the following ones in our experiments.

- *Constant update step model*. In this model, update step is constant throughout the resistive range of the RPU.
- *Linear update step model*. In this model, update step linearly dependent to the current value of the weight itself.
- *Exponential update step model*. In this model, update step varies in exponential manner, which reflects the real CMOS-like behaviour.

The input layer of the neural network contains 784 nodes, corresponding to the total amount of pixel resolution used in the MNIST data set. The images used in MNIST are in grey-scale meaning they only have 1 channel width. This eliminates the need for the extra nodes at the input layer or at any of the hidden layers because the network does not take the usual RGB channels into account. The output layer contains 10 nodes that are associated with 10 different classes of MNIST. These numbers correspond to $\{0,\, 1,\, 2 \ldots 9\}$. The hidden layers are customizable, so examination layer of numbers and accuracy is also conducted in section V.

B. Design Space Exploration

In a Fully-Connected Neural Network (FCNN), the design parameters represent the various hyperparameters, which define a constrained design space where the designated learning algorithm may be refined in an iterative manner by feeding in a different set of hyper-parameters on each iteration of the search space. Therefore, the best FCNN configuration can be obtained by greedily trying all possible hyper-parameter sets that exist in the design space. To reduce the dimensionality of the design space, the modified in-memory FCNN algorithm shall focus its efforts on tuning the learning rate α, activation functions, number of training epochs, number of hidden layers, batch sizes, and loss functions. These are the major hyperparameters that significantly affect the accuracy of an image data set.

In order to consider hardware-aware training, the exponential update step model is used. The noise characteristics are also fixed, providing $\gamma = 34\,\text{dB}$ for both forward and backward propagation. In section V-D, we further provide comparison of the testing accuracy at various SNR points and for different update models when considering different environments.

Since in this case, as the design space is dimensionality is small and the metaheuristics took days to produce any result whatsoever, grid search was well fitted. For grid search, we varied one parameter while keeping the other parameters fixed and once the best value of the focused parameter is obtained (keeping that value as default to the corresponding parameter) we moved on to the next parameter tuning.

IV. Experimental setup

The simulation of the network is implemented in Python, including library packages provided by IBM and PyTorch. Since both IBM toolkit and PyTorch packages are able to run in GPU, Google Colab with GPU available is chosen for the simulation environment, to speed up the simulation. [12].

To offer as fair a comparison as possible, preliminary testing was performed to settle on a baseline model. This setting of baseline model lists as follows: 4 layers neural network (784-256-128-10), optimizer $= AnalogSGD$ (Analog-aware stochastic gradient descent) from IBM library, loss function $= NLL$ (negative likelihood) from PyTorch library, activation function $= ReLU$, batch size = 32, initial learning rate

$\alpha = 0.05$, and training epoch $= 30$. Moreover, to maximize performance, the learning rate should be varied during training to escape the local minimum. Therefore, step learning rates scheduler is used multiplies learning rate by 0.5 every ten epochs here. To perform the real-world environment, AWGN is added in both the training and update phases as it is explained in section III-B. This baseline set of parameters was found to yield satisfactory results, but results could be improved with additional tuning. The accuracy over the testing set is used as a performance metric.

Ideally, all these hyperparameters should be tuned in a grid search fashion at the same time, but this would be computationally prohibitive for this project. Therefore, all subsequent experiments were run with only tuning one hyperparameter at one time while fixing the value others to those shown in the baseline model. After finding all hyperparameters, all of the hyperparameters yield the better performance are combined together to provide the final performance of the result. This method is not as robust as grid search with K-fold cross-validation but is reasonable considering the computation times involved in training those models.

For DSE, we choose the following hyperparameters to implement exploring: batch size, learning rate α, activation function, number of hidden layers, and number of nodes in each layer. We aim to achieve the minimum time and number of computations for training, while maintaining strong classification capability. In section V-D we provide comparison of classification capability at various SNR.

V. Simulation Results

A. Activation function

Non-linearity is needed in activation functions so that neural network is able to produce a nonlinear decision boundary via non-linear combinations of the weight and inputs. Therefore, four well-known activation functions are compared here, including Sigmoid, ReLU, tahh, and randomized leaky ReLU (RReLU) [13]. The results are presented in Fig. 2. It is clearly seen that the computation time are similar for all, however, the lowest accuracy was observed for sigmoid and the best result is seen when using tanh function. Besides, several values of lower and upper bounds in RReLUs were tested, but the default one from Pytorch yielding best result, which is include in Fig. 2.

B. Batch size and learning rate

Batch size and learning rate mutually affect the performance of neural networks [14]. To determine which combination of batch size and learning rate leads to the best results, a grid search was performed. The results are presented in Fig. 3. For the high learning rate, the higher batch size leads to the better results, while for the lower learning rate, the lower batch size leads to the better results. It is noteworthy that 100% could be achieved when using batch size of 128 and learning rate of 0.1.

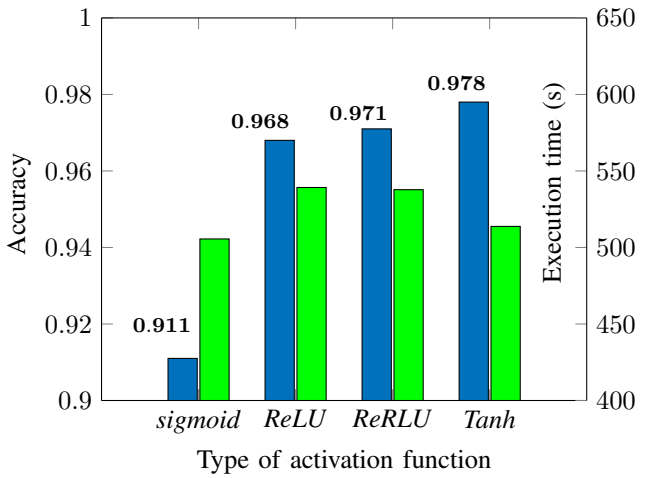

Fig. 2: Effect of activation functions on validation accuracy.

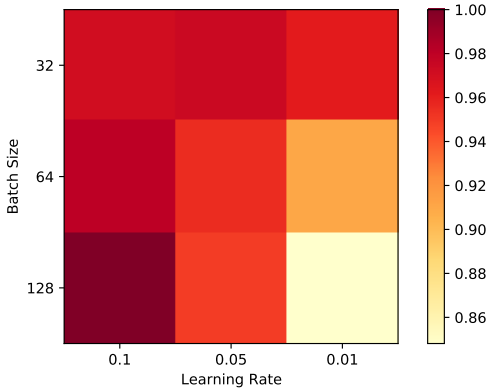

Fig. 3: Effect of batch size and learning rate on validation accuracy.

C. Number of layers of Neural Networks

The focus of this section shall be to examine how computation cost (in terms of the number of layers) affect the performance (in terms of accuracy) of an Analog FCNN.

Three experiments were conducted to shed light upon how the test set accuracy changed as the number of layers of the FCNN deepened. In each experiment, a different layer configuration for the Analog FCNN model was applied. The chosen number of layers ranged from 2 to 4 fully connected layers. The classification task at hand involved processing small 1 channel 28x28 pixels gray-scale images which do not require an abundant amount of neurons/weights to model/generalize. The chosen hidden unit configurations consisted of 512-256-128-10 for a 4 layer FCNN, 256-128-10 for a 3 layer FCNN, and 128-10 for a 2 layer FCNN. The numbers indicated between the ”-” represents the desired output size at each layer. Thus, to consider any deeper FC layer configurations or more shallow layer configurations other than the ones proposed may result in either model over-fitting or under-fitting scenarios. It is important to note that as the number of layers are increased for a FCNN, the number of weights learned within the network increases as well. Thus, the model may succumb to the danger of over-fitting as the weights can simply ”remember” the input

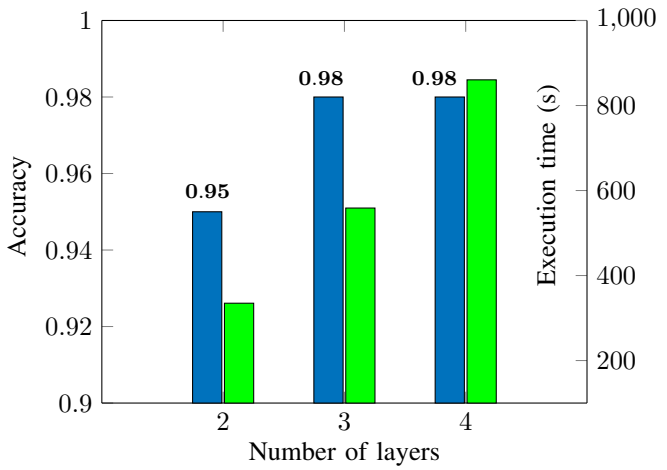

Fig. 4: Test set accuracy vs number of layers within the Analog Fully-Connected Network.

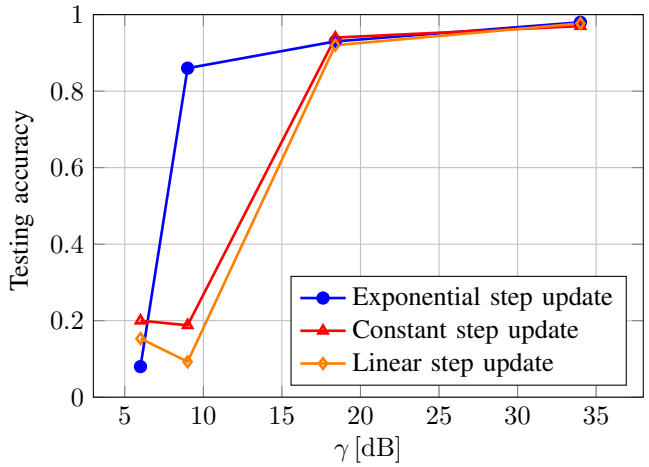

Fig. 5: Testing accuracy at various SNR for different RPU models provided by IBM libraries.

feature pixels of training images. In general, shallow layer configurations are preferred as it generalizes well to unseen data and boasts a faster processing time. According to Fig. 4 below, the results suggest that a shallow number of layers used in a FCNN can almost perform as well as it's deeper layer counter parts. It trades off 3% in accuracy for a substantial 62% decrease in run time from a 4 layer to a 2 layer FCNN.

D. Impact of the noise to the different update models

We compared each update model in various noise environment simulated with its standard parameters provided by IBM libraries. The network settings of the baseline model are used. The resulting testing accuracy at various SNR for each RPU model is shown in Fig. 5.

From the resulting figure we can see that for all models the accuracy is unacceptably low in lower SNR region, at 6 dB. As SNR increases, the accuracy is improving. The curve representing the exponential step model achieves the highest accuracy of 86% among the others at $\gamma = 9$ dB. For two other models, testing accuracy is still low at 9 dB. At higher SNR the accuracy of all models improves and eventually achieves 98%. We see that for all models the testing accuracy is high for realistic SNR region of hardware environment, 18 dB and higher.

VI. Conclusion and Future Work

In this paper, We utilized IBM's analog hardware acceleration kit to simulate RPUs to represent weight and bias values in FCNN. The MNIST dataset is applied to the Analog FCNN. By performing optimization on few selected hyperparameters, 100% testing accuracy could be achieved even with considering noise with SNR of 34 dB. Moreover, computation cost and performance are also examined. The result show that even if applying only 3 layered neural network (784-128-10), 95% accuracy can be achieved with only 3% decrements compared to the deeper ones. Also, we observe that regardless of the type of update model, the system is fault-tolerant to noise in the region of SNR, typical for hardware environment. Especially in lower SNR, the exponential update model achieves better accuracy, compared to other models, considered in the scope of the work.

Despite the decidedly strong performance of the FCNN on MNIST dataset, several other possible avenues exist for future work. Investigating other widely used CNN architectures, such as VGG-16, ResNet, and GoogLeNet, could also be tested. Incorporating other error metrics into the analysis, such as loss, precision, recall, and F-score should also be considered, as it might yield insights into weaknesses of the current model.

References

[1] T. Gokmen, Y. Vlasov, "Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations", in *Frontiers in Neuroscience*, July, 2016.
[2] IBM, "Analog Hardware Acceleration Kit" [Online]. Available: https://github.com/IBM/aihwkit. [Accessed: 09-Nov-2020].
[3] LeCun, Yann, Corinna Cortes, and C. J. Burges. "MNIST handwritten digit database", (2010): 18.
[4] D. Ielmini and H. S. Philip-Wong, "In-memory computing with resistive switching devices." *Nature Electronics 1*, no. 6 (2018): 333-343.
[5] S. Ambrogio, et al., "Equivalent-accuracy accelerated neural network training using analogue memory", in *Nature*, June, 2018.
[6] P. A. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface", in *Science 345*, p.p. 668–673 (2014), July, 2014.
[7] S. K. Esser et al., "Convolutional networks for fast, energy-efficient neuromorphic computing", in *PNAS USA 113*, no. 41, p.p. 11441–11446, Oct. 2016.
[8] D. Malowany. "Accelerate your Hyperparameters Optimization with PyTorch's Ecosystem Tools", [Online] Available: https://medium.com/pytorch/accelerate-your-hyperparameter-optimization-with-pytorchs-ecosystem-tools-bc17001b9a49.
[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", Journal of Machine Learning Research, 13:281– 305, 2012.
[10] Hutter, F., Hoos, H. H., and Leyton-Brown, K., "Sequential model-based optimization for general algorithm configuration", In Learning and Intelligent Optimization 5, 2011.
[11] J. Snoek, "Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology", PhD thesis, University of Toronto, Toronto, Canada, 2013.
[12] "Google Colaboratory", [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb.
[13] B. Xu, N. Wang, T. Chen, and M. Li: "Empirical evaluation of rectified activations in convolutional network" arXiv preprint arXiv:1505.00853, 2015.
[14] I. Kandel & M. Castelli "The Effect of Batch Size on the Generalizability of the Convolutional Neural Networks on a Histopathology Dataset", *ICT Express*, vol. 6, no. 4, pp. 312-315, 2020.