Neural Network with Binary Activations for Efficient Neuromorphic Computing

Weier Wan and Ling Li

Abstract

In this paper, we proposed techniques to train and deploy a multi-layer neural network using softmax loss for binary activations to best capitalize on its energy efficiency. These techniques include using the gradient of tanh function to approximate gradient of discrete binary threshold function during the backpropagation of training, and using a stochastic multi-sampling approach to convert high-precision input into a set of binary inputs. We used a three-layer fully connected neural network and experiment the techniques on CIFAR-10 and MNIST datasets. With the proposed strategy, we are able to train and deploy a model with binary activations with no or little loss of performance compared to a model with high-precision neurons.

I. Introduction

Artificial neural network has been adopted to achieve state-of-the-art performance across many tasks. However, the deployment of ANN on conventional hardware such as CPU and GPU remains highly inefficient in terms of both power and speed. Even the best deep network model can take days of training and consume kWs of power when deployed. The challenge becomes more severe at mobile platforms with tight hardware constraints. Moreover, with the ever increasing size of sensory networks, it is desirable to have certain data-processing capability at front-end IoT devices with limited power-budget and real-time processing requirement. Therefore, it is essential to design a novel hardware architecture to enable more efficient deployment of neural network models.

Separated memory and logic limits the efficiency for the conventional hardware architecture. It is also the main bottleneck for most machine learning systems with a huge amount of parameters. The same parameters are repeatedly retrieved from memory many times for different pieces of data. However, memory access is the most expensive operation in terms of both power and speed due to large interconnects aspect ratio in modern semiconductor technology [1]. Thus we want to reduce the number of memory access and data movement to enable faster and more energy-efficient computation.

One novel hardware architecture that can significantly reduce data movement is by performing the most intensive part of computation right at where the parameters are stored. The essential component of this hardware architecture is a non-volatile memory (NVM) crossbar array. The array can be used to compute multiply-and-accumulate (MAC) with superior efficiency in an analog fashion in contrast to the digital computation in traditional hardware [2][3][4][5]. For the NVM crossbar array, we proposed a binary representation at both input and output of the array to further improve energy-efficiency and design simplicity.

In this paper, we focus on adapting the conventional multi-layer neural network to be compatible with the proposed hardware architecture without sacrificing performance. Intuitively, replacing a real-value (e.g. 64-bit) representation with binary representation will cause loss in information and affect model performance. Moreover, there are some intrinsic problems with using binary representations. The foremost one is that a binary activation function is not continuous and therefore its gradient vanishes everywhere except at the threshold. Without adaptation, binary neurons are not compatible with gradient-descent training. Besides, in many cases, the input to the network is encoded in high-precision numbers (e.g. 64-bit). Directly trimming the precision down to 1-bit will inevitably loses lots of information. Therefore how to convert input into binary numbers while preserving most information is another aspect that we will study in this paper.

To address these problems, we proposed three techniques to train and deploy a multi-layer neural network using softmax loss with only binary activations. Through experiments on CIFAR-10 and MNIST dataset, we showed that these techniques will result in little or no loss of performance compared to using a high-precision neuron. In addition, we incorporated the real characteristics of NVM devices such as the bounded value of NVM devices.

The paper is structured as follows. Section II briefly discusses the proposed hardware architecture focusing on the basic operating principle of NVM crossbar array. Section III outlines the three techniques that we proposed to train and deploy a fully connected neural network model using softmax loss with binary activations. Section IV gives experiment results of the proposed techniques on CIFAR-10 and MNIST dataset and some explanations on the effectiveness of the techniques. Section V concludes the paper and suggests some future directions of this work.

II. NVM Crossbar Array Based Hardware Architecture

The NVM device in this paper refers particularly to the emerging non-transistor based memory such as resistive memory (RRAM) and phase change memory (PCM). Depending on the material composition, these memories usually have the properties of small-unit size, low programming energy and 3D integratability on top of CMOS circuit. However, the exact characteristics of NVM elements are beyond the scope of this paper. In this context, we simplified the NVM device as linear resistors with programmable resistance.

Figure1(a) shows the structure of a NVM crossbar. The horizontal and vertical lines represent metal wires in two layers. The red dots at intersections represent two-terminal NVM elements with one end connected to horizontal wire (BL) and the other end connected with vertical wire (SL). The basic operation to read a memory cell is by applying a voltage pulse across the selected element and read the corresponding current.

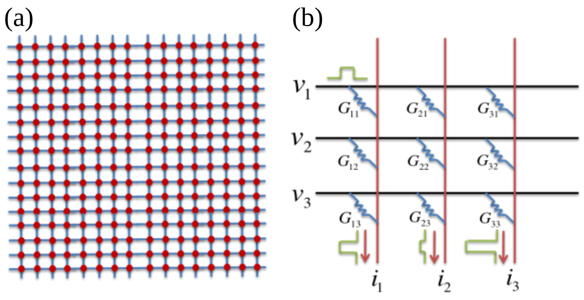


Figure 1. (a) Structure of non-volatile memory crossbar array. (b) Use NVM crossbar array to perform multiply and accumulate (MAC).

We can use this structure to perform multiply and accumulate (MAC) based on Ohm's law (Figure 1(b)) for the programmable conductance of each element. Such operation can be used to implement affine transformation layer in a multi-layer neural network. For the details of the operation, we first encode an input vector V as voltage pulses applied on each row, and encode parameters matrix G as conductance of NVM elements inside the crossbar array. Then applying Ohm's law, we can compute the matrix vector multiplication: I = G*V by reading the resulted current. In this scheme, NVM devices act as both memory to store parameter value and computing unit to perform multiplication. The computation happens right at where the parameters are stored, therefore eliminating data movement. Ideally, under the scheme a MAC can be performed in O(1) time, independent of the input vector length. However in reality, it is still limited by the size of the crossbar array.

To construct a multi-layer network using the same crossbar structure at each layer, we need to convert the summed current into certain voltage. However, to build a high-precision voltage converter involves complicated and energy-hungry circuitry building such as multi-bit ADC, which can overshadow the efficiency of crossbar array [6][7]. We thus propose to constrain the inputs and the activations at each layer of the network to only take binary values of -1 and 1. Such implementation requires only an integrator and a simple voltage comparator, which is simple and efficient. Figure 2 shows the proposed architecture.

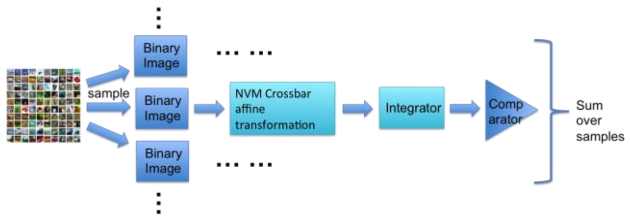


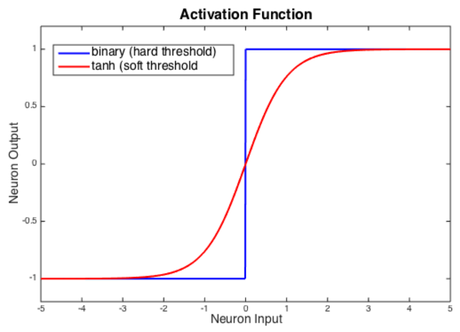
Figure 2. Proposed network architecture

III. Techniques for Binarizing Activation of Networks

As discussed in the introduction, there are two problems associated with using binary representations. The first one is that a discontinuous activation function is not compatible with back-propagation; and the second one is that a lot of real-world inputs are high-precision values. To address these issues, we applied the following three techniques:

1. Using derivative of tanh function to approximate gradient of binary activation function. The tanh function is chosen because it shares some essential similarities with a binary activation function. Figure 3 shows the activation function and its derivative for both binary and tanh neuron. The tanh neuron can be viewed as a "soft version" of the binary thresholding function: its value approach -1 or 1 with growing input magnitude, and its derivative is peaked at input equal to zero. Meanwhile, its gradient is non-zero for a relatively wide range of input, and thus during the gradient-descent training, the up-stream gradient signal can be back-propagated through the network.

(a)



(b)

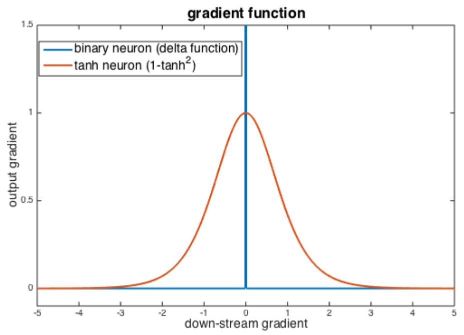


Figure 3. (a) Binary (hard threshold) and tanh (soft threshold) function. (b) Derivative of the binary and tanh activation functions.

2. Binarizing during training (In contrast to binarizing during testing). Now with tanh neuron, we defined binarizing during testing as using binary activations only during testing (when we deploy the model), whereas binarizing during training as using it also during the forward-propagation phase of the training. The main difference between these two binarizing approaches is that for binarizing during testing, tanh neuron is applied for both forward and backward propagation, and replaced with a binary neuron only when deploying the trained model; whereas for binarizing during training, binary neurons is applied during forward propagation of training and only derivative of tanh during back-propagation. Therefore the model is trained to work with binary activation from the very beginning. Figure 4 summarizes the workflow of these two different techniques.

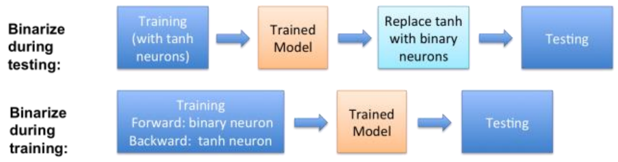

Binarize during testing:
Training (with tanh neurons)
Trained Model
Replace tanh with binary neurons
Testing
Binarize during training:
Training Forward: binary neuron Backward: tanh neuron
Trained Model
Testing

Figure 4. Comparison of the workflow between binarization during testing and binarization during training.

3. Stochastic multi-sampling for real-value input. In order to preserve information stored in high precision input data, we propose to use a probabilistic approach. First we normalize the input to be within the range [-1 1]. The normalized input is treated as the expectation value of a Bernoulli random variable x, where

$$x = \begin{cases} +1 & with\ prob = \varphi \\ -1 & with\ prob = 1-\varphi \end{cases}$$

The expectation value

$$E(x) = \varphi - (1-\varphi)$$

We can then calculate φ:

$$\varphi = \frac{E(x)+1}{2}$$

We can still train the model using high-precision input data. However deploying the trained model onto the crossbar hardware, we sample from the above distribution P(x) multiple times to get a set of binary inputs. Each binary input is passed through the entire network, and at the final affine transformation layer, where the output corresponds to scores of different classes if the network is used for a classification task for instance. The score is then summed up from each sample and the highest accumulated score one gives the winner category. With enough samples, the network can eventually recover the information of the high-precision input.

IV. Experiment Results

To evaluate the effectiveness of the proposed binarization techniques, we construct a three layer fully connected neural network with 256 neurons at each intermediate layer. The choice of 256 neurons is based on the array size that has already been demonstrated for the NVM crossbar structure [4]. We train and test the network on CIFAR-10 and MNIST datasets.

A reference model with real-value tanh neuron at all layers and high-precision input data is built for comparison. To train a good model closed to real optimum, we experiment on different sets of hyper-parameters including learning rate, regularization factors and initialization factor using cross-validation. The best models that we obtain are able to achieve 51.6% classification accuracy on CIFAR-10 and 97.7% on MNIST. With the reference, we conduct the following experiements.

1. Approxiamte gradient. As explained in the previous section, the gradient of tanh neuron is used to approximate gradient of binary activation function during backpropagation. Table 1 summarizes the results from binarization during testing and training. For binarizing during testing, we directly take the trained reference model and replace all neurons with binary threshold functions; for binarizing during training, we train a new model using binary activation function during forward propagation.

	Reference	Binarize During Testing	Binarize During Training
CIFAR-10	51.6	30.3	49.8
MNIST	97.7	80.7	97.4

Table 1. Experiment results of the proposed binarization techniques.

The results show that binarizing during training technique gives little loss in performance for both dataset, whereas directly replacing tanh neuron with binary neuron during testing leads to much worse performance, especially for CIFAR-10 the performance suffers a 41.3% drop compared to the reference. To analyze why binarizing during testing does not work well, we plot the distribution of value after the affine transformation and after the tanh non-linearlity for all layers in the reference model.

From the plot (Figure 5), the output of the affine transformation layers for CIFAR-10 dataset turns out to be very sparse. Most elements take values very closed to zero. And therefore, even after applying tanh non-linearity, which is supposed to encourage output to diverge into -1 or +1 direction, a large portion of output still centered around zero. Simply replacing this tanh function with binary threshold function will result in close to zero value which is forced to be either -1 or +1, and thus causing a large portion of information to be lost or distorted. The situation is less severe for MNIST dataset since the distribution peaked at two ends after tanh non-linearity. This distinction might originate from the nature of the dataset such as its size and complexity. However, for both dataset, we can conclude that using binarizing during training technique will yield a much better performance.

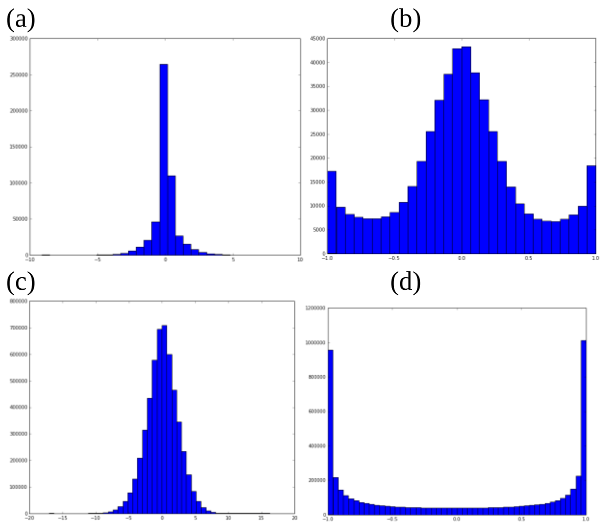


Figure 5. Output distribution of (a) the affine transformation layers for CIFAR-10 dataset (b) the tanh non-linearity layers for CIFAR-10 dataset (c) the affine transformation layers for MNIST dataset (d) the tanh non-linearity layers for MNIST dataset.

2. Stochastic multi-sampling for high-precision input. As explained above, to preserve information contained in high-precision data while still using only binary number for input, we treat input as Bernoulli random variables and sample multiple times from the distribution to get a set of binary value inputs during testing. As comparison, we also try to deterministically convert high-precision input into binary number based on if its value is greater than zero. Table 2 summarize the results:

	Binarize During Trainin g	**Determi nistic Binarize Input**	**Stochastic Multi-Sampling (8 samples)**	**Stochastic Muti-Sampling (64 samples)**
CIFAR-10	49.8	35.6	47.9	49.8
MNIST	97.4	97.0	97.5	97.7

Table 2. Experiment results of deterministic and stochastic input binarization methods.

The results show that with enough samples, the stochastic multi-sampling approach can eventually recover the performance of using high-precision input data. For both datasets, the stochastic approach could out-perform the deterministic approach. The MNIST dataset suffer much less loss in performance when performing deterministic binarization mainly because the gray-scale input images are already closed to black-and-white images that can be represented using binary numbers. Figure 6 shows that the performance of stochastic multi-sampling converges to high-precision input with increasing number of samples.

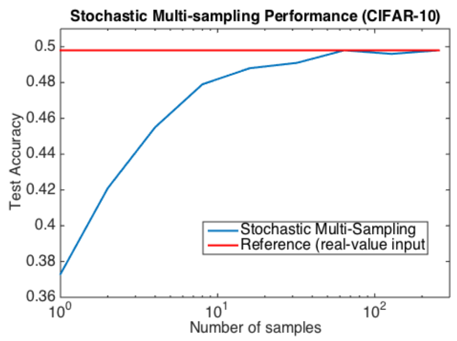
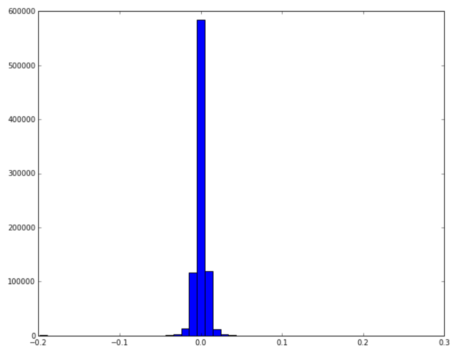


Figure 6. Performance of stochastic multi-sampling technique with different number of samples.

3. Bounded weights. Technically, the parameters of a neural network can take any value during training. But when we deploy a trained network onto a real NVM crossbar architecture, we need to consider that real NVM devices have limited range of conductance value and limited precision. As a result, we might need to clip the parameters with large value to be within a certain range. To study how much effect this will have on the model performance, we first plot the weight value distribution for a trained model on CIFAR-10. Figure shows that most weights take values very closed to zero, and only a very small fraction takes value with larger magnitude. Figure shows how the performance varies when we change the threshold value. We conclude that due to the sparsity of trained weights, clipping the weights usually will not lead to significant performance loss.

(a)



(b)

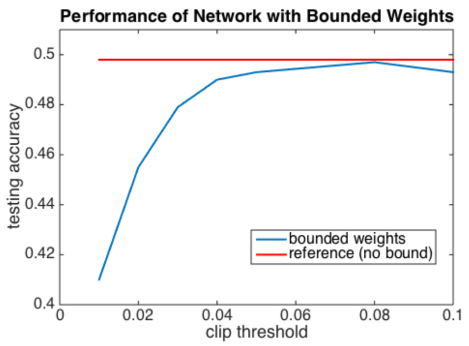


Figure 7. (a) Weights value distribution for a trained 3-layer fully connected neural network. (b) Performance of network with different weight-clipping thresholds.

V. Conclusion

In this paper, we propose techniques to train and deploy a multi-layer neural network with binary activations. Such network is compatible with a novel hardware architecture using NVM crossbar array and binary current-to-voltage converter. This hardware architecture is expected to provide orders of magnitude higher energy-efficiency than traditional hardware. Through our experiments of a 3-layer fully connected neural network on CIFAR-10 and MNIST datasets, we found that by using the derivative of tanh neuron to approximate gradient of discrete binary function, and using binary neurons during not only testing but also training, the network can perform almost as well as a network with high-precision tanh neurons. Furthermore, by applying stochastic multi-sampling method for real-value input data and sampling enough times, the network can work with purely binary input with no loss of performance. When deploying the model onto real NVM-crossbar hardware, the range and precision limitation of NVM devices will not have much effect on performance due to the sparsity of trained weights.

To further explore this topic, we suggest the following three directions. First, besides the limitation of conductance range, real NVM devices exhibit many non-idealities such as non-zero failure rate, stochastic distribution of conductance during programming; meanwhile the NVM crossbar array has problem with IR drop and sneak path. These non-ideal factors should be taken into consideration in the experiment in order to understand hardware-algorithm interaction. Second, the binary neural network discussed in this paper still has its last layer's output in high-precision value, representing the scores of each class. To make it fully-binary, one methodology one may be able to take is to play with samples obtained using the stochastic sampling approach. Finally, all experiments in this paper are conducted on a 3-layer fully connected neural network. If the same techniques can be applied to deeper network or other network architecture such as convolutional neural network is not yet to be known, and would be an interesting topic to be pursued.

Reference

[1] M. Horowitz. Energy table for 45nm process, Stanford VLSI wiki. [Online]. Available: https://sites.google.com/site/seecproject

[2] G. W. Burr, R. M. Shelby, J. W. Jang, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," *Iedm*, vol. 95120, no. 408, pp. 0–2, 2014.

[3] S. B. Eryilmaz, D. Kuzum, S. Yu, and H. S. P. Wong, "Device and system level design considerations for analog-non-volatile-memory based neuromorphic architectures," *Tech. Dig. - Int. Electron Devices Meet. IEDM*, vol. 2016–Febru, p. 4.1.1-4.1.4, 2016.

[4] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[5] P. Chen, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J. S. Seo, Y. Cao, and S. Yu, "Technology-Design Co-optimization of Resisti ive Cross-point Array for Acceler rating Learning Algorithm ms on Chip," *Proc. 2015 Des. Autom. Test Eur. Conf. Exhib.*, pp. 854–859, 2015.

[6] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *2016 ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, pp. 14–26, 2016.

[7] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," *Def. Adv. Res. Proj. Agency*, no. Figure 1, pp. 1–7, 2016.