Soft Binarized Neural Network for Inference using RRAM Crossbars with Minimal Quantization Errors

Rishi Nandha V (EE21B111), Satvik Malapaka (EE21B080)
Department of Electrical Engineering, IIT Madras

***Abstract*—The increasing demand for energy-efficient hardware to implement neural networks has become crucial, driven by the rising interest in Generative AI algorithms. RRAM crossbars have emerged as a promising solution for such implementations due to their high parallelism and energy efficiency. In this work, we propose a new technique, Soft Binarization, which approximates binary behavior of weights during the ex-situ training to align neural network models with RRAM crossbar characteristics. By incorporating this approach during training, quantization-induced errors are minimized, and hardware-software alignment is improved. Notably, we demonstrate that the accuracy drop due to quantization-induced errors can be reduced to as low as 2%.**

I. Introduction

The increasing computational demands of neural networks have led to significant interest in in-memory computing and neuromorphic computing paradigms. In-memory computing minimizes energy overhead by performing computations directly within memory, thereby eliminating the need for frequent data transfers between memory and processing units. A notable approach is to translate matrix multiplications—central to neural network operations—into the interaction of input voltages with stored conductance values, producing current outputs.

This functionality can be realized using **RRAM-based crossbar arrays**, where electrodes intersect at memory cells that simultaneously store synaptic weights (as conductance) and perform computation. **Resistive Random-Access Memory (RRAM)**, a non-volatile device exhibiting electric hysteresis, can reliably be programmed into **ON** or **OFF** states. This makes crossbars with RRAMs as the memory cells an attractive solution for implementing energy-efficient neural network accelerators.

Previous work has demonstrated the feasibility of implementing neural networks using RRAM crossbar arrays. Li *et al.* [1] successfully employed a crossbar-based approach to implement a multilayer perceptron. While their primary work was introducing a feedback-based programming method to support continuous weights rather than binary (ON/OFF), they've also demonstrated that converting continuous weights to binary weights, resulted in a significant accuracy drop of about **15.6%**. We refer to such a drop in accuracy due to binarization of weights as the **quantization error in inference**.

In this work, we proceed with a binary programming approach and focus on minimizing quantization errors by accounting for the binary nature of weights during ex-situ training. Although this method introduces a slight accuracy reduction in software inference, it ensures a more reliable performance replication in hardware inference. To achieve this, we propose a technique termed "Soft Binarization". Additionally, we also use activation functions that closely fit the characteristic of the real circuit element that will be used in the hardware implementation. With this methodology, we demonstrate that the accuracy drop can be reduced to as low as **2%** for the same dataset used by Li *et al.*[1]. The training set is illustrated in Fig.1.

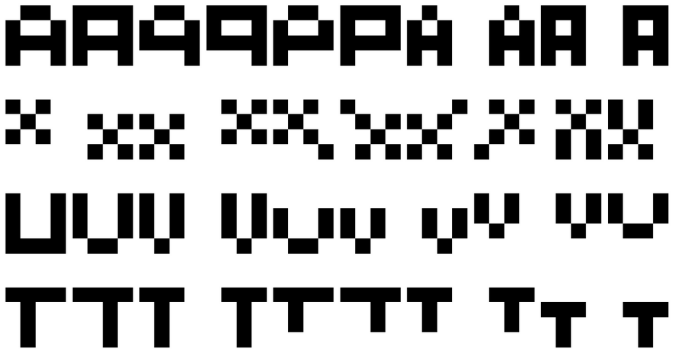

Fig. 1. Training Images from the Dataset

II. Soft Binarized Neural Network

To model the behavior of the RRAM cells, we approximate that, under the same writing voltage, the conductance in all RRAM cells identically becomes the same, thereby neglecting variations. We denote this conductance in the ON state as G_{ON}. In reality, the conductance also exhibits non-linearity with respect to the reading voltage; however, to a small-signal extent, we neglect these deviations. Similarly, the conductance in the OFF state is denoted as G_{OFF}.

The table below summarizes the RRAM parameters used in the **Stanford compact model** [2], including the writing and reading voltages, observed G_{ON} and G_{OFF}, and the validity range of the small-signal approximation.

Gap Coefficient $\mathbf{g_0}$	Voltage Coefficient $\mathbf{V_0}$	Prefactor $\mathbf{I_0}$
0.94 nm	0.21 V	19.5 μA

Escape Velocity v_0	β [2]	γ_0 [2]	α [2]
1×10^7	9.0	30	0.5

V_{WRITE}	V_{READ}	G_{ON}	G_{OFF}	Small-Signal
1.5 V	$\pm$0.1 V	7.7×10^{-5}	2.88×10^{-6}	$\pm$ 0.15 V

To minimize the accuracy drop caused by quantization errors, we address the fundamental reason for this disparity—**the forward pass in conventional neural networks employs continuous weights**, whereas RRAM-based hardware is restricted to discrete values G_{ON} or G_{OFF}. A straightforward approach would be to binarize the weights in software; however, this introduces a non-differentiable computation graph, causing the back-propagation process to fail.

To resolve this issue, we apply a sigmoid or tanh activation to the weights, which approximates a binary step function while retaining differentiability. This operation is performed prior to matrix multiplication during forward propagation, with the range of the weights constrained between G_{ON} and G_{OFF}. Additionally, we introduce a prefactor ζ, referred to as the **sharpness**, as a hyperparameter to prevent gradients during back-propagation from vanishing.

$$\textbf{soft\_bin}\,(x) = (G_{\text{ON}} - G_{\text{OFF}}) \cdot \textbf{sigmoid}\,(\zeta \cdot x) + G_{\text{OFF}} \quad (1)$$

Thus, the forward propagation equation is defined as follows, where the superscript corresponds to the layer index, the subscript corresponds to the neuron's index within that layer, g denotes the activation function of the layer, and a represents the activation of the neuron:

$$a_i^{[l+1]} = g^{[l+1]}\left(\sum_k \textbf{soft\_bin}\,(w_{ik}) \cdot a_k^{[l]}\right) \quad (2)$$

For our dataset, we implement a neural network with three layers, containing 16, 8, and 4 neurons, respectively. The computation graph of the network is illustrated in Fig. 2. Soft_Bin() is the function discussed above. InvAmp() is the activation function discussed in the next section

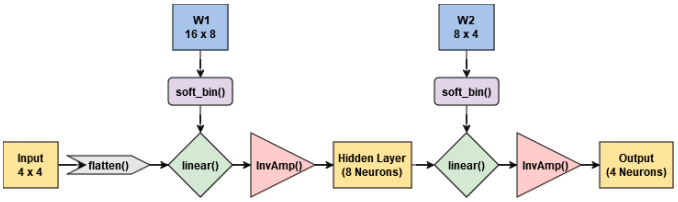

Fig. 2. Computation Graph

III. Inverting Amplifier Activation

The neural network needs an activation function between the two layers to effectively learn non-linear trends in the training data. The current mode output of the RRAM crossbar also needs to be converted into a voltage mode signal for the next crossbar's input. To serve both of these purposes, we design an inverting amplifier using an inverter in negative feedback with a resistor.

As a result, in the forward pass of the network shown in figure 2, InvAmp() is defined as the following:

$$\textbf{InvAmp}(x) = V_{\text{RAIL}} \cdot \tanh\left(\frac{x \cdot R_{\text{fb}}}{V_{\text{RAIL}}}\right) \quad (3)$$

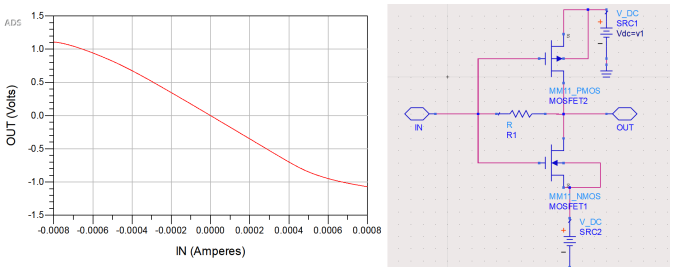

Fig. 3. **a.** InvAmp() Charecteristic **b.** Inverter in Negative Feedback

Figure 3 shows one such Inverting Amplifier and it's input current vs output voltage characteristic showing a tanh behavior.

Note that the choice of R_{fb} is critical to how the network behaves because it gives the gain of this amplifier stage. A resistance that is too large will result in input voltages to the second crossbar stage to exceed the small-signal limits. A resistance that is too small will not produce legible results.

IV. Training & Implementation

For training the neural network, a learning rate warm-up and fine-tuning schedule were applied. The network, implemented as a custom PyTorch class with circuit parameters as hyperparameters, utilized Xavier Uniform initialization with a small spread. Voltage values after the first inverting amplifier stage were monitored during training to optimize R_{fb}. Data augmentation included Dropout and noise injection. Input 1s and 0s were mapped to $+V_{\text{READ}}$ and $-V_{\text{READ}}$. Figure 4 presents the accuracy and loss curves over epochs.

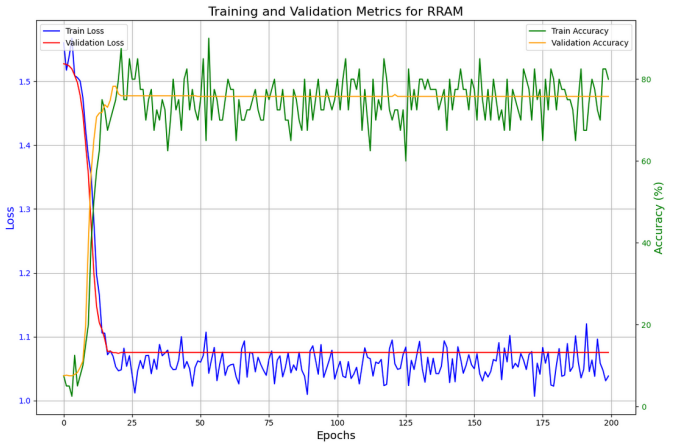

Fig. 4. Training Plot

The following table summarizes parameters used:

ζ	V_{RAIL}	R_{fb}	V_0	V_1
500	1.2 V	2 kΩ	0.1 V	-0.1 V

For programming these, 100μs pulses and a V/3 Scheme were used. We have to isolate the crossbars during programming and then connect them through the Inverting Amplifiers

for inference, hence Analog MUXes were made and used. Figure 5 shows the internal construction

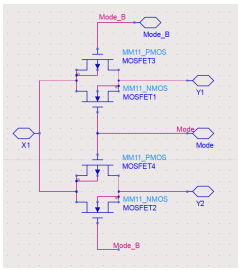

Fig. 5. Analog MUX using Transition Gates

Figure 6 shows a reduced equivalent that demonstrates how the switching between programming and inference is implemented and shows the complete path from the input till the output through the RRAMs, MUXes and Inverting Amplifiers

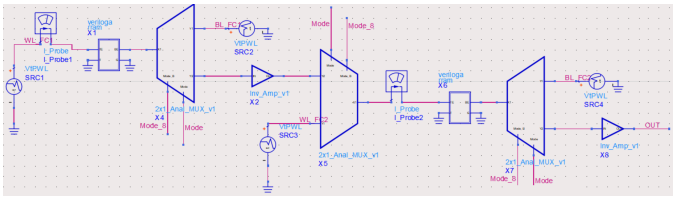

Fig. 6. Setup for switching between programming and inference

Our notebook used for training the Neural Network including the Custom PyTorch class can be found here and our final model file can be found here

V. Results

Figure 7 shows the complete circuit for the 3 - layer Neural Network made of 2 RRAM Crossbars, Inverting Amplifiers and Switching MUXes

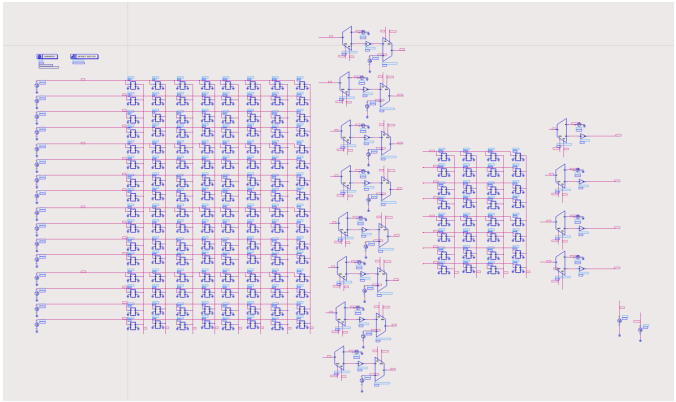

Fig. 7. Full Circuit

Figure 8 shows the output voltages for inference of 16 images as a short demonstration of the appropriate working of the setup. (RED = A_pred, NAVY BLUE = X_pred, PINK = V_pred, LIGHT BLUE = T_pred)

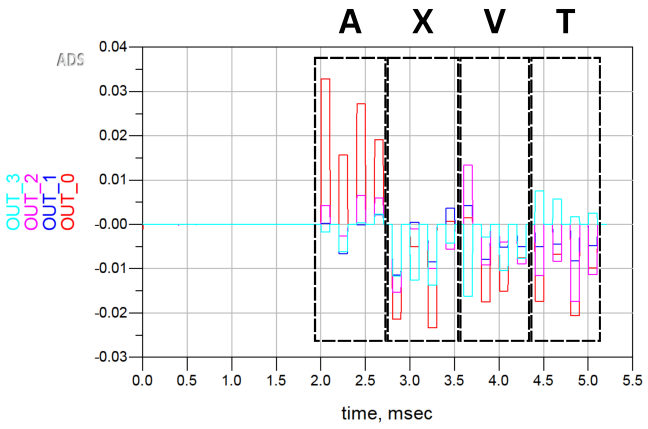

Fig. 8. Demonstration of Inference on 16 Images

Further, we compare software training accuracy, software testing accuracy and simulation testing accuracy on 160 testing images in the table presented below

Software Training Accuracy	**87.5%**
Software Testing Accuracy on 640 Images	**78.3%**
Software Testing Accuracy on 160 Images	**80.6%**
Simulation Testing Accuracy on 160 Images	**78.8%**

The table below compares the Accuracy Drop due to Quantization Error in this work and that shown in [1] with binary weights ("Hardware-Unaware" as per that work's terminology).

	Software Acc.	Crossbar Acc.	Drop due to Quantization
Li *et al.* [1]	**95%**	**79.4%**	**15.6%**
This work	**80.6%**	**78.8%**	**1.8%**

Figure 9 shows the scatter plot of predictions. Our ADS Workspace can be found here

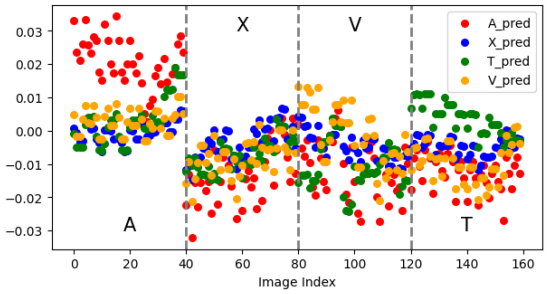

Fig. 9. Predictions Scatter Plot

References

[1] H. Li et al., "Analogue signal and image processing with large memristor crossbars," Nature Communications, vol. 9, no. 1, p. 2385, 2018, doi: 10.1038/s41467-018-04482-4

[2] Z. Jiang et al., "A compact model for metaloxide resistive random access memory with experiment verification", IEEE Trans. Electron Devices, vol. 63, no. 5, pp. 1884-1892, May 2016