Results for 3D Crossbar feed forward algorithm

June 6, 2024

Contents

1 Algorithm

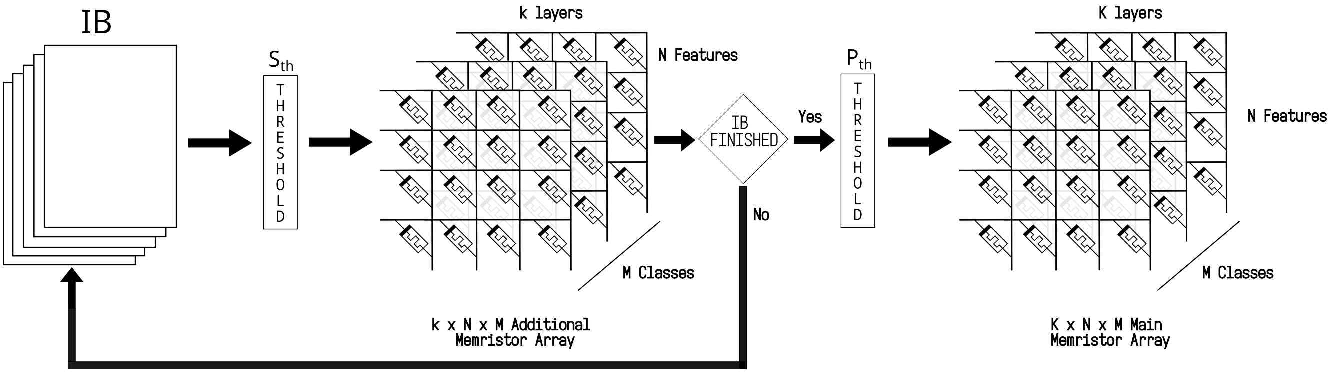

Figure 1: Flow of the algorithm.

The overall flow of the proposed learning algorithm is depicted in Fig. 1. We assume that N is the number of input attributes, and M is the number of classes in the classification problem. In the figure, NN-chip refers to the $N \times M \times K$ memristor crossbar that is used in learning, and the additional memristors (AM) denote an $N \times M \times k$ memristor array used for thresholding. Here, $K = \lceil \frac{total\_number\_of\_samples}{k} \rceil$ and $k = \lceil \log_2(batch\_size) \rceil$.

In the algorithm, inputs are applied one at a time from an input batch $\mathbf{IB} = \{\mathbf{IB}_0, \mathbf{IB}_1, \ldots, \mathbf{IB}_{K-1}\}$, where each batch $\mathbf{IB}_i$ consists of k sub-batches $\mathbf{IB}_i = \{\mathbf{SB}_{i0}, \mathbf{SB}_{i1}, \ldots, \mathbf{SB}_{ik-1}\}$. Let k denote the number of inputs in $\mathbf{SB}_{ij}$, for all $0 \leq i < k$. Each sub-batch $\mathbf{SB}_{ij}$ consists of $Minputs$, one from each class. The inputs in the sub-batches are applied for training in parallel with respect to the classes. For each class i, every sub-batch $\mathbf{SB}_{ij}$ will first update the AMs.

On the host, we have two arrays called P_{th}, S_{th} both of size M i.e. the number of classes. Additionally, two learning rates, $alpha$ and $beta$ are there for training P_{th} and Sth respectively.

For the MNIST dataset, each image has 28 × 28 pixels, where each pixel is treated as a single attribute, and hence $N = 784$. When an input from a sub-batch is processed, it is first passed through a threshold stored in P_{th} based on its class. If the given input attribute is greater than the threshold, a voltage V_{sc} sufficient to increase the conductive state by one unit is applied to the respective memristor in AM. Since the slices of the classes in the Additional memristor Crossbar Array are not connected, we can process these updates in parallel in a single cycle, in a manner similar to MAGIC. The whole sub-batch is processed in this manner. The process is repeated until there are no sub-batches left in the Input Batch. After this stage, the AMs are loaded with useful information in the form of weights.

The weights of the NN-chip memristors in the column corresponding to the class in which input inp belongs are then updated. A threshold filter is applied at this level to achieve weight updation, which helps to classify the most and least matched information. Based on the filtered values stored in AM after thresholding, the weights are updated in the respective column of the NN-chip. All the AMs are then reset to state 0, and the process is repeated for the next input batch until all the K input batches are processed and the NN is at capacity.

After this, a projection of the NN Crossbar is taken along K layers to get an $N \times M$ matrix. The training inputs of size $N \times 1$ are multiplied with the projected matrix to get the similarity with each of the M classes. The index with the highest value is the predicted class. All the inputs are processed and the wrong predictions per-class are stored. This is used to update the thresholds in P_{th} and S_{th} by multiplying the wrong predictions with the respective learning rates.

This process is repeated over a number of epochs until the desired accuracy is achieved.

1.1 Latency

Let's assume: a dataset of size 'D' where data within a single class forms a batch and the number of classes equals the number of IBs (both denoted by 'M'). We'll also assume that training input counts are equal across classes, leading to 'K' sub-batches per IB_i and each IB_{ij} having a size of 'k'.

Algorithm 1 Feed Forward Learning Algorithm with Threshold Updates

INPUT: Number of NN layers (k), input feature dimension (n), number of classes (M), AM Crossbar bit resolution (q), total number of training samples ($total\_samples$), input masking threshold (P_{th} initial), scaling threshold (S_{th} initial), learning rate for scaling threshold (α), learning rate for masking threshold (β)

OUTPUT: Trained NN Crossbar weights, updated thresholds (P_{th} final, S_{th} final)

$batch\_size = 100$
Initialize NN Crossbar: $NN \leftarrow$ Crossbar(K, N, M)
Initialize AM Crossbar: $AM \leftarrow$ Crossbar(k, N, M)
for `epoch ∈ range(0, epochs)` **do**
  for `batch_index ∈ range(0, K)` **do**
    mask $\leftarrow$ get_mask$(input_batch, P_{th})$
    $AM.update\_layer$(mask)
    if `batch_index % batch_size == 0` **then**
      am_mask $\leftarrow$ get_mask$(AM.get_xy_sum_projection(2), S_{th})$
      $ResetAM$
      $NN.update\_layer$(am_mask)
    end if
  end for
  $out \leftarrow NN.get\_xy\_sum\_projection(2)$
  normalize(out)
  wrongs $\leftarrow$ test(out)
  for `i ∈ range(0, M)` **do**
    S_{th}[i]$+= \alpha \cdot$ wrongs[i]
    P_{th}[i]$+= \beta \cdot$ wrongs[i]
  end for
end for
return out (trained weights)

Since updating AMs takes one cycle per training input, a complete sub-batch needs 'p' cycles for AM updates. We'll also need one cycle for the NN-chip update and another to reset the AMs. Therefore, a single sub-batch's processing requires a total of $C_{IB_{ij}} = k + 2$ cycles. To process a whole class, we multiply the cycles for a single sub-batch by the number of sub-batches in that class: $C_{IB} = C_{IB_{ij}} \times K$.

Calculating the cycles for training the entire dataset over a single epoch involves summing the cycles for each class: $C_D = C_{IB}$, since they are computed parallelly. The algorithm usually needs around 8 to 10 epochs for optimal accuracy, so the average training cycle count is approximately $10 \times C_D$.

2 Results

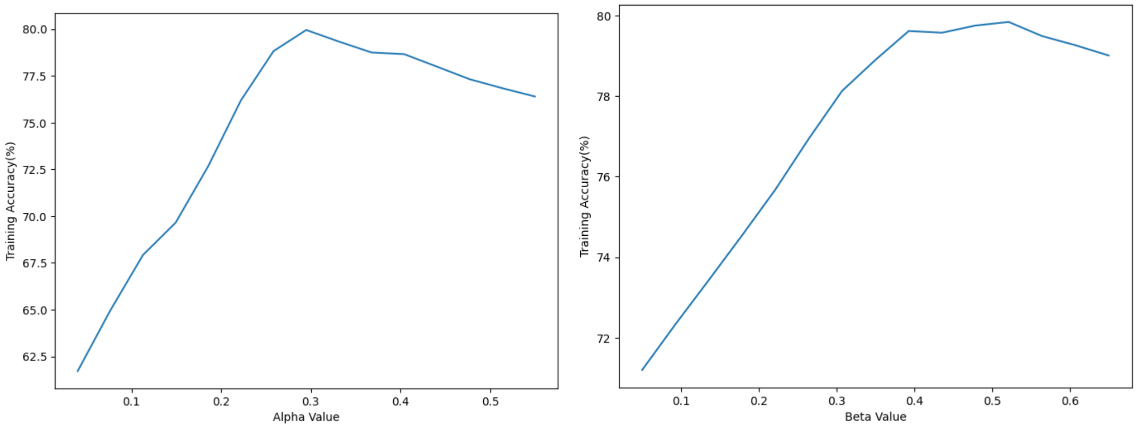

Figure 2: (a) Impact of alpha on the training, (b) Impact of beta on the training.

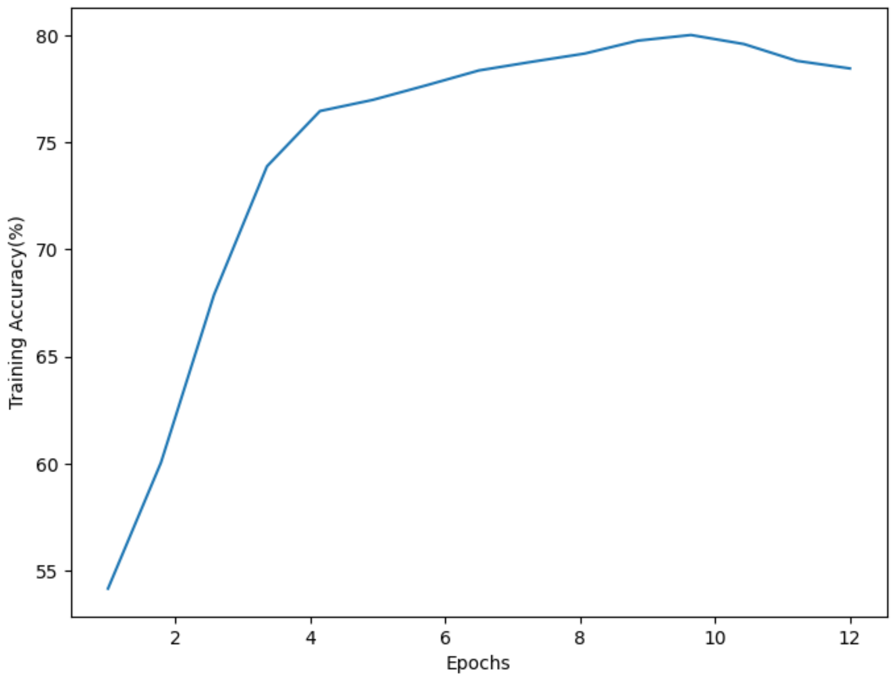

Figure 3: Progress of the training process over 12 epochs.

Table 1: Performance comparison of the proposed algorithm.

Approach	Description	Input size	Net-Size	Accuracy	# of epochs	Accuracy%
Proposed	HSBT	784 (28 × 28)	784→10	80.00%	10	–
[1]	HSBT	784 (28 × 28)	784→10	94.00%	10	-14
[2]	HSBT BP	760 (19 × 20)	760→10	89.90%	–	-9.9
[3]	HSBT BP	49 (7 × 7)	50→10	92.00%	–	-12
[4]	HABT BP	784 (28 × 28)	784→ 42→10	92.00%	1000	-12
[5]	HSBT BP	784 (28 × 28)	784→10	83.85%	10	-3.85
[6]	HSBT RWC	784 (28 × 28)	AlexNet	94.79%	1000	-14.79
	HSBT BP + RWC			98.81%		-18.81
	HSBT Delta Rule			85.19%		-5.19
	HSBT RWC+Delta			97.95%		-17.95
[7]	HABT (OCTAN)	64 (8 × 8)	64→100→10	81.80%	52	-1.8
[8]	HABT SLMS	25 (5 × 5)	–	84.00%	250	-4.00
	HABT LMS			82.00%		-2.00

References

[1] Dev Narayan Yadav, Phrangboklang Lyngton Thangkhiew, Kamalika Datta, Sandip Chakraborty, Rolf Drechsler, and Indranil Sengupta. Feed-forward learning algorithm for resistive memories. *Journal of Systems Architecture*, 131:102730, 2022. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2022.102730. URL https://www.sciencedirect.com/science/article/pii/S1383762122002156.

[2] Miao Hu, Catherine E. Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R. Stanley Williams, J. Joshua Yang, Qiangfei Xia, and John Paul Strachan. Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018. doi: https://doi.org/10.1002/adma.201705914. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201705914.

[3] Peng Gu, Boxun Li, Tianqi Tang, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. Technological exploration of rram crossbar array for matrix-vector multiplication. In *The 20th Asia and South Pacific Design Automation Conference*, pages 106–111, 2015. doi: 10.1109/ASPDAC.2015.7058989.

[4] Olga Krestinskaya, Khaled Nabil Salama, and Alex Pappachen James. Learning in memristive neural network architectures using analog backpropagation circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(2):719–732, 2019. doi: 10.1109/TCSI.2018.2866510.

[5] Dev Narayan Yadav, Kamalika Datta, and Indranil Sengupta. Analyzing fault tolerance behaviour in memristor-based crossbar for neuromorphic applications. In *2020 IEEE International Test Conference India*, pages 1–9, 2020. doi: 10.1109/ITCIndia49857.2020.9171788.

[6] Shyam Prasad Adhikari, Changju Yang, Krzysztof Slot, Michal Strzelecki, and Hyongsuk Kim. Hybrid no-propagation learning for multilayer neural networks. *Neurocomputing*, 321:28–35, 2018. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.08.034. URL https://www.sciencedirect.com/science/article/pii/S0925231218309846.

[7] Mohammad Ansari, Arash Fayyazi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. Octan: An on-chip training algorithm for memristive neuromorphic circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(12):4687–4698, 2019. doi: 10.1109/TCSI.2019.2934560.

[8] Cory Merkel and Dhireesha Kudithipudi. A stochastic learning algorithm for neuromemristive systems. In *2014 27th IEEE International System-on-Chip Conference (SOCC)*, pages 359–364, 2014. doi: 10.1109/SOCC.2014.6948954.