BATMANN: A Binarized-All-Through Memory-Augmented Neural Network for Efficient In-Memory Computing

Yuan Ren*, Rui Lin*, Jie Ran, Chang Liu, Chaofan Tao, Zhongrui Wang, Can Li, Ngai Wong*
Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong

***Abstract*—The traditional von Neumann architecture suffers from heavy data traffic between processing and memory units, which incurs high power and latency. To cope with the booming use of neural networks on edge devices, a promising way is to perform in-memory computing through exploiting the next-generation memristive devices. This work proposes a 2-level resistive random-access memory (RRAM)-based memory-augmented neural network (MANN), named binarized-all-through MANN (BATMANN), that is end-to-end trainable and allows both the controller and memory to be seamlessly integrated onto RRAM crossbars. Experiments then show the superiority of BATMANN in doing few-shot learning with high accuracy and robustness.**

***Index Terms*—RRAM, memory augmented, binary, neural networks, in-memory computing**

I. Introduction

The past decade of escalated advancement in machine learning (ML) and artificial intelligence (AI) have achieved unprecedented success in applications such as computer vision and natural language processing [1], [2]. Deep neural networks (DNNs) via deep learning have surpassed various traditional solutions, and spurred the industry to adapt to the ubiquitous AI technologies. Nonetheless, despite the proliferation of software-oriented AI algorithms and DNN architectures, their hardware realization is facing stiff challenges. Existing von Neumann computing architecture incurs massive data traffic between processing elements and memory units, resulting in high power consumption and latency. This is compounded by the growing need of edge (viz. user-end) AI that is largely constrained by resource-limited hardware and stringent power budget. All these are promting the quest for an advanced microelectronics platform that supports low-power, high-speed neural networks to sustain the evolution of the AI era.

To this end, in-memory computing on non-volatile memory (NVM) has emerged as an attractive solution for realizing AI accelerators, namely, by processing data directly on-site in memory to minimize data transfer. Various NVM candidates include the resistive random-access memory (RRAM), phase-change memory (PCM), magnetic random-access memory (MRAM), etc. Among these, RRAM cells show the best reliability and compatibility with complementary metal-oxide-semiconductor (CMOS) processes, and can be readily arranged into crossbars for performing matrix-vector product to largely speedup AI inference via analog computing [3]. Such advantages render RRAM a promising candidate for in-memory computing to fuel the next-generation AI.

Building upon the recent demonstration of a memory-augmented neural networks (MANN) stitching DNN and hyperdimensional computing for few-shot learning [4], this work provides two major advancements:

- A binarized-all-through MANN, called BATMANN, is proposed that uses simple 2-level RRAM cells and allows seamless integration of both the controller (encoder) and memory unit onto RRAM crossbars. To our knowledge, BATMANN is the *first* purely RRAM controller-memory integration, distinguishing itself from the memory-only realization in [4].
- It is shown *for the first time* that state-of-the-art binary neural network (BNN) design techniques can be adapted to the end-to-end training of a MANN giving high output accuracy. This provides high research and practical insights into next-generation in-memory computing.

II. Background

A. Memory-Augmented Neural Network (MANN)

Recurrent Neural Networks (RNNs) are capable of exploring the temporal dependencies in input sequences of variable lengths. However, as the input length increases, RNNs suffer from vanishing gradients, exponential growth in number of parameters and costly computation due to increased memory size. Long Short-Term Memory (LSTM) [5] and Gated Recurrent Unit (GRU) [6] have been developed to alleviate the issue, but both embed the history into a single hidden vector as in RNNs, which means they still have difficulties searching through past memories to make a prediction. MANN is proposed to alleviate the gradient vanishing problem as it satisfies two criteria [7]: that the information stored in the memory is stable and element-wise addressable, and the number of learnable network parameters are not tied with the size of the memory.

There are two main components in a MANN, namely, a controller and a memory. The controller can learn how to read from and write to the memory, while the memory in a MANN is usually a content addressable memory (CAM) comprising a key and a value memory. The key memory stores and compares the learned patterns, while the value memory holds

*YR {renyuan@eee.hku.hk}, RL and NW contributed equally to this work.

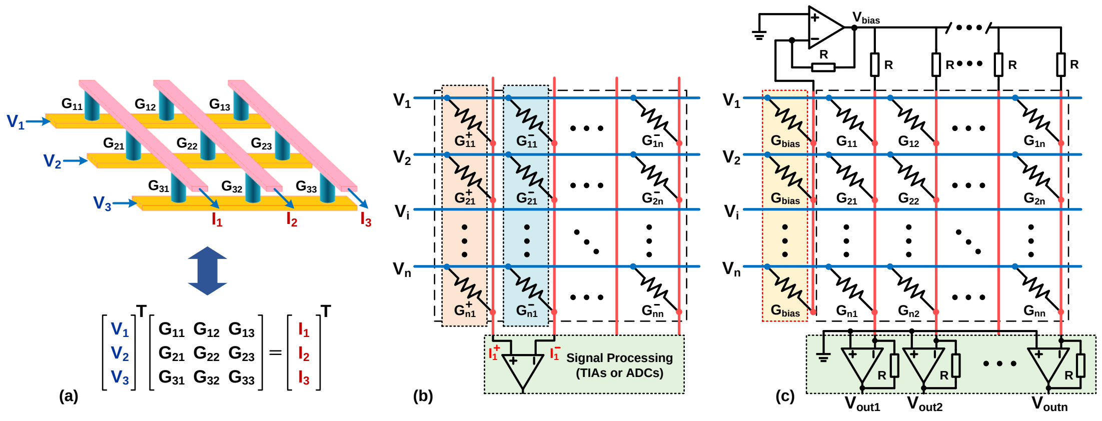

Fig. 1: (a) Vector-matrix multiplication computation in an RRAM-based crossbar; (b) Implementation of negative synaptic weight (double-column approach); (c) Implementation of negative synaptic weight (single-column approach).

the labels. It is noticeable that with CAM, the new information can be offloaded to the explicit memory without the risk of overwriting previously learned content, thus making MANN an excellent candidate for one-/few-shot learning tasks.

B. RRAM Crossbars

An RRAM crossbar/array readily implements the vector-matrix multiplication (VMM) which constitutes the workhorse operation in DNN inference. Specifically, Fig. 1(a) depicts a metal–RRAMs–metal (MRM) stack where the two-terminal RRAM cells are programmed with different conductance values G_{ij}'s to encode the matrix weights in a DNN layer, e.g., a convolutional neural network (CNN) or fully connected (FC) layer. The voltages V_i's are parallelly fed into each row as inputs, producing the outputs I_j's summed on each column according to the Ohm's law and Kirchhoff's law. Such hardware-based analog computing can efficiently process VMM in a single pass. The currents sensed from different columns are then collected via transimpedance amplifiers (TIAs) or analog-digital converters (ADCs) for back-end signal processing. In practice, each layer in a pretrained DNN model can be mapped onto a tiled architecture of RRAM crossbars [8], and the same `im2col` framework is applied to transform both CNN and FC operation into standard VMM format for acceleration on the crossbar-based in-memory computing platform.

III. BATMANN

A. Software: Design Algorithm

We harvest the latest development in binary neural network (BNN) training algorithms, namely, XNOR-Net [9] and RBNN [10], for designing BATMANN using 2-level RRAM cells throughout. In particular, XNOR-Net attempts to minimize the quantization error arising from mapping the full-precision weights to their (bipolar, i.e., ± 1) quantized levels with a learnable per-channel scaling factor. Whereas RBNN further accounts for the angular bias between the full-precision and bipolar weights and tries to minimize it during training.

To concisely describe the BATMANN design flow, we use X_t and X_v to denote the training and validation datasets, respectively. Their corresponding labels are Y_t and Y_v. The training dataset X_t is constructed by items from M classes, and each class contains a few samples. It is worth noting that M is generally large, while the number of samples in each class is often scarce and far from being ample as in MNIST or ImageNet. For example, in the omniglot [11] dataset widely used for few-shot learning (FSL), it contains 1623 characters, each with only 20 samples written by different people. For X_v, it has the same structure as X_t, but the classes in it are often disjoint with X_t.

Algorithm 1 describes the BATMANN training on software. For the controller in our proposed BATMANN, we highlight that its last FC layer is also binarized rather than 8-bit or full-precision as in most BNNs. In addition, when making predictions during the learning phase, we use the dot product to calculate the similarity and choose standard absolute function to be the sharpening function (Algorithm 1, line 8) instead of cosine similarity and soft absolute, as employed in [4]. The whole workflow and architecture of BATMANN are visualized in Fig. 2, the binarized last FC layer renders the BATMANN hardware-centric and homogeneous to 2-level RRAM cells, without loss in output accuracy as will be demonstrated by experimental results in Section IV.

B. Hardware: Circuit Specifics

A challenge when implementing a DNN model on RRAM-based crossbars is that the synaptic weights in a DNN layer can be positive or negative, whereas the conductance of an RRAM cell is always physically positive. Various efforts [12], [13] have derived different circuit schemes to implement negative synaptic weights, which can be classified into the double-column (DC) and single-column (SC) approaches. In the DC approach, each synaptic weight is programmed into an RRAM cell pair, whose signed values are encoded into G_{ij}^{+} and G_{ij}^{-}, respectively (Fig. 1(b)). For the first differential columns, the output current of the positive column (I_1^{+}) and negative column (I_1^{-}) equals the dot product of the voltage vector V_i by the conductance G_{i1}^{+} and G_{i1}^{-} respectively. The subtraction of the differential pair ($I_1^{+} - I_1^{-}$) can be obtained

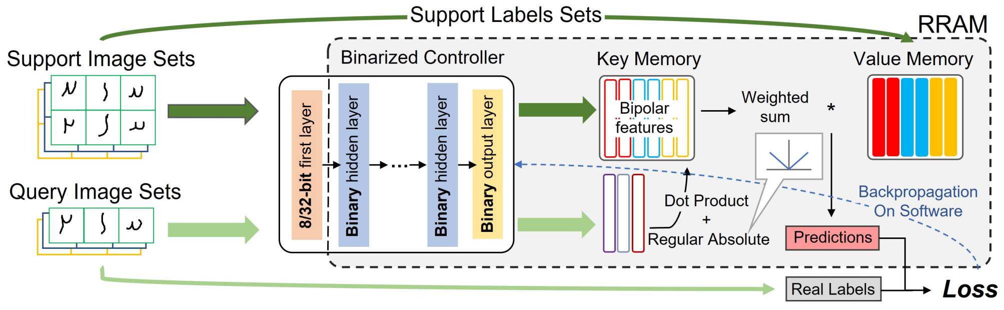

Fig. 2: The architecture of the proposed BATMANN implemented on RRAM, which contains not only the key-value memory as in [4] but also a binarized controller, all realizable with 2-level RRAM cells. It is worth noting that: 1) only the first layer of the controller is 8/32-bit, whereas the remaining layers (including the last FC layer) are all binarized; 2) the similarity measure and the sharpening function are consistent during learning and inference phases, without gradient approximator (e.g., softabs [4]) during backpropagation. These unique features in BATMANN permit neat alignment between software training and hardware implementation, and deliver higher system accuracy.

Algorithm 1 BATMANN Training

Input: Training data $\{X_t, Y_t\}$, validation data $\{X_v, Y_v\}$, a randomly initialized controller with parameters Θ, learning rate η, number of ways m (a subset of the total number of M classes), number of shots n, number of queries in each class n_q (in learning phase) and n_v (in validation), training episode N_t, validation episode N_v, validation interval N_i, and validation threshold t.
Output: A ***binarized*** mature controller with trained parameters $\hat{\Theta}$.

```
1:  for $i = 1, \cdots, N_t$ do
2:      $X_t^S \leftarrow$ Sample $m$ classes from $X_t$, and from each class sample $n$ items to generate the support set.
3:      $Y_t^S \leftarrow$ Get the corresponding one-hot labels.
4:      $X_t^Q \leftarrow$ For each class, randomly choose $n_q$ items from the remaining samples to obtain the query set.
5:      $Y_t^Q \leftarrow$ Get the corresponding one-hot labels.
6:      $F_t^S$, $F_t^Q \leftarrow$ Get the bipolar features by passing $X_t^S$ and $X_t^Q$ through the binarized controller.
7:      Key-Value Memory $\leftarrow$ Store $F_t^S$ and $Y_t^S$
8:      $\hat{Y}_t^Q \leftarrow$ Predict the labels by comparing the similarities between $F_t^Q$ and $F_t^S$ through dot product and absolute function.
9:      $L \leftarrow$ Use $\hat{Y}_t^Q$ and $Y_t^Q$ to calculate the loss, using XNOR or RBNN training scheme.
10:     $\Theta := \Theta - \eta \cdot \frac{\partial L}{\partial \Theta}, \hat{\Theta} := \Theta \leftarrow$ Update the parameters.
11:     if $\mathrm{mod}(i, N_i) == 0$ then
12:         Repeat lines 2 to 8 for $N_v$ times, but use $X_v$ and $Y_v$ to generate the support and query sets. # Validation
13:         $Acc_v \leftarrow$ Calculate the accuracy.
14:         if $Acc_v > t$ then
15:             break
16:         end if
17:     end if
18: end for
```

from $I_1^+ - I_1^- = \sum_{i=1}^n V_i(G_{i1}^+ - G_{i1}^-)$. In practice, the conductance values $G_{ij}^+, G_{ij}^- \in [G_{\min}, G_{\max}]$ are programmed by an extra large voltage, such that the value of $G_{ij} \in [G_{\min} - G_{\max}, G_{\max} - G_{\min}]$ is retained afterwards due to its memristive nature. The subtracted current is then sensed with a differential transimpedance amplifier (TIA) or analog-to-digital converter (ADC) for back-end signal processing.

Compared with the DC approach, half of the total RRAM devices are saved in the SC approach at the expense of extra peripheral analog circuits and one additional RRAM column as the bias function (Fig. 1(c)). The inputs of each layer are respectively delivered into bias RRAM cells. Meanwhile, the opposite terminals of bias RRAMs are collected together and connected to the negative terminal of a closed-loop operational amplifier. At the back-end of the RRAM crossbar, owing to Kirchhoff's law, the output current of each column is delivered into its own inverting amplifier for current-to-voltage conversion. Without considering nonidealities of the amplifier, the output voltage V_{outj} on the column j can be obtained from $V_{outj} = \sum_{i=1}^n V_i(G_{bias} - G_{ij})R$, which shows that the output of each column contains a $(G_{bias} - G_{ij})R$ factor that allows it to achieve negative synaptic weights on hardware by leveraging the conductance G_{bias} of the bias RRAM cells.

IV. Experiments

We trained the proposed BATMANN using PyTorch. All experiments were run on two Tesla V100-SXM2 Graphics cards, each with 32GB frame buffer. Table I shows the controller evaluation results on 20-way 5-shot FSL task based on the Omniglot dataset [11] under different learning and inference schemes. The first experiment is our own implementation of the learning and inference phases described in [4] whose codes are not released. Using it as the baseline, we observe that when using XNOR as the binarized controller training scheme, the accuracy after learning only slightly decreases by 0.07%. Comparing the third experiment with the baseline, using RBNN as the training scheme in the learning phase, the accuracy increases by 0.74%, indicating BNNs can extract the features well, in fact even better than regular full-precision controllers in this case. Experiments 3 and 5 are under the training settings described in Algorithm 1. It is worth noting that BATMANN$_X$ controller, which uses dot product and regular absolute not only in inference but also in the learning (viz. backpropagation) phase, has the best performance 96.30%. While the classification accuracy of BATMANN$_R$

No.	Learning				Inference				Acc. (%)
	Controller	Key	Similarity	Func	Controller	Key	Similarity	Func	
1	FP32	FP32	Cosine	Softabs	FP32	Bipolar	Dot	Abs	95.56
2	XNOR [9]	Bipolar	Cosine	Softabs	Bipolar	Bipolar	Dot	Abs	95.49
3	BATMANN$_X$	Bipolar	Dot	Abs	Bipolar	Bipolar	Dot	Abs	**96.53**
4	RBNN [10]	Bipolar	Cosine	Softabs	Bipolar	Bipolar	Dot	Abs	96.30
5	BATMANN$_R$	Bipolar	Dot	Abs	Bipolar	Bipolar	Dot	Abs	5.00

TABLE I: Controller evaluation results under different learning and inference schemes. Func: Sharpening function. FP32: full-precision controller. XNOR (RBNN): binarized controller with 8-bit (FP32) first conv layer and 8-bit last FC layer training in the XNOR (RBNN) scheme. BATMANN$_X$ (BATMANN$_R$): binarized controller with 8-bit (FP32) first conv layer and binarized last FC layer training in the XNOR (RBNN) scheme.

controller training under the same setting only scores 5.00%, i.e., a significant drop. BATMANN$_R$ fails to converge as standard absolute function, an approximation of the softabs, fails to update the rotation matrix to minimize the angular bias reaching the optimal parameters. Based on the above observations, we indicate that BATMANN (using XNOR) can have better performance than the full-precision controller, and it is much neater and hardware-centric.

When deployed on RRAM crossbars, the decline of performance is inevitable owing to the device variation of real-world RRAM cells. Subsequently, it is desirable to take the non-idealities into account and characterize the robustness of our design. We employ MemTorch [14] to map BATMANN onto RRAM crossbars to evaluate its performance under device uncertainties. Specifically, the DC approach is adopted (cf. Fig. 1 & Section III-B), but with a 1-Transistor-1-Resistor (1T1R) RRAM cell structure [15]. At the device level, the memristance $R_{on} = 1/G_{\max} = 1k\Omega$ and $R_{off} = 1/G_{\min} = 10k\Omega$ are set as in [15]. Based on the Omniglot dataset for FSL, the proposed BATMANN$_X$ network achieves a maximum accuracy of 96.53% for 20-way 5-shot. On top of BATMANN$_X$, we simulate the effect of stochastic, normally distributed device variation on the network classification accuracy. Fig. 3 plots the trend of accuracy degradation with an increased standard deviation σ from 0 to 1400 with respect to the memristance of each RRAM cell. It is seen that the output accuracy remains almost unchanged for σ up to 600, and drops sharply when σ goes beyond 1000 (mainly due to the programming failure of R_{on}). This further confirms the robustness of the proposed controller-memory-in-one BATMANN, on a par with the memory-only MANN in [4].

V. Conclusion

This paper has proposed a memory augmented neural network (MANN) wherein both the encoder and memory units are end-to-end trained and realized with RRAM crossbars using simple 2-level cells. Such first-of-its-kind binarized-all-through MANN (BATMANN) provides a promising solution for in-memory AI computing. Our numerical examples on few-shot learning have demonstrated the superiority of BATMANN in terms of system accuracy and robustness.

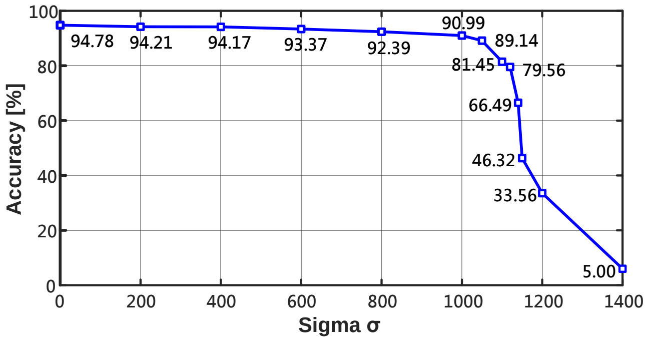

Fig. 3: Accuracy vs. RRAM device variation.

VI. Acknowledgement

This work is support in part by the General Research Fund (GRF) project 17206020, and in part by ACCESS – AI Chip Center for Emerging Smart Systems, Hong Kong SAR.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, pp. 770–778, 2016.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[4] G. Karunaratne, M. Schmuck, M. L. Gallo, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, "Robust high-dimensional memory-augmented neural networks," *Nat. Commun.*, vol. 12, 2021.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[7] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *ICML*, pp. 1842–1850, PMLR, 2016.

[8] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled rram architecture," in *IEEE IEDM*, pp. 14–4, IEEE, 2019.

[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, pp. 525–542, Springer, 2016.

[10] M. Lin, R. Ji, Z. Xu, B. Zhang, Y. Wang, Y. Wu, F. Huang, and C.-W. Lin, "Rotated binary neural network," in *Proc. NeurIPS*, pp. 7474–7485, 2020.

[11] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[12] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nat. Commun.*, vol. 4, no. 1, pp. 1–7, 2013.

[13] S. N. Truong and K.-S. Min, "New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog neuromorphic computing," *JSTS: J. Semicon Tech. & Sci.*, vol. 14, no. 3, pp. 356–363, 2014.

[14] C. Lammie, W. Xiang, B. Linares-Barranco, and M. R. Azghadi, "Memtorch: An open-source simulation framework for memristive deep learning systems," *arXiv preprint arXiv:2004.10971*, 2020.

[15] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, *et al.*, "In situ training of feed-forward and recurrent convolutional memristor networks," *Nat. Mach. Intell*, vol. 1, no. 9, pp. 434–442, 2019.