Efficient Hardware Implementation for Online Local Learning in Spiking Neural Networks

Wenzhe Guo[1], Mohammed E. Fouda[2], Ahmed M. Eltawil[1], Khaled Nabil Salama[1]
[1]Department of ECE, CEMSE Division, King Abdullah University of Science and Technology, Saudi Arabia
[2]Center for Embedded & Cyber-physical Systems, University of California-Irvine, Irvine, CA, USA 92697-2625

***Abstract*—Local learning schemes have shown promising performance in spiking neural networks and are considered as a step towards more biologically plausible learning. Despite many efforts to design high-performance neuromorphic systems, a fast and efficient neuromorphic hardware system is still missing. This work proposes a scalable, fast, and efficient spiking neuromorphic hardware system with on-chip local learning capability that can achieve competitive classification accuracy. We introduce an effective hardware-friendly local training algorithm that is compatible with sparse temporal input coding and binary random classification weights. The algorithm is demonstrated to deliver competitive accuracy. The proposed digital system explores spike sparsity in communication, parallelism in vector-matrix operations, and locality of training errors, which leads to low cost and fast training speed. Taking into consideration energy, speed, resource, and accuracy, our design shows 7.7× efficiency over a recent spiking direct feedback alignment method and 2.7 × efficiency over the spike-timing-dependent plasticity method.**

Keywords—Neuromorphic computing, spiking neural networks, local training, deep learning, backpropagation

I. INTRODUCTION

Brain-inspired spiking neural networks (SNNs) have attracted ever-growing attention from research communities for their superior energy efficiency to artificial neural networks (ANNs) and biological plausibility [1, 2]. Unlike the mature backpropagation (BP) algorithm for ANNs, the training methodology for SNNs is still not clearly defined. Unsupervised biological learning rules, such as spike-timing-dependent plasticity (STDP), are able to train SNNs on different tasks [3, 4]. STDP performs training by using local spike activities and makes hardware implementation simple and energy-efficient [5-7]. However, its training performance is limited. Recently, supervised training methods adapted from the BP algorithm were proposed to directly train SNNs, which produced accuracy close to ANNs in various tasks [8, 9].

Despite its effectiveness, the standard BP suffers from frequent memory access, massive matrix operations, computational inefficiency, and a lengthy training process, not suitable for hardware implementation [10, 11]. Thus, various local training algorithms based on BP were proposed to tackle the issues above [10-12]. Instead of waiting for the errors coming from the final layer, the local training methods use errors computed in the local classifier and train the corresponding layers, which significantly improves the training runtime. The whole network consists of training-isolated modules, making the hardware design scalable because the network can be built by simply cascading the same local training module. Few works have considered local training in SNNs [13, 14]. *Kaiser et al.* demonstrated the effectiveness of local training with random classifiers in SNNs. However, their complex training algorithm and neural models are not suitable for real-time deployment on constrained hardware platforms. Inspired by this work, we introduce a hardware-friendly and effective local training algorithm with good classification performance and propose a scalable and efficient digital spiking neuromorphic hardware architecture with on-chip local learning capability.

The contributions of this work are summarized as follows.

- We propose an efficient and hardware-friendly local training algorithm with time-to-first-spike neural coding.
- We apply binary random weights in the local classifiers to reduce hardware overhead without incurring performance degradation.
- We present a scalable hardware architecture with parallel processing for local training.

II. NEURAL MODELS AND LOCAL LEARNING

A. Time-to-first-spike coding

Various information coding schemes have been hypothesized to explain the information transmission in nervous systems including rate coding, phase coding. and burst coding [15]. Recently, time-to-first-spike (TTFS) coding was demonstrated to be fast and energy efficient as it only uses the a single spike to transmit information [15]. The coding method converts input pixels into single spikes within a time window by comparing the normalized input pixels with an exponentially-decaying threshold P_{th}, defined by

$$P_{th} = \exp(-t/\tau_{th}) \tag{1}$$

where τ_{th} is the decaying time constant. A spike is generated when the pixel exceeds the threshold, and the input is prohibited from generating more spikes.

B. Neuron models

To model the dynamics of spiking neurons, we used an integrate-and-fire (IF) model that consists of linear equations and a threshold condition because of its computational efficiency. The model can be written in the matrix form as

$$Z^l[t+1] = Z^l[t] + W^l S^{l-1}[t+1] \tag{2}$$

$$U^l[t+1] = Z^l[t+1] + b^l - \theta S^l[t] \tag{3}$$

$$S^l[t+1] = \Theta(U^l[t+1]) \tag{4}$$

where l indicates the layer number, W is the synaptic weight matrix, S^{l-1} is the input spike matrix, Z^l is the synaptic input matrix, U^l is the membrane potential matrix, b^l is the bias matrix, S^l is the output spikes matrix, θ is the reset constant, and $\Theta(\cdot)$ is a unit step function. An output spike is generated if the membrane potential is larger than 0.

C. Local learning model

The training process is illustrated in Fig. 1. Each layer in the network is attached with a local linear classifier with random weights. Predictions are computed by classifying the output spikes from neurons at the end of training window, which

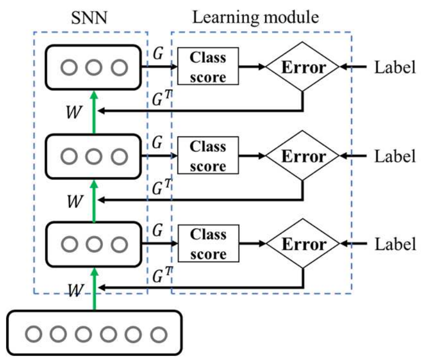

Fig. 1 SNN architecture with local learning algorithm.

simplifies the training algorithm. The prediction vector is expressed as

$$Y^l = G^l S^l[t_{win}] \tag{5}$$

where G^l is a constant random weight matrix, and Y^l is the prediction matrix, t_{win} is the length of training time window. We used a mean square error (MSE) loss function L^l for its simple hardware implementation. Hence, by ignoring the temporal dependence on the reset, S^l in (3), the gradients of the weights in layer *l* can be derived as,

$$\frac{\partial L^l}{\partial W^l_{ij}} = \sum_k G^l_{ki}\left(Y^l_k - T^l_k\right) B_i[u] \sum_{\tau=1}^{t_{win}} S^{l-1}_j[\tau] \tag{6}$$

where T^l is the target vector containing the one-hot encoding of the corresponding label, $B_i[u]$ is a rectangular function with a width, α, taken as a surrogate derivative function to approximate the derivative of the firing function for the neuron *i*. We can define an error term propagated from the local classifier to the neuron layer as

$$\delta^l_i = \sum_k G^l_{ki}\left(Y^l_k - T^l_k\right) \tag{7}$$

The weight gradients can be written in the matrix form as

$$g^l_W = \frac{\partial L^l}{\partial W^l} = \delta^l \odot B^l[U^l] \otimes \sum_{\tau=1}^{t_{win}} S^{l-1}[\tau] \tag{8}$$

where $\odot$ denotes element-wise multiplication and $\otimes$ is outer product operator. The final weight updates are computed by a stochastic gradient descent (SGD) with momentum optimization algorithm [16] where the velocity vector V and weights updates are defined as

$$V[t+1] = \mu V[t] + g^l_W, \ \ and \tag{9}$$

$$W^l = W^l - \eta V[t+1] \tag{10}$$

respectively, where μ is the momentum and η is the learning rate.

III. Software Experiments and Results

In the simulation, multilayer networks with different sizes and depth were tested on MNIST dataset. Each input example was presented to the networks for 25 ms time window with the resolution of 0.5 ms. The batch size was chosen as 128 and simulations were run for 100 epochs. SGD with momentum was used as the optimizer. A step-decay scheduling method was used to reduce learning rate by a factor of 2 every 20 epochs. All the simulations were performed in Pytorch framework. As for hyperparameter setting, in the TTFS coding model, τ_{th} was chosen as 6. In the neuron models, θ was chosen as 0.2. In the local learning models, α, μ and η are given as 0.1, 0.6, and 0.8, respectively.

SNNs with different hidden layer sizes and depth were trained by the proposed local training algorithm. In the training algorithm, the local classifiers with random weights were used. The impact of the precision of the random weights was studied. We experimented with two precisions: 32-bit floating-point random number and binary random number (-1 or 1). The classification results are shown in Fig. 2, where MNIST-Bin indicates that binary random weights were used in the local classifiers and in other cases, 32-bit floating-point random weights were used.

In general, accuracy improves with the increasing network depth and hidden layer size. From the results, we can see that the local training algorithm can deliver good classification accuracy. Using binary random weights does not necessarily lead to accuracy degradation. Instead, it can improve the accuracy in most cases. Binary weights can largely reduce the memory size required to store the weights of local classifiers and simplify the computations for generating local predictions in (5) and errors in (7), thus lowering power consumption. For example, in (7), the multipliers that are necessary to compute the errors in the case of 32-bit floating point random weights can be replaced with multiplexers.

IV. Hardware Implementation

A. Architecture overview

Local training enables parallel forward and backward processing and a scalable architecture design. We propose a digital architecture with local training as illustrated in Fig. 3. The proposed design is composed of an input coding module and multiple neuron layers. Each neuron layer consists of a neuron module, a local classifier module, and a learning module. Multiple processing elements are implemented for accelerating training process. The scalability of the design is manifested in that the training of each layer is independent and extra layers can be easily cascaded using the same design.

1) Input coding: The input coding module receives inputs, such as frame pixels, and converts them into binary spikes based on the TTFS coding method described in Section II-A. The module produces a time-decaying threshold through the exponential function in (1) that is implemented as a memory table that stores pre-calculated values. The first-spike generation is realized by comparing the input and the threshold and imposing a large refractor period on the firing neurons.

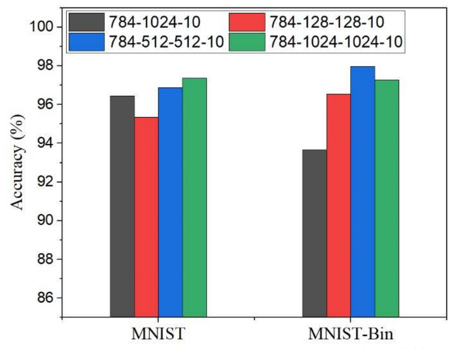

Fig. 2 Classification accuracy on MNIST dataset for different network sizes. MNIST-Bin indicates that SNNs were trained by local classifiers with binary random weights. In other cases, the weights of the local classifiers were 32-bit floating random number.

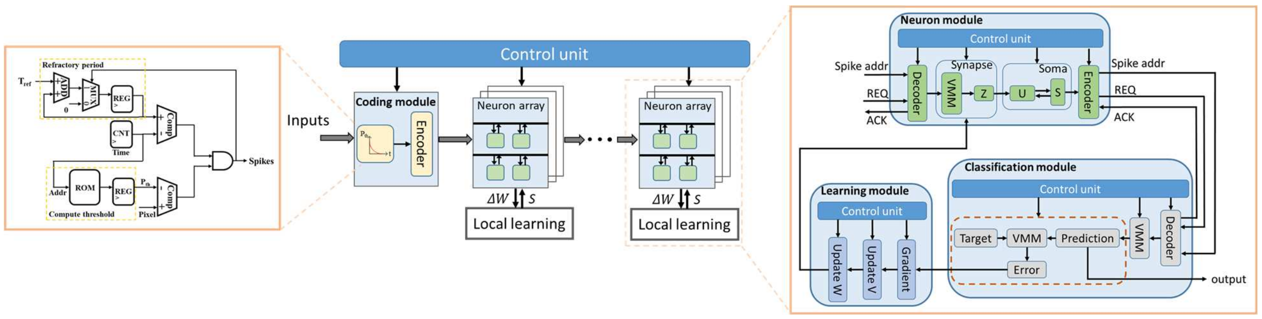

Fig. 3 Overview of the proposed digital architecture with local training.

2) Spike communication: Spikes are transmitted between modules (i.e., layers) through an event-address representation (AER) method that encodes binary spikes into corresponding neural addresses and controls the communication by REQ/ACK handshake signals. A pair of encoders and decoders are used for transmission.

3) Neuron module: Each module receives spike addresses and decodes the addresses into binary spikes. A vector-matrix multiplexing (VMM) engine uses multiplexing-add operations to sum up all the weights activated by the input spikes, i.e. computing W^lS^{l-1}, and update the synaptic input Z according to (2). Neuron membrane potentials are then updated, and output spikes are generated if the firing condition is met. No multiplications are needed due to the binary inputs.

4) Classification module: The local classifier makes predictions based on the output spikes from the neuron module through a VMM using random weights based on (5). We have demonstrated that binary random weights (-1 or 1) are effective in training. The weights are actually stored as 1 bit (0 or 1) on chip by ignoring the sign bit. So, during computation, 0 acts as -1. Local errors are computed by another VMM based on (7).

5) Learning module: The learning module implements the SGD with momentum optimization method. Gradients of the network parameters are computed based on (8) that contains element-wise vector-vector operations, and the element-wise multiplication is replaced with multiplexing because of the binary nature of the surrogate gradients, B. Additional memory is allocated to save the velocity vector V that is accumulated in the directions of the gradients. The parameters are then modified by the velocity vector with a learning rate.

6) Control: Control units are used to manage data flow and schedule element-level computations and architecture-level parallel processing. At each layer, training is executed while the spikes are forwarded and processed in the next layer.

B. Parallel design

Matrix operations are the most prominent operations when it comes to deep learning. The acceleration of deep learning algorithms relies heavily on the design of VMM engines. In the proposed local training method, VMM is used for three purposes: computing synaptic inputs, making predictions, and generating local errors. For computing synaptic input and making predictions, the design of VMMs replaces multipliers with multiplexers thanks to the binary spike inputs. We explored two degrees of parallelism in the design. Fig. 4 illustrates a design example for computing W^lS^{l-1} in layer l according to (2). The design of other VMMs follows the same principle. Each neuron layer is allocated with memory to store synaptic weights. In order to process the matrix operation in parallel, weight matrices can be partitioned into multiple blocks, for example, $m \times n$ in Fig. 4. Accordingly, the input vector is partitioned into m blocks. The results are saved into n memory blocks. Together with pipelining, the parallel design can accelerate the matrix operations proportionally by $m \times n$ times, but at the cost of more hardware resource. In this design, we chose $m = 4$ and $n = 4$ for acceleration.

C. Results and discussions

The proposed design was implemented on Xilinx Virtex-7 VC709 FPGA board. TABLE 1 presents the results of implementations with various network sizes and comparisons with two prior works using different training algorithms. Implementation results include hardware resources, training latency for one input example, and corresponding training power consumption. SNNs were run at the frequency of 100 MHz. The power consumption was estimated post routing using the power analysis tool in Vivado Design Suite that provides detailed analysis and accurate estimation [17]. Computations were based on fixed-point data representation with 10 integer and 20 fractional bits.

Clearly, hardware resource utilization grows considerably with the network size. The training latency increases only by around 0.4 ms thanks to the parallel execution of forward and backward updates in the local training method. Two prior works are included for comparisons. *Lee et al.* proposed a spike-train level direct feedback alignment (ST-DFA) training method to train SNNs on FPGA, in which errors are propagated backward from the last layer to all the upper layers simultaneously through random weights [18]. This algorithm can lead to 95.72% accuracy on MNIST in the network with 100 hidden

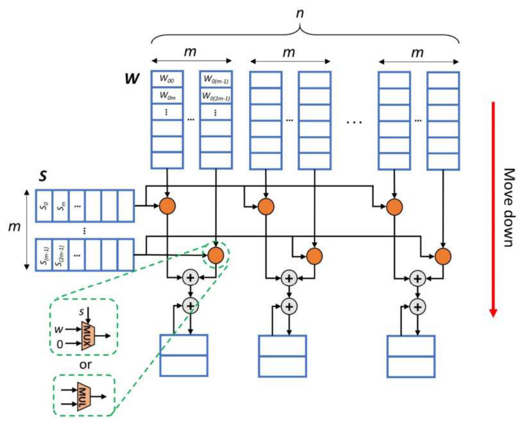

Fig. 4 VMM engine design. A design example of computing synaptic inputs based on input spikes S and synaptic weights W is presented.

neurons. Our training method achieves 93.62% accuracy in the same network. The DFA method has high complexity and is not hardware-friendly. In TABLE 1, the DFA implementation was only for cropped inputs with 196 input neurons, whereas the other methods were for the full-size inputs of MNIST images, 784. Even with a smaller input size, the method consumes 7.2× LUTs, 3.9×FFs, and 5.2× DSPs, leading to 1.3× training latency and 1.2 × power. *Guo et al.* implemented an unsupervised STDP for training SNNs on FPGA. Similarly, the STDP method is also a local training method, utilizing local spiking activity to modify synaptic weights [4]. Compared with our local training method and the DFA method, STDP achieves limited classification performance, e.g., 84.43% on MNIST. Due to its simplicity, this method consumes fewer resources and less power but with higher latency. To provide an overall comparison, we defined a figure of merit (FOM) as

$$FoM = Energy \times Latency \times \#\,slices \times AccuracyLoss$$

which considers energy, latency, hardware resource, and classification accuracy. Based on the FOM, our design shows 7.7× efficiency over the DFA method and 2.7× efficiency over the STDP method. To sum up, our digital system requires a small number of resources, leads to the fastest training speed, and achieves competitive classification accuracy.

TABLE 1
IMPLEMENTATION RESULTS OF SNNS WITH LOCAL TRAINING ON FPGA AND COMPARISON WITH PRIOR WORKS.

Training method	**Local training (our work)**			**ST-DFA [18]**	**STDP [4]**
Hidden layer(s)	128-128	128	100	100	100
Accuracy (%)	94.53	93.78	93.62	95.72	84.43
LUT	24526	9580	10163	73027	8761
FF	9536	3140	3159	12329	5607
BRAM	487.5	217	216.5	N.A.	37.5
DSP	37	21	21	110	28
Latency (ms)	4.83	4.43	3.68	4.80	16.17
Power (mW)	253.58	192.03	188.02	224.00	15.86
FoM	2.51	0.71	0.52	4.03	1.41

V. Conclusions and Future work

This work proposed an effective and hardware-friendly local training algorithm for SNNs, which is well-suited for neuromorphic hardware design. The proposed training algorithm receives input spikes converted from TTFS neural coding scheme and trains each layer of SNNs separately. Binary random weights in the local classifiers were demonstrated to be effective in training without accuracy loss, which simplifies the algorithm for low-cost hardware implementation. The proposed digital hardware architecture explores the parallelism in matrix operations and performs parallel execution of forward and backward updates. It outperforms prior works in terms of resource utilization and training speed.

However, several challenges need to be addressed for more practical applications. The proposed hardware implementation is not optimized, especially in the parallel design. High parallelism would result in inefficient use of on-chip memory, high resource utilization, high power consumption, and limited speedup. Thus, the design space regarding the two degrees of parallelism, m and n, can be further explored for optimization under a particular design metric, such as energy or energy-delay product. We will extend the current work into a large-scale implementation, such as deep convolutional networks, based on the proposed local training algorithm by incorporating fine-grained parallel processing design. Additionally, integer precision training will be considered as an efficient way to further improve the energy efficiency of the training without sacrificing the accuracy [19].

Acknowledgment

This work was funded by the King Abdullah University of Science and Technology (KAUST) AI Initiative, Saudi Arabia.

References

[1] T. Tang, R. Luo, B. Li, H. Li, Y. Wang, and H. Yang, "Energy efficient spiking neural network design with RRAM devices," in *International Symposium on Integrated Circuits (ISIC)*, 10-12 December 2014, pp. 268-271.

[2] B. Han, A. Sengupta, and K. Roy, "On the energy benefits of spiking deep neural networks: A case study," in *International Joint Conference on Neural Networks (IJCNN)*, 24-29 July 2016, pp. 971-976.

[3] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks,* vol. 99, pp. 56-67, 2018.

[4] W. Guo, H. E. Yantir, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Toward the Optimal Design and FPGA Implementation of Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* pp. 1-15, 2021.

[5] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-Efficient Neuron, Synapse and STDP Integrated Circuits," *IEEE Transactions on Biomedical Circuits and Systems,* vol. 6, no. 3, pp. 246-256, 2012.

[6] J. Park and S. Jung, "Presynaptic Spike-Driven Spike Timing-Dependent Plasticity With Address Event Representation for Large-Scale Neuromorphic Systems," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 67, no. 6, pp. 1936-1947, 2020.

[7] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-Neuron 1M-Synapse 3.8-pJ/SOP Spiking Neural Network With On-Chip STDP Learning and Sparse Weights in 10-nm FinFET CMOS," *IEEE Journal of Solid-State Circuits,* vol. 54, no. 4, pp. 992-1002, 2019.

[8] H. Mostafa, "Supervised Learning Based on Temporal Coding in Spiking Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 29, no. 7, pp. 3227-3235, 2018.

[9] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going Deeper With Directly-Trained Larger Spiking Neural Networks," in *AAAI Conference on Artifial Intelligence*, 2-4 February 2021,

[10] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep Supervised Learning Using Local Errors," *Frontiers in Neuroscience,* vol. 12, no. 608, 2018.

[11] M. Payvand, M. E. Fouda, F. Kurdahi, A. M. Eltawil, and E. O. Neftci, "On-Chip Error-Triggered Learning of Multi-Layer Memristive Spiking Neural Networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 10, no. 4, pp. 522-535, 2020.

[12] E. S. Marquez, J. S. Hare, and M. Niranjan, "Deep Cascade Learning," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 29, no. 11, pp. 5475-5485, 2018.

[13] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)," *Frontiers in Neuroscience,* vol. 14, no. 424, 2020.

[14] C. Ma, J. Xu, and Q. Yu, "Temporal Dependent Local Learning for Deep Spiking Neural Networks," in *International Joint Conference on Neural Networks (IJCNN)*, 18-22 July 2021, pp. 1-7.

[15] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems," *Frontiers in Neuroscience,* vol. 15, no. 212, 2021.

[16] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," presented at the Proceedings of the 30th International Conference on International Conference on Machine Learning, Atlanta, GA, USA, 2013.

[17] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis," *IEEE Access,* vol. 5, pp. 2747-2762, 2017.

[18] J. Lee, R. Zhang, W. Zhang, Y. Liu, and P. Li, "Spike-Train Level Direct Feedback Alignment: Sidestepping Backpropagation for On-Chip Training of Spiking Neural Nets," *Frontiers in Neuroscience,* vol. 14, no. 143, 2020.

[19] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks,* vol. 125, pp. 70-82, 2020.