Learning and Real-time Classification of Hand-written Digits With Spiking Neural Networks

Shruti R. Kulkarni, John M. Alexiades, Bipin Rajendran
Department of Electrical and Computer Engineering,
New Jersey Institute of Technology, Newark, NJ, 07102, USA
Email: {srk68, jma59, bipin}@njit.edu

***Abstract*—We describe a novel spiking neural network (SNN) for automated, real-time handwritten digit classification and its implementation on a GP-GPU platform. Information processing within the network, from feature extraction to classification is implemented by mimicking the basic aspects of neuronal spike initiation and propagation in the brain. The feature extraction layer of the SNN uses fixed synaptic weight maps to extract the key features of the image and the classifier layer uses the recently developed NormAD approximate gradient descent based supervised learning algorithm for spiking neural networks for training. On the standard MNIST database images of handwritten digits, our network achieves an accuracy of 99.80% on the training set and 98.06% on the test set, with nearly $7\times$ fewer parameters compared to the state-of-the-art spiking networks. We further use this network in a GPU based user-interface system demonstrating real-time SNN simulation to infer digits written by different users. On a test set of 500 such images, this real-time platform achieves an accuracy exceeding 97% while making a prediction within an SNN emulation time of less than $100\,\text{ms}$.**

***Index Terms*—Spiking neural networks, classification, supervised learning, GPU based acceleration, real-time processing**

I. Introduction

The human brain is a computational marvel compared to man-made systems, both in its ability to learn to execute highly complex cognitive tasks, as well as in its energy efficiency. The computational efficiency of the brain stems from its use of sparsely issued binary signals or spikes to encode and process information. Inspired by this, spiking neural networks (SNNs) have been proposed as a computational framework for learning and inference [1]. General purpose graphical processing units (GP-GPUs) have become an ideal platform for accelerated implementation of large scale machine learning algorithms [2]. There have been multiple GPU based implementations for simulating large SNNs [3]–[8], with most of these targeting the forward communication of spikes through large networks of spiking neurons and/or local weight update based on spike timing difference. In contrast, we demonstrate a highly optimized implementation scheme for parallel global weight update and spike based supervised learning on GPU platforms and use the framework for real time inference on digits captured through a touch-screen interface by users.

Previous efforts to develop deep convolutional spiking networks started by using second generation artificial neural networks (ANNs) with back-propagation of errors to train the network and thereafter converting it into spiking versions [9]–[12]. There have been several supervised learning algorithms proposed to train the SNNs, by explicitly using the time of spikes of neurons to encode information, and to derive the appropriate weight update rules to minimize the distance between desired spike times and observed spike times in a network [13]–[17]. We use the Normalized Approximate Descent (NormAD) algorithm to design a system to identify handwritten digits. The NormAD algorithm has shown superior convergence speed compared to other methods such as the Remote Supervised Method (ReSuMe) [13].

Our SNN is trained on the MNIST database consisting of $60,000$ training images and $10,000$ test images [18]. The highest accuracy SNN for the MNIST was reported in [16], where a two-stage convolution neural network achieved an accuracy of 99.31% on the test set. Our network, in contrast, has just three layers, with about $82,000$ learning synapses ($7\times$ fewer parameters compared to [16]) and achieves an accuracy of 98.06% on the MNIST test dataset.

The paper is organized as follows. The computational units of the SNN and the network architecture is described in section II. Section III details how the network simulation is divided among different CUDA kernels. The user-interface system and the image pre-processing steps are explained in Section IV. We present the results of our network simulation and speed related optimizations in Section V. Section VI concludes our GPU based system implementation study.

II. Spiking Neural Network

The basic units of an SNN are spiking neurons and synapses interconnecting them. For computational tractability, we use the leaky integrate and fire (LIF) model of spiking neurons, where the evolution of the membrane potential, $V_m(t)$ is described by:

$$C\frac{dV_m(t)}{dt} = -g_L(V_m(t) - E_L) + I(t) \tag{1}$$

Here $I(t)$ is the total input current, E_L is the resting potential, and C ($300\,\text{pF}$) and g_L ($30\,\text{nS}$) model the membrane capacitance and leak conductance, respectively [13]. Once the membrane potential crosses a threshold ($V_m(t) \geq V_T$), it is reset to its resting value E_L and remains at that value till the neuron comes out of its refractory period ($t_{ref} = 3\,\text{ms}$).

The synapse, with weight $w_{k,l}$ connecting input neuron k to output neuron l, transforms the incoming spikes (arriving

This research was supported in part by grants from National Science Foundation Award 1710009, CISCO, and the McNair Fellowship Program.

at times $t_k^1, t_k^2, \ldots$) into a post-synaptic current ($I_{k,l}$), based on the following transformation,

$$c_k(t) = \sum_i \delta(t - t_k^i) * \left(e^{-t/\tau_1} - e^{-t/\tau_2}\right) \tag{2}$$

$$I_{k,l}(t) = w_{k,l} \times c_k(t) \tag{3}$$

Here, the summed δ function represents the incoming spike train and the double decaying exponentials with $\tau_1 = 5\,\text{ms}$ and $\tau_2 = 1.25\,\text{ms}$ represent the synaptic kernel.

A. Network architecture & spike encoding

We use a three-layered network where hidden layer performs feature extraction and the output layer performs classification (see Fig. 1). The network is designed to take input from 28×28 pixel MNIST digit image. We translate this pixel value into a set of spike streams, by passing the pixels as currents to a layer of 28×28 neurons (first layer). The current $i(k)$ applied to a neuron corresponding to pixel value k, in the range $[0, 255]$ is obtained by the following linear relation:

$$i(k) = I_0 + k \times I_p \tag{4}$$

where $I_p = 101.2\,\text{pA}$ is a scaling factor, and $I_0 = 2700\,\text{pA}$ is the minimum current above which an LIF neuron can generate a spike (for the parameters chosen in equation 1). These spike streams are then weighted with twelve 3×3 synaptic weight maps (or filters) to generate equivalent current streams using equations 2 and 3. These 12 spatial filter maps are chosen to detect various edges and corners in the image. Thus, these weight kernels and receptive fields implement spatial convolution (with a stride of one pixel) akin to convolutional neural networks, albeit with *a priori* chosen fixed weights.

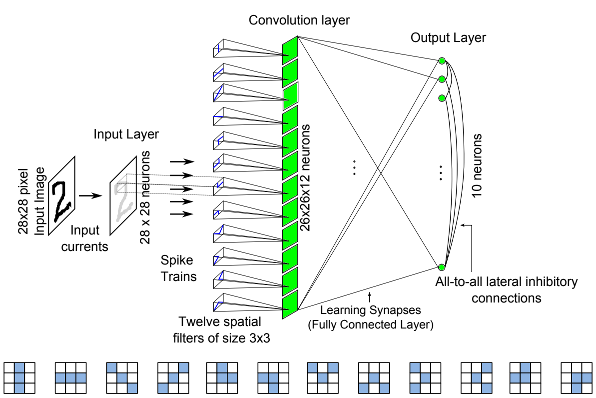

Fig. 1: The 28×28 pixel images from the MNIST database are converted to spike trains of duration T, weighted with twelve 3×3 synaptic weight maps (below) resulting in twelve 26×26 current streams and then feed to the corresponding feature map neurons. There are 10 output neurons corresponding to each digit. The weights of the fully-connected feed-forward synapses to the output layer neurons (8112×10) are adjusted using the NormAD learning rule [13].

The output layer consists of 10 neurons, one for each of the ten digits. We train the network so that the correct neuron in the output layer generates a spike train with a frequency close to 285 Hz and the other output neurons issue no spikes during the training interval, T (set to 100 ms in baseline experiments). This layer also has lateral inhibitory connections that helps to prevent the non-label neurons from spiking for a given input. The output neuron with the highest number of spikes is declared the winner of the classification.

B. Learning layer

The 8112×10 synapses connecting the hidden layer neurons to the 10 output layer neurons are modified during the course of training. The NormAD [13] supervised learning algorithm is used to calculate the weight update:

$$\Delta \mathbf{w} = r \int_0^T e(t) \frac{\hat{\mathbf{d}}(t)}{\|\hat{\mathbf{d}}(t)\|} dt, \qquad \hat{\mathbf{d}}(t) = \mathbf{c}(t) * \hat{h}(t) \tag{5}$$

where $\hat{h}(t) = (1/C)\exp(-t/\tau_L)$ with $\tau_L = 1\,\text{ms}$, and r is the learning rate [13]. The magnitude of weight update depends on the normalized value of $\hat{\mathbf{d}}(t)$, and is evaluated only when there is a difference between the desired and output spikes (i.e., $e(t) = S^d(t) - S^o(t)$ is non-zero).

III. CUDA IMPLEMENTATION

We realize our SNN on a GPU platform using the CUDA-C programming framework. A GPU is divided into streaming multiprocessors (SM), each of which consists of stream processors (SP) that are optimized to execute math operations. The CUDA-C programming framework exploits the hardware parallelism of GPUs and launches jobs on the GPU in a grid of blocks each mapped to an SM. The blocks are further divided into multiple threads, each of which is scheduled to run on an SP, also called a CUDA core. Since memory transfer between CPU and GPU local memory is one of the main bottlenecks, all network variables (i.e., neuron membrane potentials and synaptic currents) are declared in the global GPU memory in our implementation. The simulation equations (1,2,3) are solved numerically in an iterative manner at each time step.

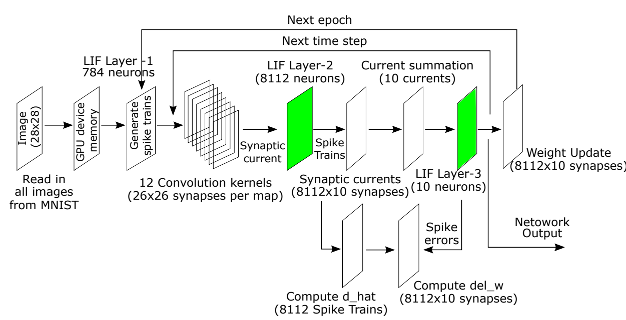

Fig. 2: Diagram showing the different variables of the network being computed each time step and how the signals flow across different layers. The dimensions within the brackets are the sizes of those variables and their respective CUDA kernels.

Fig. 2 shows the forward pass and backward pass for weight update during the training phase. Image pixels read into the GPU memory are passed as currents to layer one neurons (grid size of 28×28) for a period T. The filtering process involves 2D convolution of the incoming spike kernels and the weight matrix (3×3). The computation is parallelized across 12 CUDA kernels, each with a grid size of 26×26 threads. Each thread computes the current to the hidden layer neurons, indexed as a 2D-array i, j, $\{0 \leq i, j, \leq 25\}$ at a time-step n, based on the following spatial convolution relation:

$$I_{in}(i,j,n) = \sum_{a=0}^{2}\sum_{b=0}^{2} w_{conv}(a,b) \times c(i+a, j+b, n) \quad (6)$$

where c represents the synaptic kernel (as in equation 2) calculated from the spike trains of the 28×28 pixels and $w_{conv}(a,b)$ represents each of the weights from the 3×3 filter matrix.

The membrane potential of an array of k LIF neurons, for applied current $\mathbf{I}(n)$ (as described in equation 1) is evaluated using the second order Runge-Kutta method as:

$$\mathbf{k_1} = [-g_L(\mathbf{V_m}(n) - E_L) + \mathbf{I}(n)]/C \quad (7)$$

$$\mathbf{k_2} = [-g_L(\mathbf{V_m}(n) + \mathbf{k_1}\Delta t - E_L) + \mathbf{I}(n)]/C \quad (8)$$

$$\mathbf{V_m}(n+1) = \mathbf{V_m}(n) + [(\mathbf{k_1} + \mathbf{k_2})\Delta t/2] \quad (9)$$

Each thread k independently checks if the membrane potential has exceeded the threshold to artificially reset it.

$$\text{If } V_m^k(n+1) \geq V_T \Rightarrow V_m^k(n+1) = E_L \quad (10)$$

Refractory period is implemented by storing the latest spike issue time, n_k^{last} of each neuron in a vector $\mathbf{R}$; the membrane potential of a neuron is updated only when the current time step $n > n_k^{last} + (t_{ref}/\Delta t)$.

The synaptic current from neuron k in hidden layer to neuron l in output layer as given in equation 3 can be re-written to be evaluated in an iterative manner, thereby avoiding the evaluation of expensive exponential of the difference between the current time n and previous spike times n_k^i. The synaptic current computation, at time step n, for each of the (k,l) synapse is spawned in CUDA across 8112×10 kernels as:

$$a_k(n) = a_k(n-1) \times \exp(-\Delta t/\tau_1) + \delta(n - n_k^i) \quad (11)$$

$$b_k(n) = b_k(n-1) \times \exp(-\Delta t/\tau_2) + \delta(n - n_k^i) \quad (12)$$

$$c_k(n) = a_k(n) - b_k(n) \quad (13)$$

$$I_{k,l} = w_{k,l} \times c_k(n) \quad (14)$$

where $a_k(n)$ and $b_k(n)$ represent the rising and falling regions of the double exponential synaptic kernel. The strength of the synapses between the hidden and output layers is initialized to zero during training. At every time step, we calculate the error function for each output neuron, based on the difference between the observed and desired spikes. Next, $\hat{d}_k$ (see equation 5) for the spikes originating from neuron k is computed as:

$$\hat{d}_k(n) = \hat{d}_k(n-1)e^{-\Delta t/\tau_L} + (c_k(n)\Delta t)/C \quad (15)$$

Once $\hat{d}_k(n)$ is evaluated, we compute its norm across all k neurons and determine the instantaneous $\Delta w_{k,l}(n)$ for all the $81,120$ synapses in parallel, if there is a spike error. At the end of presentation, the accumulated $\Delta w_{k,l}$ is used to update the synaptic weights in parallel. The evaluation of the total synaptic current and the norm is performed using parallel reduction in CUDA [19]. During the inference or testing phase, we calculate the synaptic currents and membrane potentials of neurons in both layers to determine spike times, but do not evaluate the $\hat{\mathbf{d}}$ term and the weight update.

IV. Real-time inference on user data

We used the CUDA based SNN described in the previous section, to design a user interface that can capture and identify the images of digits written by users in real-time from a touch-screen interface. The drawing application to capture the digit drawn by the user is built using OpenCV, an image processing library. The captured image from the touch screen is pre-processed using standard methods similar to that used to generate the MNIST dataset images [18]. We use OpenCV to convert user drawn images to the required format which is a grayscale image of size 28×28 pixels. The network is implemented on the NVIDIA GTX 860M GPU which has 640 CUDA cores. The preprocessing phase takes about 25 ms, and this image is then passed to the trained SNN. The CUDA process takes about 300 ms to initialize the network in the GPU memory, after which the network simulation time depends on the presentation time T and the time step interval Δt.

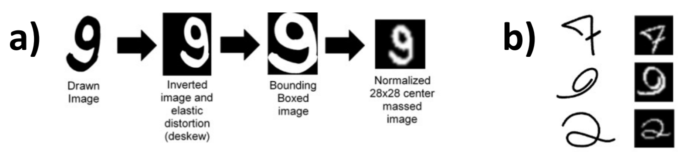

Fig. 3: (a) Outline of the preprocessing steps used to convert the user input to a 28×28 image that is fed to the network, (b) Examples of user input (left) and the pre-processed 28x28 pixel images fed to the SNN (right).

A. Image Preprocessing

Fig. 3(a) shows the simple preprocessing steps used to create the input signal to the SNN from the captured image and Fig. 3(b) shows some sample pre-processed images. We first apply an elastic distortion to vertically align the digit, and then resize it maintaining its aspect ratio. The digit is placed by its center of mass, and a border is applied to ensure that the final image is of size 28×28 pixels.

V. Results

We trained the network on the MNIST training data-set consisting of $60,000$ images, for 20 epochs. Our network achieves an error of 0.2% on the training set and and 1.94% on the test set. This is with a time step interval of $\Delta t = 0.1$ ms when the network is simulated for $T = 100$ ms. Table I lists the state-of-the-art networks (ANN and SNN) for the MNIST classification problem. It can be seen that though these networks have classification accuracies exceeding 99%, they use more than $7\times$ the number of parameters compared to our network, which is designed to simplify the computational load in developing real-time system.

If the integration time step interval used during inference is 1 ms instead of 0.1 ms, the MNIST test error increases only

TABLE I: Comparison of our SNN with state-of-the-art

Network and learning algorithm	Learning synapses	Accuracy
Deep Learning [20]	$2,508,470$	99.79%
ANN converted to SNN [9]	$1,422,848$	99.12%
4-layer convolution SNN [16]	$581,520$	99.31%
SNN, with NormAD (this work)	$81,120$	98.06%

by about 0.4%, but there is a $10\times$ reduction in the processing time. For instance, when each digit is presented for a period of $T = 75$ ms, the network can be simulated in an average wall clock time of 65 ms (compared to 650 ms with $\Delta t = 0.1$ ms), making real-time processing possible (see Fig. 4(a)). We tested

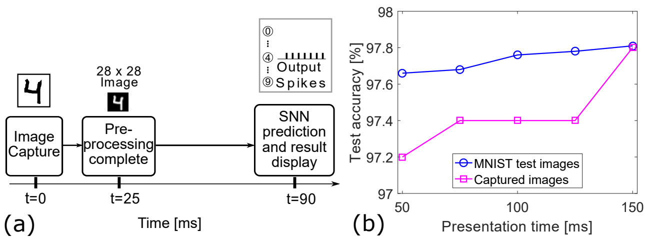

Fig. 4: (a) Various stages of classifying a user's input: the image pre-processing takes 25 ms and the 75 ms SNN emulation is completed in real-time. (b) Accuracy results as a function of presentation time, with the integration time steps of $\Delta t = 1$ ms. The accuracy trend for the images captured from the touch-screen interface is comparable to the MNIST test data-set.

the network's accuracy, with $\Delta t = 1$ ms configuration on a sample set of 500 handwritten digits collected from various users through our user-interface system. Fig. 4(b) shows the accuracy variation as a function of presentation time. At $T = 75$ ms, we measure an accuracy of 97.4% on our set of 500 captured images, while on the MNIST test-set it was 97.68%. The slight loss in performance compared to the MNIST dataset is attributed to the deviations from the statistical characteristics of the captured images through our interface with respect to the MNIST dataset.

VI. Conclusion

We have made two novel contributions in this work. First, we developed a simple three-layer spiking neural network, that performs spike transformation, feature extraction and classification. All information processing and learning within the network is performed entirely in the spike domain. With approximately 7 times lesser number of synaptic weight parameters compared to the state of the art spiking networks, we show that our approach achieves classification accuracy exceeding 99% on the training set of the MNIST database and 98.06% on its test set. The trained network implemented on the CUDA parallel computing platform is also able to successfully identify digits written by users in real-time, demonstrating its true generalization capability.

We have also demonstrated a general framework for implementing spike based neural networks and supervised learning with non-local weight update rules on a GPU platform. At each time step, the neuronal spike transmission, synaptic current computation and weight update calculation for the network are all executed in parallel in this framework. Using this GPU implementation, we demonstrated a touch-screen based platform for real-time classification of user-generated images.

Acknowledgment

This research was supported in part by the CAMPUSENSE project grant from CISCO Systems Inc, the Semiconductor Research Corporation, McNair Scholars program and the National Science Foundation grant 1710009.

References

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] A. Coates *et al.*, "Deep learning with COTS HPC systems," in *30th International Conference on Machine Learning*, 2013.

[3] A. K. Fidjeland and M. P. Shanahan, "Accelerated simulation of spiking neural networks using GPUs," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[4] J. M. Nageswaran *et al.*, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural networks*, vol. 22, no. 5, pp. 791–800, 2009.

[5] E. Yavuz, J. Turner, and T. Nowotny, "GeNN: a code generation framework for accelerated brain simulations," *Nature Scientific Reports*, vol. 6, no. 18854, 2016.

[6] D. Yudanov *et al.*, "GPU-based simulation of spiking neural networks with real-time performance and high accuracy," in *International Joint Conference on Neural Networks*, July 2010.

[7] F. Naveros *et al.*, "Event-and time-driven techniques using parallel CPU-GPU co-processing for Spiking Neural Networks," *Front. in Neuroinformatics*, vol. 11, 2017.

[8] J. L. Krichmar, P. Coussy, and N. Dutt, "Large-scale spiking neural networks using neuromorphic hardware compatible models," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 4, p. 36, 2015.

[9] P. U. Diehl *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[10] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[11] B. Rueckauer *et al.*, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *arXiv preprint arXiv:1612.04052*, 2016.

[12] E. Hunsberger and C. Eliasmith, "Training Spiking Deep Networks for Neuromorphic Hardware," *arXiv preprint arXiv:1611.05141*, 2016.

[13] N. Anwani and B. Rajendran, "NormAD -Normalized Approximate Descent based supervised learning rule for spiking neurons," in *International Joint Conference on Neural Networks*, July 2015, pp. 1–8.

[14] F. Ponulak and A. Kasinski, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[15] A. Mohemmed *et al.*, "SPAN: Spike Pattern Association Neuron for Learning Spatio-Temporal Spike Patterns," *International Journal of Neural Systems*, vol. 22, no. 04, 2012.

[16] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks using Backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.

[17] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "CONE: Convex-Optimized-Synaptic Efficacies for Temporally Precise Spike Mapping," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 849–861, April 2017.

[18] "The MNIST database of handwritten digits," available at http://yann.lecun.com/exdb/mnist/.

[19] M. Harris, "Optimizing parallel reduction in CUDA," available at http://bit.ly/2gd7fSb.

[20] L. Wan *et al.*, "Regularization of Neural Networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 1058–1066.