Fast Classification Using Sparsely Active Spiking Networks

Hesham Mostafa[1], Bruno U. Pedroni[2], Sadique Sheik[3], and Gert Cauwenberghs[1,2,3]
[1]Institute for Neural Computation, [2]Department of Bioengineering, [3]BioCircuits Institute
UC San Diego, California, USA
Email: {hmmostafa,bpedroni,ssheik,gert}@ucsd.edu

***Abstract*—Spike generation and routing is typically the most energy-demanding operation in neuromorphic hardware built using spiking neurons. Spiking neural networks running on neuromorphic hardware, however, often use rate-coding where the neurons spike rate is treated as the information-carrying quantity. Rate-coding is a highly inefficient coding scheme with minimal information content in each spike, which requires the transmission of a large number of spikes. In this paper, we describe an alternative type of spiking networks based on temporal coding where neuron spiking activity is very sparse and information is encoded in the time of each spike. We implemented the proposed networks on an FPGA platform and we use these sparsely active spiking networks to classify MNIST digits. The network FPGA implementation produces the classification output using only few tens of spikes from the hidden layer, and the classification result is obtained very quickly, typically within 1-3 synaptic time constants. We describe the idealized network dynamics and how these dynamics are adapted to allow an efficient implementation on digital hardware. Our results illustrate the importance of making use of the temporal dynamics in spiking networks in order to maximize the information content of each spike, which ultimately leads to reduced spike counts, improved energy efficiency, and faster response times.**

I. Introduction

Artificial neural networks (ANNs) using analog-valued neurons have emerged as an extremely powerful paradigm for solving a myriad of learning problems [1]. ANNs are typically trained and deployed on standard graphical processing units (GPUs). However, there is growing interest in custom low-power devices that can run pre-trained neural networks in real-time in order to extract high-level semantic information form unstructured natural input coming from the environment. The field of neuromorphic engineering has a long tradition of implementing spiking neural network (SNN) architectures in custom low-power devices [2]–[4]. This has motivated several approaches for mapping ANNs onto SNNs in order to leverage the existing low-power spiking neuromorphic hardware for tackling the problems at which ANNs have been so successful. Unlike an ANN neuron, the output of a spiking neuron is a stream of all-or-nothing events or spikes. To map an ANN onto an SNN, a rate-based representation is typically used [5]–[8] where the firing rate of a spiking neuron represents the analog output of the corresponding ANN neuron.

Spiking rate-based representations are highly inefficient in terms of hardware implementation, since they result in a large number of weight lookups and a large volume of spike traffic in the routing fabric. That is because enough spikes have to be generated so that the spike counts are a good representation of the neurons' mean firing rates. In a conventional ANN, a neuron's output is multiplied by the weight vector and the result is used to update the target neurons. In a rate-based SNN, however, the weight vector has to be looked up multiple times, one for each spike. These multiple weight lookups for the same weight vector can easily be the most energy-demanding operation in the SNN implementation together with the energy needed to route each spike to its destination.

In this paper, we describe an alternative SNN architecture in which information is represented in spike times, not spike rates. It was previously shown that SNNs that use a temporal-coding scheme can be effectively trained using backpropagation [9]. We use a more recent formulation that does not make use of explicit delay elements, and which allows the derivation of a closed form differentiable relation between spike times and between spike times and network weights [10]. In the temporal-coding representation, the output of each neuron is taken as the time of its first output spike. The resulting spiking networks have very sparse activity which makes them ideal for low-power hardware implementation. We describe how these spiking networks can be efficiently implemented on digital hardware and present results from an FPGA implementation of a multi-layer feedforward SNN that carries out a classification of MNIST digits. We show that the network reaches a classification decision very quickly before the majority of the neurons have spiked, which reduces the number of needed weight lookups. In section II, we review the SNN network model and the training approach based on temporal coding previously presented in ref. [10]. We describe the adaptation of the network model to a digital FPGA-based implementation in section III. We present results from the the FPGA implementation in section IV before concluding in section V.

II. Network Model

In this section, we review the network model and the training technique for temporal networks described in ref. [10]. The spiking neurons are non-leaky integrate and fire neurons with membrane potential dynamics given by:

$$\frac{dV^j_{mem}(t)}{dt} = I^j_{syn}(t) \tag{1}$$

where V^j_{mem} and I^j_{syn} are the membrane potential and synaptic currents of neuron j respectively. Neuron j spikes when V^j_{mem} crosses the firing threshold which we arbitrarily set to 1. The synaptic current is given by:

$$\frac{dI^j_{syn}(t)}{dt} = -I^j_{syn}(t) + \sum_i \sum_r w_{ji}\delta(t - t^r_i) \tag{2}$$

where w_{ji} is the synaptic weight from neuron i to neuron j and t^r_i is the time of the r^{th} spike from neuron i. The summation over i runs over all pre-synaptic neurons. The summation over r runs over all the spikes from neuron i. $\delta(x)$ is the Dirac delta function. Thus, incoming spikes trigger a jump in the synaptic current with a magnitude equal to the synaptic weight. The synaptic current otherwise decays exponentially. We set the decay time constant to 1 as it is the only time constant in the network and can be arbitrarily chosen.

Assume a neuron receives N spikes at times $\{t_1,..,t_N\}$ with weights $\{w_1,..,w_N\}$ from N source neurons. Assume the neuron spikes for the first time at time t_{out}. By integrating Eqs. 1 and 2, we can explicitly write the membrane potential at time $t < t_{out}$:

$$V_{mem}(t) = \sum_{i=1}^{N} \Theta(t - t_i) w_i (1 - exp(-(t - t_i))) \tag{3}$$

$$\Theta(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Assume only a subset of these input spikes with indices in $C \subseteq \{1,..,N\}$ had arrived before t_{out} where $C = \{i : t_i < t_{out}\}$. We call this set of input spikes the causal set of input spikes. From Eq. 3, t_{out} is then implicitly defined as:

$$1 = \sum_{i \in C} w_i (1 - exp(-(t_{out} - t_i))) \tag{4}$$

where 1 is the firing threshold. Hence,

$$exp(t_{out}) = \frac{\sum_{i \in C} w_i exp(t_i)}{\sum_{i \in C} w_i - 1} \tag{5}$$

Equation 5 differentiably relates the time of the first output spike, t_{out}, to the times of all spikes in the causal set of input spikes and their synaptic weights.

To use these networks in a classification setting, we construct a multi-layer neural network where the top layer has as many neurons as the number of classes. For MNIST classification where there are 10 classes, we have 10 output neurons in the top layer. The input is used to drive the bottom layer. The goal is to train the network so that the neuron corresponding to the correct class fires first among the top layer neurons. We impose a differentiable cost function on the spike times of the top-layer neurons which is minimized when the neuron corresponding to the correct class fires early and fires first among the top layer neurons. We then use gradient descent to adjust the weights in the network so as to minimize the cost function. In the networks we present in this paper, we only allow each neuron to spike once for each input presentation. We adopt this strategy to encourage sparse spiking activity. For more details about the training procedure, we refer the reader to ref. [10].

III. Description of FPGA implementation

For a digital implementation, the continuous dynamics in Eqs. 1 and 2 have to be discretized in time and value. The digital implementation will simulate the differential equations 1 and 2 in a time-stepped manner:

$$V^j_{mem}(t + \Delta t) = V^j_{mem}(t) + I^j_{syn}(t)\Delta t \tag{6}$$

$$I^j_{syn}(t + \Delta t) = I^j_{syn}(t) + \sum_i \sum_r w_{ji}\delta(t - t^r_i) - I^j_{syn}(t)\Delta t \tag{7}$$

The time-stepped updates of the membrane potential and the synaptic current involve only additions and subtractions except for the terms which have multiplications by Δt. These multiplications can be made efficiently implementable in digital hardware if Δt has the form 2^{-k} for some positive integer k. The multiplication can then be cheaply implemented as a k-bit right bit-shift. In our digital implementation, we choose $k = 7$, i.e, $\Delta t = 2^{-7}$. Since the synaptic time constant was chosen to be 1, it takes 128 time steps to simulate the network for one synaptic time constant. Choosing k too small would compromise the temporal resolution of the network since the time step would be large, while choosing k too large would cause the rounding errors when dividing by 2^k (the k-bit right bit-shift) to become too high.

We use 16 bits in signed Q2.13 fixed point format to represent the V_{mem} and I_{syn} of each neuron. The weights are signed with 8-bit resolution. The trained real-valued weights are almost all in the range of $[-0.5, 0.5]$. These weights were scaled by 2^8, rounded and clipped so that the range $[-0.5, 0.5]$ maps to the integer range $[-127, 127]$. Since the 16-bit fixed-point representation of I_{syn} has a 13-bit fractional part, i.e, it is the real-valued I_{syn} from the ideal network scaled up by 2^{13}, the integer weights have to be scaled up by $2^{13-8} = 32$ before adding them to I_{syn}. This is cheaply implemented by sign-extending the 8-bit weight to 16 bits, and shifting the sign-extended weight left by 5 bits before adding it to I_{syn}.

We implemented the digital version of the proposed spiking networks on a Xilinx Spartan6-LX150 FPGA. Figure 1 shows a schematic of the FPGA architecture. The main compute blocks are the neuron cores. In our current implementation, there are 8 neurons cores. Each neuron core contains a 256x36 bits block ram element that stores the current state (I_{syn} and V_{mem}) of 256 neurons, together with 4 bits of control information per neuron. One of the control bits is used to indicate whether a neuron has spiked or not in the current simulation run. If a neuron has spiked once, it is not allowed to spike again. In each time step, each core proceeds sequentially through the state memory to update the states of the 256 neurons according to Eqs. 6 and 7. If a neuron's membrane potential goes above threshold, the core stores the 8-bit core neuron address in its spike FIFO. If the FIFO is full, the core halts until the FIFO has an empty place. The memory manager goes through the 8 spike FIFOs in a round-robin manner. If a FIFO is non-empty, the memory manager reads the core address of the spiking neuron from the FIFO, appends a core id tag to obtain the global neuron address and uses this address to fetch the synaptic weights and global addresses of the target neurons from an external DDR2 memory. The memory manager then sequentially broadcasts each fetched weight and target neuron address to the cores.

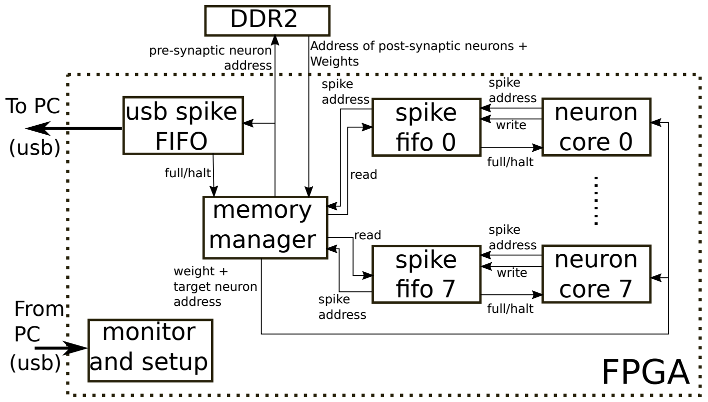


Fig. 1: Architecture of the FPGA implementation of spiking networks. For clarity, only two of the eight neuron cores are shown. The eight neuron cores can be used to implement up to $256 * 8 = 2048$ neurons. See main text for description of the architecture.

If an address matches the core's address range, the core halts and uses the broadcasted weight to update I_{syn} of the target neuron.

The address used to fetch data from the external DDR2 memory implicitly encodes the address of the spiking neuron. This address is stored in a global FIFO (the usb spike FIFO) together with the index of the current time step. The USB spike FIFO is read through USB by the host PC. If this FIFO is full, the memory manager halts to allow the host time to read the addresses of the spiking neurons. A time step is complete once all the cores have finished going through their internal memory once to update the states of their neurons, and when spike FIFO 0 to spike FIFO 7 are empty and there are no pending transactions with the DDR2. Once a time step is complete, a new time step is triggered by activating the cores so that they go through their internal state memory again to update the neuron states and emit spikes for neurons that crossed the threshold.

The monitor and setup block is used to set up the initial states of V_{mem} and I_{syn} of each neuron, and to set up the timestep at which the simulation should end. It can also be used to query the state of any neuron. The monitor and setup block can configure any neuron as a stimulus neuron by storing the timestep at which it should emit a spike in the memory space usually reserved for its V_{mem} and I_{syn}, and setting the neuron-specific control bits to indicate this is a stimulus neuron. This solution is adequate for the proposed networks as each stimulus neuron spikes at most once as we discuss in the next section. The entire design takes up 1% of the FPGA flip-flop resources (2460 registers), 4% of the logic resources (4065 slice LUTs), and 1% of the block ram resources (5 block RAM elements).

IV. FPGA Classification Performance

We trained the proposed spiking network on the MNIST classification task. The MNIST database contains 70,000 28x28 grayscale images of handwritten digits. The images were binarized to two intensity values: high and low. The training and validation sets of 60,000 labeled digits were combined and used for training. We used a feedforward spiking network with 784 input neurons, 600 hidden neurons, and 10 output neurons. Each of the input neurons represents a pixel. If a pixel has high intensity, the corresponding neuron spikes at time 0. Neurons corresponding to low-intensity pixels (the digit background) do not spike. We allow each hidden neuron to spike only once for each digit presentation and the classification result is the index of the output neuron that spiked first. The network state is reset after each digit presentation.

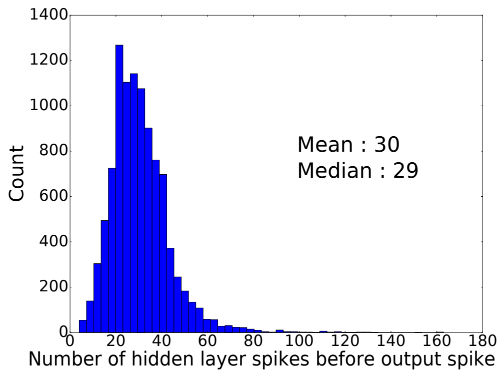
The network can halt as soon as one of the output neurons spikes (as the identity of the output neuron that spiked encodes the classification result). Training encourages the output neuron encoding the correct class to spike as early as possible, which often happens before the majority of the hidden layer neurons have even spiked. We tested the FPGA network on the 10000-digit test set of the MNIST dataset (the test set was not seen by the training algorithm). The network attained an error rate of 3.02%. Figure 2a shows the distribution of the number of hidden layer spikes that were emitted before the network made a decision by emitting a spike from an output neuron. The distribution is across the 10000 simulation runs for the test set digits. On average, only 5% (30 neurons) of the hidden layer neurons spike during the classification of a digit. The small subset of hidden neurons that spike are different for each digit. The input layer activity is also sparse since, on average, only 105 pixels have high intensity in an MNIST digit, i.e, on average, only 13% of the input neurons spike to represent the input digit.

The sparse nature of the spiking activity translates to a greatly reduced weight lookup overhead since only 5% of the hidden-to-output weights, and only 13% of the input-to-hidden weights need to be looked up to classify a digit on average. Especially in architectures where the weight memory is not embedded within the neural network accelerator, the reduced memory lookup requirements can translate to significant energy savings due to reduced weight memory traffic.

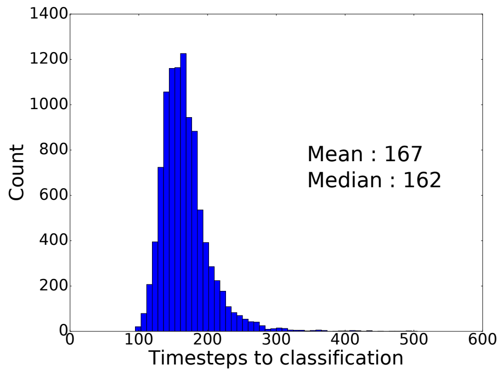
Figure 2b shows the distribution of the number of timesteps needed to produce an output spike. The classification output is produced on average after 167 timesteps. i.e, after $167/128 = 1.3$ synaptic time constants.

V. Conclusions

Exploiting or enforcing sparsity in neural activity is one of the most attractive ways to improve the efficiency of custom hardware implementations of various neural network architectures. In ANNs for example, sparse activity can be obtained by using rectifying non-linearities such as rectifying linear units [11] to obtain neuronal outputs that can be exactly zero, which can then be optimized away in subsequent computations or used to compress the representation of neural activity [12]. The SNN approach we have described in this paper utilizes an alternative form of sparsity that is specific to SNNs: by training the network in a way that encourages the correct output neuron to spike early, the network learns to make a decision(generate an output spike) and halt the computation as soon as possible. This time to first spike [13] classification model thus results in sparse activity as the computation is halted before the majority of hidden layer neurons had a chance to spike. This form of sparsity can not be realized in feedforward ANNs as they are evaluated layer-by-layer from input to output and can not incrementally carry out computations and then dynamically



(a)



(b)

Fig. 2: (a) Distribution of the number of hidden spikes needed to produce the classification result across the MNIST test set. (b) Distribution of the number of simulation timesteps until an output spike is produced.

halt once enough evidence for a particular classification class has been obtained.

The proposed spiking networks, however, are unable to match the accuracy of conventional ANNs on MNIST. Feed-forward fully-connected (non-convolutional) ANNs typically achieve error rates between 0.9% and 2% [14], while our SNN FPGA implementation achieves a 3.02% error rate. We believe this is partly due to the training pressure to produce an output spike as soon as possible, which forces the output neurons to spike quickly and ignore the information encoded in the activity of late-spiking hidden layer neurons. Rate-based SNNs with a similar architecture to ours can achieve error rates of 1.3% [15]. This, however, is achieved using thousands of spikes and long integration times, while in the proposed network, the total spike count is 135 spikes on average and the decision is made after a few synaptic time constants.

The sparse spiking activity in the proposed networks can result in siginificant energy savings compared to spiking networks based on rate-coding. Indeed, for architectures that need to fetch weights from external memory on each spike, an overwhelming portion of energy dissipation would come from accessing the off-chip memory, making a reduction in spiking activity a top priority. Reduction in spike activity also cuts the energy cost in the spike routing fabric. This cost could also be signifcant, especially in cases where the network is distributed across multiple chips. Biological networks are also organized in order to reap the advantages of sparse representations: in the visual processing pathway in the brain, neural activity is highly sparse [16]. Representing each stimulus using relatively few active neurons is more energetically efficient than a dense representation [17]. Due to the large number of neurons, the representational capacity of such a sparse coding scheme is extremely large [18].

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A re-configurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in Neuroscience*, vol. 9, no. 141, 2015. [Online]. Available: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2015.00141/abstract

[3] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*. IEEE, 2014, pp. 675–678.

[4] P. Merolla, J. Arthur, R. Alvarez-Icaza, A. Cassidy, J. Sawada, F. Akopyan, B. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[5] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in Neuroscience*, vol. 7, no. 178, 2013. [Online]. Available: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00178/abstract

[6] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[7] 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.

[8] S. Esser, P. Merolla, J. Arthur, A. Cassidy, R. Appuswamy, A. Andreopoulos, D. Berg, J. McKinstry, T. Melano, D. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, 2016.

[9] S. Bohte, J. Kok, and H. La-Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[10] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *arXiv preprint arXiv:1606.08165*, 2016.

[11] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[12] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 262–263.

[13] W. Maass and C. Bishop, *Pulsed Neural Networks*. MIT Press, 1998.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[15] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, 2016.

[16] S. Thorpe, A. Delorme, R. V. Rullen *et al.*, "Spike-based strategies for rapid processing," *Neural networks*, vol. 14, no. 6-7, pp. 715–725, 2001.

[17] B. Olshausen and D. Field, "Sparse coding of sensory inputs," *Current opinion in neurobiology*, vol. 14, no. 4, pp. 481–487, 2004.

[18] P. Földiák and M. Young, "Sparse coding in the primate cortex," *The handbook of brain theory and neural networks*, vol. 1, pp. 1064–1068, 1995.