IEEE International Symposium on Circuits and Systems, MAY 2017, this is not the final revision of the paper

Hardware Implementation of Convolutional STDP for On-line Visual Feature Learning

A. Yousefzadeh[1], T. Masquelier[2], T. Serrano-Gotarredona[1], and B. Linares-Barranco[1]

[1]Instituto de Microelectrónica de Sevilla (CSIC and Univ. de Sevilla), Sevilla, Spain {reza, bernabe}@imse-cnm.csic.es
[2] CERCO UMR 5549, CNRS – Université de Toulouse, F-31300, France

***Abstract*— We present a highly hardware friendly STDP (Spike Timing Dependent Plasticity) learning rule for training Spiking Convolutional Cores in Unsupervised mode and training Fully Connected Classifiers in Supervised Mode. Examples are given for a 2-layer Spiking Neural System which learns in real time features from visual scenes obtained with spiking DVS (Dynamic Vision Sensor) Cameras.**

***Keywords*— Neuromorphic Systems, Spike Time Dependent Plasticity (STDP), Spiking Neural Networks, Hardware Implementation of Neural Systems, Learning Systems.**

I. INTRODUCTION

Biological brains constantly learn new incoming information. They never stop learning. Our goal in this work is to develop an embedded and low power hardware for online unsupervised learning of visual features by using bio-inspired Dynamic Vision Sensors (DVS) [1] and spiking neural networks (SNNs). SNNs have interesting features like event-driven power consumption and pseudo-simultaneity [2].

In this work, we developed and implemented a new algorithm for hardware implementation that has been inspired by Masquelier's pioneering work on STDP (Spike Time Dependent Plasticity) [3]. They developed an algorithm for face recognition using still image frames. Intensity to delay conversion was used to generate artificial spike trains from each frame. They used simple Integrated-and-Fire (IF) neurons without leakage because after each frame presentation, the network resets all neuron states. They allowed maximum one spike per each neuron for each frame.

In our application, we wanted to use a DVS camera as the input source. However, a DVS does not use intensity to delay encoding. A DVS pixel generates a signed event when there has been a given relative change in light ($\Delta I/I = C$). Additionally, we wanted to perform training on the continuous input event flow coming from the DVS. For this, we rely on synchrony detection, which is very close to what happens in biological perception [4]. Synchrony based processing is a kind of temporal processing (as oppose to rate encoding) but it does not rely on precise spike ordering (as opposed to rank order encoding). What matters is that spikes appear close enough to each other in time. In synchrony-based processing, a visual feature is represented by pseudo-synchronous spikes coming from specific synapses.

Bichler et al. [5] introduced an interesting algorithm for online feature extraction based on STDP. The algorithm successfully detected cars passing from a freeway by using DVS input with unsupervised learning. They used a simplified STDP version to enhance processing speed. However, because they used fully connected neurons topologies, different neurons learned the same features in different positions.

By using Convolutional Neural Networks, one can share a set of weights (kernel) between many neurons. In this way, each kernel learns a feature independent of its position. Convolutional Spiking Neural Networks (CSNN) are more efficient than fully connected SNNs in terms of processing and memory for pattern recognition tasks.

In the next Sections we will briefly explain the learning algorithm and its hardware implementation.

II. LEARNING ALGORITHM

In this work, we used a simple 2-layer neural network, as shown in Fig. 1. The first layer is an unsupervised learning convolutional layer for feature extraction. The second layer is a simple supervised-learning classifier. Both layers use a simplified STDP-based learning rule, explained later in this Section.

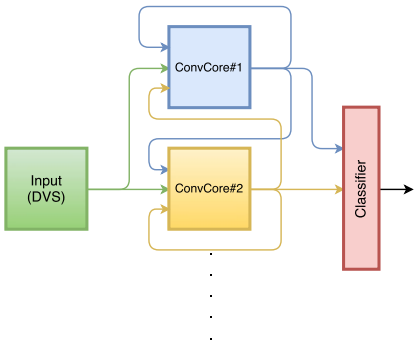


Fig. 1. Simple Network Topology used in this Work

A. Neuron model

The first layer implements a reduced number of convolutional populations equipped with STDP for feature detection through unsupervised learning. Convolutional populations are called here also "ConvCore". Each ConvCore

can have a few kernels that can be fixed or plastic (with STDP). In this Section we just used one layer of ConvCores and we used positive input spikes coming from the DVS output. For inhibition, we used reset rather than applying an inhibitory kernel. We used simple Leaky Integrated and Fire (LIF) neurons. Each incoming spike adds the synaptic weight to the membrane value[1]. When the neuron's membrane value reaches a threshold, it generates a post-synaptic spike, increase threshold of ConvCore to the current membrane value and reset its membrane value. This simple threshold adaptation mechanism is implemented to regulate activity of ConvCores and guarantee completion between them.

Leakage in these neurons has been implemented by using an approximation for the exponential decay. One option for leakage implementation is to update each neuron only when it receives spikes. This method is fully event-driven, but needs to keep track of last update time for each neuron. If processing time is more important than memory consumption, such fully event-driven neuron is recommended. However, in FPGA implementations, memory limitations are typically more stringent. For this reason we choose to not store the last update time for each neuron, but to apply leakage to all neurons periodically. The update period depends on the stimulus but for normal real-time DVS data, 1ms is reasonable. It will take just a few micro-seconds in the FPGA to update the leakage for all the neurons.

For the exponential leakage approximation, we implemented the following operation on the membrane value

$$V_{new} = V - (V \gg n_{leak}) \qquad (1)$$

where symbol (>> n) represents a bitwise right shift of n bits. We used $n_{leak} = 4$, which is equivalent to leaking to 15/16 of the previous membrane value every millisecond. This corresponds to a membrane equivalent time constant of 16ms, which is in the biological range.

The same leakage circuit is used for thresholds as part of threshold adaptation method. This leakage will allow decreasing of threshold for ConvCores that were inactive for a considerable amount of time.

If kernel size is [Kx, Ky], each neuron from the previous layer is connected to $Kx \times Ky$ neurons in a ConvCore with synaptic weights equal to the kernel values. After adding the kernel to membrane values of the neurons in a ConvCore, if a neuron's membrane value exceeds the threshold, an output spike will be generated. Then, the learning process updates the kernel weights.

To guarantee competition during learning, we used a winner-take-all mechanism [6] and inhibitory kernels as shown in Fig. 2. First of all, after updating neurons with an event kernel projection, more than one neuron's membrane value may have reached its threshold. In this case, only the one with the highest value among all the ConvCores will generate a spike. Afterwards, all the neurons in the ConvCore that fired will be reset to their resting value. This will stop neurons in the same ConvCore to learn multiple features. By using this mechanism, a ConvCore can learn only one feature in different positions. At the same time, an inhibitory kernel inhibits neurons in the other ConvCores in the same kernel area (see Fig. 2). For this we used a simple reset rather than applying an inhibitory kernel. This second competition mechanism is needed to discourage different ConvCores from learning the same features.

B. Layer 1: Unsupervised Convolutional STDP Learning

STDP is a bio-inspired learning rule that modifies the strength of a neuron's synapses as a function of the precise temporal relations between pre- and post-synaptic spikes [7]. There are different variants of STDP rules but all of them share a common concept. Synaptic weights are updated on a per-spike basis and the synaptic update depends on the time difference between pre- and post-synaptic spikes.

Here we used a new STDP rule which is highly efficient for hardware implementation, if not the most reported so far. It is very similar to Bichler's proposal [5], where all the synapses of a neuron are equally depressed upon reception of a post-synaptic spike, except for the synapses that were activated with a pre-synaptic spike a short time before, which are strongly potentiated. Synapses that were active shortly before post synaptic spikes are potentiated. Therefore, implementation of such a rule needs to store a timestamp for the incoming spikes. Also, a buffer is needed to save the last incoming spikes and the size of this buffer depends on the input spike rate. In hardware implementations, normally buffer sizes are fixed and cannot be adjusted via a parameter. Consequently, it is hard to estimate the best buffer size for all the applications.

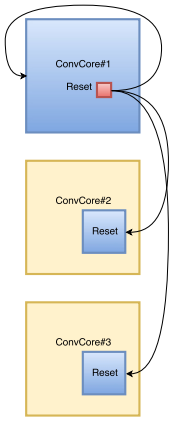


Fig. 2. Competition mechanisms. After a kernel update in ConvCore#1, the neuron in red has max value after passing the threshold, and therefore is the only one spiking within the kernel area. After this, all neurons in ConvCore#1 will be reset, as well as all neurons of the other ConvCores inside the kernel area (region in blue will be reset).

We have modified this rule to make it more hardware friendly. In the proposed STDP rule rather than limiting the

[1] We intentionally use here the term "membrane value", as opposed to the more conventional terminology of membrane voltage or potential, because in our hardware implementation it will be stored as a plain 9-bit integer value in the interval [0, 511], which is completely different from a physical voltage in the range of millivolts.

pre- to post- time window, we limited the number of synapses to be potentiated. This way, when a post synaptic spike is generated, a logic block will find specific number of active synapses that have contributed in the firing. In our proposed rule, there is no need to do time stamping on spikes because always a predefined number of synapses will be potentiated. If parameters are chosen carefully, leakage will not allow a neuron to fire in case the last pre-synaptic spikes arrived long time before the post synaptic spike, thus preserving synchrony. This rule also stops general potentiation or general depression. In addition, we added another mechanism to equally potentiate all the selected synapses regardless of number of spike coming from a synapse. kernel weights are normalized after potentiation. This way, all the synapses will be depressed equally with an adaptive rate.

C. *Layer 2: Supervised STDP Learning*

To classify the ConvCores output activities, a layer of simple fully connected supervised STDP neurons has been designed as output layer of our spiking neural network. A supervised STDP neuron has an extra external input (called "supervisor"), encoded also through AER (Address Event Representation), which forces post-synaptic spikes from this neuron when its representative "category" (or feature) is present at the input. Therefore, whenever a "supervisor" spike arrives, the corresponding active synapses will be potentiated. Otherwise active synapses will be depressed.

III. Hardware Implementation

To do real-time learning and feature extraction, we implemented the above algorithm with HDL (Hardware Description Language) on FPGA[2]. Fig. 3 shows the hardware setup that was used. We used a silicon retina (DVS) as input and a Spartan-6 FPGA Node-Board [8] for the network. USB-AER2 [9] boards were used to send spikes in real-time to a computer for visualization.

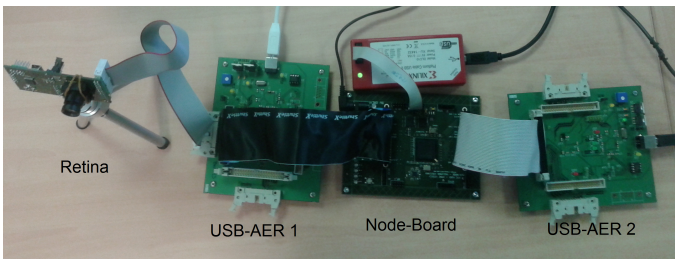


Fig. 3. Hardware Setup for Learning Experiments

Fig. 4 shows the block diagram of the FPGA implementation inside the Node-Board Spartan-6 FPGA. It contains ConvCores and supervised STDP Neurons core and AER interfaces (to handle asynchronous communications with outside FPGA). The number of ConvCores and the configuration of layers can be customized. Different cores communicate with Address Event Representation (AER) events [9].

A conceptual block diagram of the ConvCores is shown in Fig. 5. Each ConvCore contains a convolutional processor to perform convolution and two RAMs to keep the neurons membrane values (Neuron RAM) and synaptic weights (Kernel RAM). A STDP processor is shared between all ConvCores in one block because STDP events are rare and only one STDP processor is fast enough to handle them. STDP processor connected to a circular buffer to keep the last spikes and use them in STDP learning. Xilinx's Chipscope debug tool is used to program the initial parameters and to monitor the kernels evolution online using a computer.

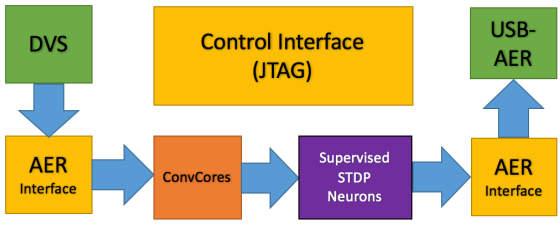


Fig. 4. FPGA System Implementation Block Diagram (DVS and USB-AER boards are outside of FPGA)

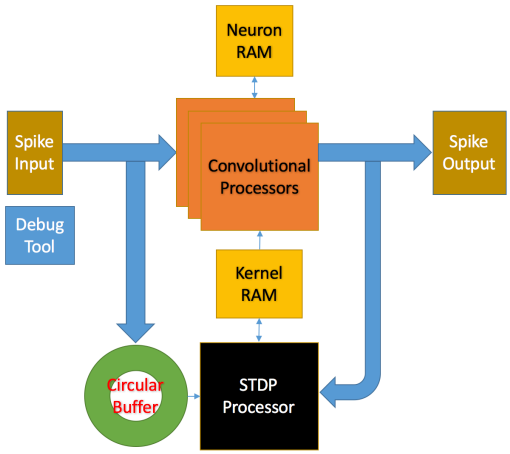


Fig. 5. Simplified Block diagram of ConvCore in FPGA

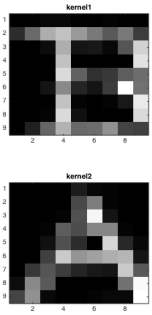
The amount of recourses needed to implement the ConvCore scheme on FPGAs depends on number of neurons and kernels. For example, once we implemented this core on a Spartan-6 FPGA (XC6SLX150T-3) with 32x32 input pixels, 6 kernels with 9x9 weights and 512 words for the circular buffer. With these parameters, ConvCores take 1276 slices (out of 23K available slices) of FPGA. Among these occupied slices, 587 slices belong to convolutional processors and 537 slices belong to STDP processor. Update of membrane values for each input event takes 90 clock cycles and STDP learning takes less than 900 clock cycles for updating kernels. Also, for each millisecond, one leakage update process takes 1025 clock cycles. When clock frequency is 100MHz, each convolution takes 0.9us and each STDP process takes less than 9us. These delays are reasonable for on-line learning in real-time.

[2] To receive Verilog codes please send an email to bernabe@imse-cnm.csic.es

IV. Implementation Results

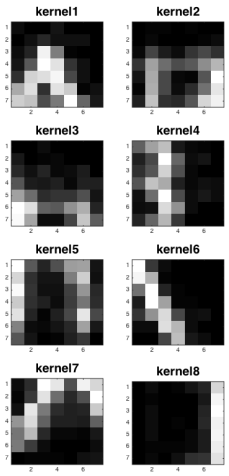
To test our network, we used two simple letters ('A' and 'B') and moved them in front of the DVS to generate spikes. Fig. 6 (a) shows a screenshot from the jAER software [10] used to visualize DVS spikes. Due to space limitation, we only presented the results of layer 1.

(a)



(b)

(c)



(d)

Fig. 6. (a, c) Screen captures of jAER software to visualize DVS output. Black dots show negative spikes while white dots show positive ones. (b, d) Reconstruction of kernel weights after learning. To see the complete recording videos (including parameters) and online evolution of kernels refer to [11].

STDP kernels learn the features that repeat more. For Convolutional STDP, when kernel size is larger than the object, we expect the kernel to learn the whole object. Otherwise, kernels should learn just some parts of the objects as features. Different objects may have common features, so it is natural to extract characteristic features and use them for object recognition.

We tested our hardware with kernel sizes of 9x9 while subsampling the output of the DVS from 128x128 down to 32x32. First we used two ConvCores. Fig. 6 (b) shows the reconstruction of kernel weights after learning the input shown in Fig. 6 (a). In this case because sizes of objects are smaller than '9x9', kernels learned the whole object templates.

In another experiment we presented the same stimulus to DVS but closer, so that the objects resulted in sizes larger than the kernels, as shown in Fig. 6 (c). In this case we used eight kernels of 7x7. The kernels learned characteristic features from the two letters. Reconstructed kernel weights for this experiment are shown in Fig. 6 (d).

V. Conclusions

In this work we have introduced a digital implementation of an algorithm for online STDP learning of visual features by using real live visual data from a DVS camera.

VI. Acknowledgements

This work was supported in part by the EU H2020 grants 644096 "ECOMODE" and 687299 "NEURAM3", by Samsung Advanced Institute of Technology grant NPP, by Spanish grants from the Ministry of Economy and Competitivity TEC2012-37868-C04-01 (BIOSENSE) and TEC2015-63884- C2-1-P (COGNET) (with support from the European Regional Development Fund), and by Andalusian grant TIC-6091 (NANONEURO). ARY is supported by a Spanish FPI Scholarship from the Ministry of Economy and Competitivity.

VII. References

[1] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3us Latency 4mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Amplifiers," *IEEE J. Solid-State Circuits,* vol. 48, no. 3, March 2013.

[2] J. A Pérez-Carrasco e. al., "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing--Application to Feedforward ConvNets," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35, no. 11, Nov. 2013.

[3] Timothée Masquelier et al, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLoS Comput Biol ,* 2007.

[4] Timothée Masquelier e. al., "Microsaccades enable efficient synchrony-based coding in the retina: a simulation study," *Scientific Reports,* vol. 6, 04 2016.

[5] O. Bichler et al., "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity," *Neural Networks,* vol. 32, 8 2012.

[6] M. Oster et al, "Computation with spikes in a winner-take-all network," *Neural Comput.,* Sep. 2009.

[7] H. Markran et al, "Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs," *Science,* vol. 275, 01 1997.

[8] T. Iakymchuk et al, "An AER handshake-less modular infrastructure PCB with x8 2.5Gbps LVDS serial links," *IEEE International Symposium on Circuits and Systems (ISCAS),* June 2014.

[9] Rafael Serrano-Gotarredona et al, "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing–Learning–Actuating System for High-Speed Visual Object Recognition and Tracking," *IEEE Transactions on Neural Networks,* Sept. 2009.

[10] F. Corradi et al, "jAER," [Online]. Available: https://sourceforge.net/projects/jaer/.

[11] A. Yousefzadeh, "Real-Time Videos," 2016. [Online]. Available: https://youtu.be/05NyI3qp05g & https://youtu.be/VRz1uLXa4KA