Spiking Neural Network Self-configuration for Temporal Pattern Recognition Analysis

Josep L. Rosselló, Ivan de Paúl, Vincent Canals, and Antoni Morro

Electronic Systems Group, Physics Department,
Universitat de les Illes Balears (UIB),
07122 Palma de Mallorca, Spain
j.rossello@uib.es

Abstract. In this work we provide design guidelines for the hardware implementation of Spiking Neural Networks. The proposed methodology is applied to temporal pattern recognition analysis. For this purpose the networks are trained using a simplified Genetic Algorithm. The proposed solution is applied to estimate the processing efficiency of Spiking Neural Networks.

Keywords: Neural Networks, Spiking Neural Networks, Hardware implementation of Genetic Algorithms.

1 Introduction

The development of efficient solutions for the hardware implementation of neural systems is currently one of the major challenges for science and technology. Due to their parallel-processing nature, among other applications, neural systems can be used for real-time pattern recognition tasks and to provide quick solutions for complex problems that are intractable using traditional digital processors [1,2]. The distributed information processing of Neural Networks also enhances fault tolerance and noise immunity with respect to traditional sequential processing machines. Despite of all these processing advantages, one of the main problems of dealing with neural systems is the achievement of optimum network configurations since network complexity increases exponentially with the total number of neural connections. Therefore, the development of learning strategies to quickly obtain optimum solutions when dealing with huge network configuration spaces is of high interest for the research community.

Recently, a lot of research has been focused on the development of Spiking Neural Networks (SNN) [3] as they are closely related to real biological systems. In SNN information is codified in the form of voltage pulses called Action Potentials (APs). At each neuron cell the AP inputs are weighted and integrated in a single variable defined as the Post-Synaptic-Potential (PSP). The PSP is time dependent and decays when no APs are received. When input spikes excite the PSP of a neuron sufficiently so that it is over a certain threshold, an Action Potential is emitted by the neuron and transmitted to the rest of the network.

C. Alippi et al. (Eds.): ICANN 2009, Part I, LNCS 5768, pp. 421–428, 2009.

In this work we present a practical implementation of SNN. We also develop a simple architecture that can be used for training SNN. The proposed self-learning solution has been applied for temporal pattern recognition. Based on this study, we also provide a new metric to estimate the processing capacity of NN. The rest of this paper is organized as follows: in section 2 we show the proposed SNN self-learning architecture, while in section 3 we apply the proposed system to temporal pattern recognition analysis. Finally in section 4 we present the conclusions.

2 Digital SNN Architecture

2.1 Digital Spiking Neuron Model

As mentioned in the previous section, in SNN the information is codified in the form of voltage pulses. We used a simplified digital implementation of the real behavior of biological neurons. In the proposed system, the PSP decay after each input spike is selected to be linear instead of the real non-linear variation while the refractory period present after each spike emission has been neglected. The main objective of this work is not to provide an exact copy of the real behavior of biological systems but to develop a useful Neural Network configuration technique. In Fig. 1 it is shown an example of the dynamic behavior of the digital model implemented.

In the digital version implemented the PSP is codified as a digital number. At each spike integration the PSP is increased a fixed value that depends on the type and the strength of the connection. Therefore, positive (negative) increment values are associated to excitatory (inhibitory) connections. Each neuron is implemented using a VHDL code in which the connection strength is selected

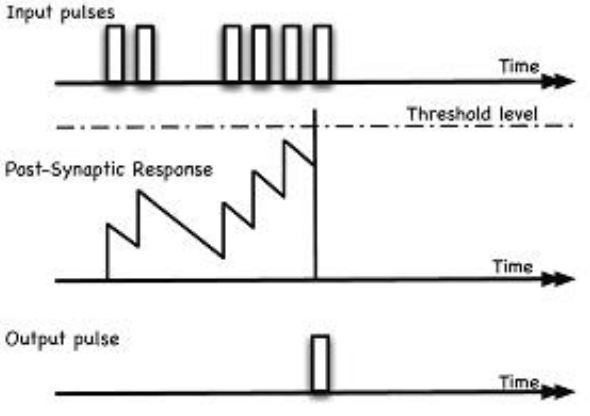

Fig. 1. Dynamic behavior of the digital implementation of spiking neurons. Action potentials are represented as digital pulses while post-psynaptic potential variation is assumed to be linear with time. The typical refractory period after each output spike has been neglected in this model.

to be a fraction of the neuron threshold (in particular, we selected a fraction of $\pm 2/5$ for both excitatory and inhibitory connections).

2.2 Self-learning Architecture

The proposed self-learning architecture is shown in Fig. 2. It consists in two basic blocks, a Genetic Algorithm Circuitry (GAC) and a Fitness Circuitry (FC). The GAC generates new configurations based on the better configuration obtained, that is stored in the configuration register. Using a Random Number Generator (RNG) a random mutation vector is generated. The mutation vector is operated using XOR gates with the better configuration found until the moment (placed in the configuration register). The result is a new configuration (binary output of XOR block) that is equal to the previous except in those cases where the RNG provides a HIGH state. The new configuration is directly applied to the SNN when the controlling signal of the GAC multiplexer (SL) is HIGH (self-learning selection). When signal SL is LOW (operation mode) the better configuration obtained until that moment is applied to the SNN.

The FC block evaluates the aptness of each new configuration for a selected network task. During the training mode an evaluation circuitry (EC) compares the SNN behavior with respect to the expected behavior, thus evaluating a cost function (the configuration fitness). The value obtained in this process is then compared to the one associated to the better configuration at the moment (stored at the fitness register). When a better fitness is found at the end of the evaluation time, the digital comparator output is set to a HIGH state and both the fitness and the configuration registers are updated with the new values. When the system is in operation mode, the SNN configuration is fixed to the better solution obtained at the moment. A global reset is used to start with pre-selected initial conditions.

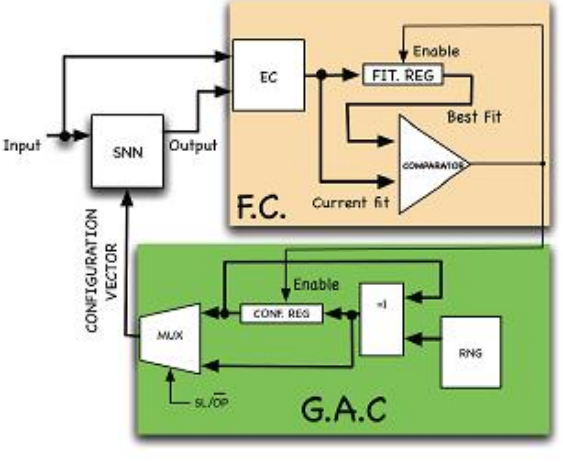

Fig. 2. Block structure for the dynamic configuration of SNN. The GAC block is used for the network configuration while the FC block evaluates the network efficiency.

2.3 Random Vector Generation

The mutation vector is used to generate a new configuration that is equal to the previous one, except in those cases where the RNG provides a HIGH state. The election of the RNG is important since all the possible mutation vectors must have the same probability of being generated. Therefore, the percentage of mutation ranges between the 0% (mutation vector 00...0) and the 100% (mutation vector 11...1). Using this strategy we ensure the possibility of moving from a local minima to a deeper (and therefore better) minima. Make note that, since the system is directly implemented in hardware it can sweep millions of different configuration vectors per second, thus obtaining a good solution in a reasonable time (although, of course, the absolute minimum is not guaranteed).

For the generation of the mutation vector we can choose either a pseudo-random or a random number generator. In FPGA applications we can use the first one since it is easily implemented using LFSR registers. For VLSI implementations we can choose a lower-cost solution as a true random number generator [4]. The solution proposed in [4] represent a lower cost in terms of hardware resources if compared to LFSR counters.

3 Application to Temporal Pattern Recognition

We applied the proposed SNN architecture to evaluate the processing behavior of various networks. The selected SNN task is the temporal pattern recognition (that is directly related to the "memory" capacity of the system). During the training mode, a finite sequence of vectors is repeatedly applied at the SNN input (the training bit sequence) and the task of the network consists in recognizing the sequence: at each time step the SNN has to provide the next bit of the sequence (see the illustration of Fig. 3). The network efficiency is evaluated estimating the probability of the SNN prediction success. At each evaluation

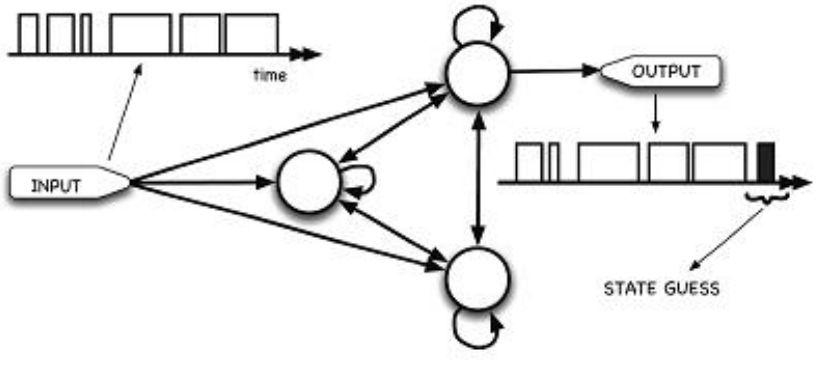

Fig. 3. A complete SNN implements all the possible inter-neuron connections. Such networks are trained to recognize temporal patterns.

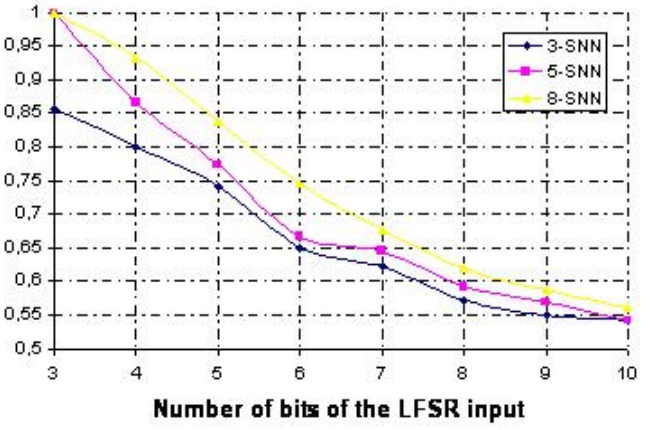

Fig. 4. Success probability of a spiking neural network of 3, 5 and 8 neurons assembled in a complete topology. As can be appreciated, the total "memory" of the system is proportional to the number of neurons of the network.

step a mutated configuration provided by the GAC is used to configure the SNN. The FC bloc evaluates the probability of success of SNN predictions and the mutated configuration is therefore stored or discarded (see Fig. 2).

We configured different networks containing three, five and eight neurons, each one connected to the rest of the network thus assembling a complete topology. In Fig. 3 we show the case with three neurons (defined as a 3-SNN).

The training bit sequence must be as complex as possible to maximize the pattern recognition difficulty. Therefore, we selected the generation of pseudo-random strings provided by LFSR digital blocks. Pseudo-random bit strings are characterized to have the same statistical properties as random sequences with the only characteristic that pseudo-random sequences have a periodicity. In our experiment we used training sequences of N=7, 15, 31, 63, 127, 255, 511 and 1023 bits (using LFSR of 3 until 10 bits length). With the selection of this type of pseudo-random sequences, the memorization task difficulty is maximized.

We applied each pseudo-random training sequence to three different SNN with complete topology (using 3, 5 and 8 neurons). Each SNN is configured by the proposed genetic-based self-learning architecture. At each time step, the network has to guess the next bit that will be provided by the LFSR. Once the configuring circuitry has been stabilized to an optimum configuration we evaluate the probability of success associated to this final configuration. In Fig. 4 we provide the different prediction success for each network as a function of the number of bits of the LFSR. It is observed that, as the number of bits of the sequence increases and as the number of neurons decreases the network presents a lower prediction success. We defined a new performance metric of Neural Networks that we refer to as the M-index that measures the ability of the system to recognize a pseudo-random bit sequence. The SNN M-index is defined as *the maximum pseudo-random sequence length that the network is able*

to recognize with a 90% of success probability. This index is a good indicator of the network processing capability, independently on the type and topology of the selected neural system. The M-index of the three networks were 5, 12 and 22 for the 3-SNN, 5-SNN and the 8-SNN respectively.

4 Conclusions

In this work we proposed a simple architecture for SNN self-configuration. The proposed system implements in hardware a simplified genetic algorithm. We applied the proposed architecture to temporal pattern recognition analysis. Different pseudo-random sequences were applied to different networks and the success probability was evaluated. A new performance metric has been developed that may be applied to measure the processing capacity of networks.

Acknowledgments. This work was supported in part by the Balearic Islands Government in part by the Regional European Development Funds (FEDER) and in part by the Spanish Government under projects PROGECIB-32A and TEC2008-04501.

References

1. Malaka, R., Buck, S.: Solving nonlinear optimization problems using networks of spiking neurons. In: Int. Joint Conf. on Neural Networks, Como, pp. 486–491 (2000)
2. Sala, D.M., Cios, K.J.: Solving graph algorithms with networks of spiking neurons. IEEE Trans. on Neural Net. 10, 953–957 (1999)
3. Gerstner, W., Kistler, W.M.: Spiking neuron models. Cambridge University Press, Cambridge (2002)
4. Rosselló, J.L., Canals, V., de Paúl, I., Bota, S., Morro, A.: A Simple CMOS Chaotic Integrated Circuit. IEICE Electronics Express 5, 1042–1048 (2008)

Appendix: VHDL Code of the Digital Spiking Neuron

Each neuron is implemented using a VHDL code in which the connection strength is selected to be a fraction of the neuron threshold. Subsequently we show the VHDL code for a digital neuron with four inputs and the previously described characteristics. Weight is selected to be equal to $\pm 2/5$ (threshold value of 20 for the post-synaptical potential and weight of ± 8).

```
-- VHDL model for a digital Spiking Neuron********
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY neurona IS

    PORT -- each input is codified with two bits
    (
        clk         : IN    STD_LOGIC;
        in1         : IN    STD_LOGIC;
        in1x    : IN    STD_LOGIC_VECTOR(0 to 1);

        in2         : IN    STD_LOGIC;
        in2x    : IN    STD_LOGIC_VECTOR(0 to 1);

        in3         : IN    STD_LOGIC;
        in3x    : IN    STD_LOGIC_VECTOR(0 to 1);

        in4         : IN    STD_LOGIC;
        in4x    : IN    STD_LOGIC_VECTOR(0 to 1);

        outx        : out    STD_LOGIC

    );

END neurona;

ARCHITECTURE neuron OF neurona IS

BEGIN
canviestat:
PROCESS (clk)
    VARIABLE state: INTEGER RANGE 0 TO 31;
BEGIN
    IF (clk'EVENT) and (clk='1') THEN
        outx<='0';
        IF (in1='1' and in1x="01") THEN
            state:=state+8;
        END IF;
        IF (in1='1' and in1x="10") THEN
            state:=state-8;
        END IF;
        IF (in2='1' and in2x="01") THEN
            state:=state+8;
        END IF;
        IF (in2='1' and in2x="10") THEN
            state:=state-8;
        END IF;
        IF (in3='1' and in3x="01") THEN
```

```
            state:=state+8;
        END IF;
        IF (in3='1' and in3x="10") THEN
            state:=state-8;
        END IF;
        IF (in4='1' and in4x="01") THEN
            state:=state+8;
        END IF;
        IF (in4='1' and in4x="10") THEN
            state:=state-8;
        END IF;
        state:=state-1;
        IF (state>20) THEN
            outx<='1';
            state:=1;
        END IF;
        IF (STATE=0) then
            outx<='0';
        END IF;
    END IF;
END PROCESS canviestat;
END neuron;
```