Evolutionary FPGA-based Spiking Neural Networks for Continual Learning*

Andrés Otero[1], Guillermo Sanllorente[1], Eduardo de la Torre[1] and Jose Nunez-Yanez[2]

[1] Centro de Electrónica Industrial,Universidad Politécnica de Madrid,Madrid, Spain
{joseandres.otero, g.sanllorente, eduardo.delatorre}@upm.es
[2] Department of Electrical Engineering, Linköping University, Sweden
jose.nunez-yanez@liu.se

Abstract. Spiking Neural Networks (SNNs) constitute a representative example of neuromorphic computing in which event-driven computation is mapped to neuron spikes reducing power consumption. A challenge that limits the general adoption of SNNs is the need for mature training algorithms compared with other artificial neural networks, such as multilayer perceptrons or convolutional neural networks. This paper explores the use of evolutionary algorithms as a black-box solution for training SNNs. The selected SNN model relies on the Izhikevich neuron model implemented in hardware. Differently from state-of-the-art, the approach followed in this paper integrates within the same System-on-a-chip (SoC) both the training algorithm and the SNN fabric, enabling continuous network adaptation in-field and, thus, eliminating the barrier between offline (training) and online (inference). A novel encoding approach for the inputs based on receptive fields is also provided to improve network accuracy. Experimental results demonstrate that these techniques perform similarly to other algorithms in the literature without dynamic adaptability for classification and control problems.

Keywords: Spiking Neural Networks; Evolutionary Algorithms; FPGAs

1 Introduction

Neuromorphic computing is gaining momentum in a new scenario emerging from the end of Moore's law. As an alternative to Von Neumann processors, neuromorphic computers substitute CPUs and memories with neurons and synapses and traditional binary encoding with spikes. The change in the computing architecture mitigates the CPU/memory bottleneck, moving to an inherently parallel strategy. At the same time, the use of spikes makes computation event-driven, obtaining low-power operation. As a complete revolution in the computing paradigm, neuromorphic computing requires novel physical realizations,

* This project has been funded by the European Commission under the project A-IQ Ready (GA. 101096658) and by the Knut and Alice Wallenberg Foundation under the Wallenberg AI autonomous autonomous systems and software (WASP) program

which have recently become available to the research community. Among them are solutions provided by academia (such as ODIN [1]) and industry (such as Intel Loihi [2] or ARM spinnaker [3]). Also, analog solutions based on memristors or in-memory computing paradigms are available in the state-of-the-art ([4]).

Although using neuromorphic computing principles for general-purpose computing is still in the future, it is already a reality for artificial intelligence workloads in which biological brains inspire architectures, such as SNNs. SNNs are envisaged as a high-potential technology but still with many challenges to reach the computational capabilities of other Artificial Neural Network (ANN) models, such as deep neural networks. Open research questions are related to the network (and neuron's) structure, the learning paradigms, and how to preserve the biological plausibility, temporal encoding, and low-power and low-rate features inherent to SNNs.

Mathematical models for biological neural behaviors have been known since the mid-20th century. The Hodgkin-Huxley model [5] describes the physiological mechanisms of neurons and prioritizes natural precision over mathematical simplicity. Alternatively, spiking-based models, such as the Izhikevich [6] or the Integrate and Fire [7], were proposed to reduce the mathematical complexity by describing the temporal behavior of cortical spike trains.

Along with the mathematical representation of the neuron behavior, it is also required to implement a learning model for the neural network. In this regard, this paper investigates the use of an evolutionary strategy for training biologically accurate spiking neural networks based on the Izikevich model. The SNN and the learning algorithm are integrated into the same SoC FPGA device, enabling continuous learning throughout the system's lifetime. This system has been adapted to solve various benchmarks and problems using supervised and reinforcement learning to demonstrate the usage of the network in complex real-time tasks. A novel receptive field strategy is proposed to encode the temporal information into spikes. Experimental results demonstrate the suitability of using evolution strategies for SNNs with hardware acceleration.

The rest of this paper is structured as follows. Section 2 reviews state-of-the-art learning techniques for SNNs. In Section 3, the architecture proposed in this paper is described, while the learning algorithm is described in Section 4. Section 5 describes the data encoding strategy, and the evaluation setup is described in Section 6. Experimental results are described in section 7, while conclusions and future work are shown in section 8.

2 Learning techniques for SNNs

This section describes the main approaches available in the state-of-the-art for learning in SNNs, including unsupervised, supervised, and reinforcement learning strategies.

The most popular unsupervised learning method targeting SNNs is Spike Timing-Dependent Plasticity (STDP). It is based on Hebb's rule introduced in 1949. In STDP, synaptic connections are reinforced based on interconnected

neurons' pre-synaptic and post-synaptic spike timings. Modifying the synaptic strengths leads to a new organization of the links in the neural network that may result in a learning phenomenon [8]. This learning technique is considered unsupervised since any explicit goal does not guide the learning process.

SpikeProp was one of the first attempts to use a supervised learning algorithm in multilayer SNNs [9]. It is an adaptation of the backpropagation algorithm used in ANNs, by introducing simplifications to cope with the discontinuous nature of spiking neurons. It is, therefore, one of the best-known, most extended learning methods based on Gradient Evaluation for SNNs. SpikeProp has been shown to deal effectively with complex problems. The algorithm was tested on various UCI datasets, such as the Iris dataset, the Wisconsin breast cancer dataset, and the Statlog Landsat dataset, using a feedforward network topology. One of the main drawbacks of SpikeProp is that it can only train a single spike for each neuron, which limits the diversity and information transmission in the SNN. Dealing with multiple spikes would be more biologically accurate than single-spike training [10].

Remote Supervised Method (ReSuMe) is another solution for single-spike training. As a supervised learning model, ReSuMe is based on error minimization between the recorded output spikes and the expected ones, which are calculated a priori based on the problem to solve. It is a temporally local algorithm, meaning that at every time step, the algorithm updates synaptic weights only for the nearest target firing times [11]. One of its main advantages is that the algorithm has been proven independent of the used neuron models. However, ReSuMe is also unsuitable for multilayer SNNs, and different algorithm adaptations have been proposed to overcome this issue [12].

In contrast to the previous learning strategies, evolution strategies are optimization techniques inspired by nature, using concepts such as mutation and selection as critical elements for exploring the design space and finding solutions in a neural network. This technique uses generations to represent the number of loops tested on a particular problem, creating new individuals through mutation for each generation. A. Belatreche et al. [13] proposed an evolutionary strategy for SNNs and tested its functionality through a Spike Response Model network. However, to the best of the author's knowledge, these evolutionary methods have not yet been tested in more biologically accurate models, such as the Izhikevich model, using hardware implementations to accelerate its computation times.

3 The Proposed SNN-based SoC Architecture

Taking inspiration from biological neural models makes SNNs computing demanding architectures. However, the simultaneous operation of all the neurons in the network makes them suitable for parallel implementations using hardware accelerators such as FPGAs or GPUs. In this work, the Izhikevich SNN model is implemented in an FPGA, which enables fast and parallel computation. In particular, the provided solution targets an FPGA SoC device (the Zynq MPSoC) to perform both training and inference at run-time, enabling the continuous evo-

lution of the network. MPSoCs combine different computing fabrics in a single device: CPUs, referred to as the Processing System (PS), and an FPGA, known as the programmable logic (PL). This combination generally results in higher computing performance, lower power consumption, and higher flexibility than homogeneous multi-core solutions. The spiking neural network is implemented as an accelerator in the PL, and the embedded dual-core ARM Cortex-A9 processor handles the learning tasks and the input spike generation. The spiking neural network is integrated as an Intellectual Property (IP) module, which is integrated with the PS using a Direct Memory Access (DMA) mechanism. In addition to using the same IP for training and inference, this structure has other advantages, such as the ability to embed more than one SNN accelerator inside the PL. In this work, solutions based on one and two IPs will be shown.

The SNN accelerator integrated as an IP is based on the work proposed at [14], consisting of an Izhikevich-based SNN architecture with up to 250K neurons initially implemented in a single Zynq 7020 device. Note that this implementation has not been tested on complex datasets before, so this work validates both the proposed evolution strategies and the SNN implementation. The SNN uses a fully connected feed-forward network, as shown in Fig. 1.

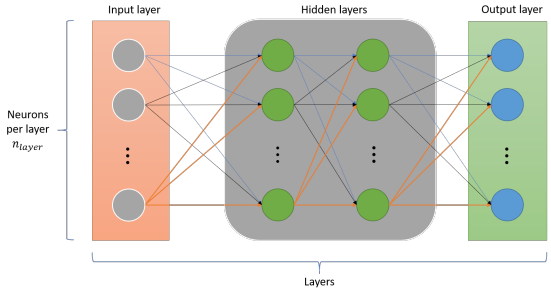


Fig. 1: Fully connected feed-forward topology.

It contains an input layer that serves as an interface with the processor in the PS. The hidden layers are internally allocated inside the IP and transparent to the user. Finally, the output layer is a delivery point for the network's results. The processor can adjust each neuron's internal weights at each time step, allowing the network parameters to be changed and trained even during runtime. This accessibility is crucial during the learning process guided by the evolutionary strategy. The same IP will be used with many candidate solutions with different weights as evolution parameters.

The IP can be customized using the network and the neuron model parameters. The network parameters constitute the main configuration points for the SNN, and they include the neurons per layer (n_layer), the number of inputs, the number of outputs, and the number of hidden layers (L). The number of inputs cannot exceed the neurons per layer parameter. Suppose a different number of

neurons is required in each network layer. In that case, the IP must be synthesized using the maximum number of neurons in a single layer as the neurons per layer n_{layer} parameter. The network architecture can be adjusted during runtime by setting the unused weights to zero, turning the synapses or even complete neurons into inactive ones. On the other hand, internal neuron parameters include the minimum time step, simulation time step, excitatory and inhibitory neuron probability, weight data precision, and internal Izhikevich neuronal model parameters. In this project, floating-point weights have been selected. The design provided initially in [14] is described in C++, targeting a High-level synthesis design methodology.

4 Proposed Learning Strategy

The learning strategy proposed in this work requires selecting a population size defined as a parameter (P) which can be modified depending on the complexity of the target application. An initializing function ($\lambda()$) fills all the genomes with random weights, creating a diverse spectrum of possible individuals at the beginning of the program. The total number of generations (G) is also a parameter that will be adapted depending on the complexity of the problem. The error or fitness function (E) represents a comparison between the actual and the target value. It is used to evaluate the achieved performance of a candidate SNN. This function will be defined, together with the algorithm parameters, for each problem to be solved. Finally, the mutation function, used as the bioinspired operator to create new individuals from the current population, uses a Gaussian distribution to generate random values from the previous ones. The overall goal of the algorithm is to obtain an optimal individual that, after some generations, can solve the required task.

More in detail, the proposed algorithm is based on an evolution strategy that uses mutation and elitism as bioinspired operators (Algorithm 1). The SNN weights (θ_i) are the parameters that the evolutionary procedure uses to explore the design space of candidate solutions, maintained as a population of individuals, each with a randomly initialized set of weights (*lines 2 - 4*). After initializing the original weights, each individual is evaluated by executing the evolved SNN against the target problem (*lines 7-8*). In each generation, the error achieved with the candidate under evaluation (E_i) is compared with the error previously stored for this member of the population (E_i^{Prev}) (*line 11*). When the performance of the new individual surpasses its predecessor, the parent is substituted in the population by the descendant, which is therefore preserved for future generations ($\theta_i^{Prev} = \theta_i$). This way of maintaining better predecessors is called elitism. In every iteration, an offspring is generated based on the parent's weight through mutation (*lines 13 and 15*).

The processors in the PS store the weight information for every individual during training. In each iteration, the weights are transferred to the SNN IP in the PL to evaluate the performance of the current individual for a given problem. Since the population comprises multiple independent individuals, several can be

Algorithm 1 Evolutionary Strategy for Learning in SNN

Require: population size P, error function E, number of generations G.
for $i = 0, 1, \ldots, P-1$ **do**
    Initialize population with random sets of weights: $\theta_i = \lambda()$
end for
for $g = 1, 2, \ldots, G$ **do**
    for $i = 0, 1, \ldots, P-1$ **do**
        Initialize SNN IP with weights θ_i
        Compute error: E_i
        if g=1 **then**
            Store individual: $\theta_i^{Prev} = \theta_i$, $E_i^{Prev} = E_i$
        else if $E_i < E_i^{Prev}$ **then**
            Store and substitute individual: $\theta_i^{Prev} = \theta_i$, $E_i^{Prev} = E_i$
            Create new individual from mutation: θ_i=mutate(θ_i^{Prev})
        else
            Create new individual from mutation: θ_i=mutate(θ_i^{Prev})
        end if
    end for
end for
return θ_P, E_P, A_P

evaluated simultaneously. This can be easily achieved when multiple SNN IPs are included in the SoC. This inherent parallelism is another benefit of the proposed strategy.

5 Data encoding strategy

One of the more relevant decisions needed when implementing SNN-based systems is how to encode the input data to be processed by the network. Different approaches exist in the literature, ranging from the most straightforward rate coding (i.e., information is provided by the number of spikes in a time window) to temporal encoding (i.e., information is provided by the exact time of spikes, for instance, the first one). In this work, a more complex approach is followed, exploiting the model of receptive fields proposed by S. M. Bohte et al. [21]. This approach utilizes a population of neurons with Gaussian activation functions, and each input variable is encoded using a variable number of neurons. The accuracy can be improved by sharpening the receptive fields and increasing the neurons affected by each input variable.

The data range for each input variable is filled with Gaussian fields that cover the entire data spectrum with an adjustable number of inputs per variable. For a variable n with range $[I^n_{max}, \ldots, I^n_{min}]$, m neurons are used. The width for all the neurons is set to $\sigma = \frac{1}{\beta} \cdot \frac{(I^n_{max} - I^n_{min})}{(m-2)}$ and for every neuron i, its center is set to $I^n_{min} + \frac{(2i-3)}{2} \cdot \frac{(I^n_{max} - I^n_{min})}{(m-2)}$. A value of 1.5 was experimentally decided to be used for β in the proposed implementation. Fig. 2 shows a graphical representation of this technique. For a given input (shown as the blue line in the top figure), the

crossing points with each Gaussian field are calculated (green triangles). These crossing points can be obtained using the probability density function for the input data. Higher values at these intersection points indicate stronger excitation for that variable, and these are transformed into lower delay times, rounded to the nearest discrete time step. For example, a value of 1 would be the maximum possible value obtained by the probability density function formula and would be then transformed into an input spike at time t=0. With the example of an input window of ten steps, values for each spike delay can go up to t=10. In this project, spike delays higher than t=9 are coded not to fire, as considered to be insufficiently excited.

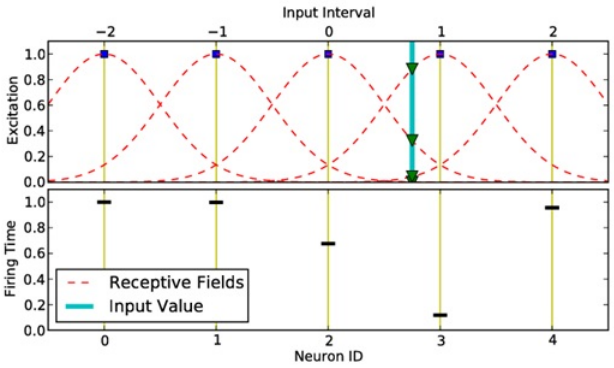


Fig. 2: Input Data encoding based on Gaussian receptive fields.

The encoding of the output produced by the SNN is also highly relevant. In supervised learning problems, the dataset's total number of instances is split into a training and a validation subset. The network is then evaluated using the training set as a reference, taking the exact timings of the output neuron spikes as the classification value assigned to a given label in the dataset. The classification error for novel samples in the dataset is computed as the distance between the target and actual spikes, as shown in Fig. 3.

The processors in the PS running the evolutionary algorithm must be aware of the SNN evaluation time to provide the inputs at the right time stamps and to understand the output in its context.

6 Evaluation Setup

The system proposed in this work has been implemented on the PYNQ-Z1 development board with a ZYNQ XC7Z020-1CLG400C MPSoC device. Two different sets of problems are used to demonstrate the adaptation capabilities of the system: classic classification problems from the UCI datasets, and control problems,

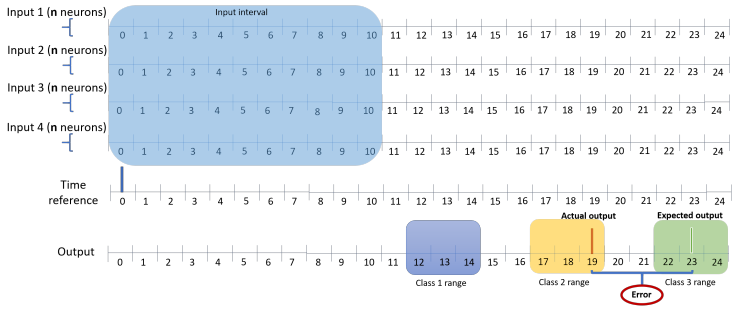


Fig. 3: Computing error in the network.

from the Gymnasium API (formerly, OpenAI's Gym library [15]). The selected problems are described next.

6.1 Iris Problem

The first of the UCI datasets selected is the Iris dataset [16]. It is one of the most well-known databases in the pattern recognition field. It contains three classes of 50 instances each, two of which are not linearly separable. The Iris dataset uses four attributes representing specific characteristics of different types of plants. The objective is to classify each instance into one of the three types of iris plants. Four Gaussian receptive fields have been selected for each variable to convert the original dataset into input spikes for the SNN. This results in 16 inputs, with an extra input used as a time reference. A single output neuron is included, with three different time windows used for classification. A complete scheme of the network's input and output encodings, including the time windows, is presented in Fig. 4. The size of the input time window is set to 9 execution time steps (light blue).

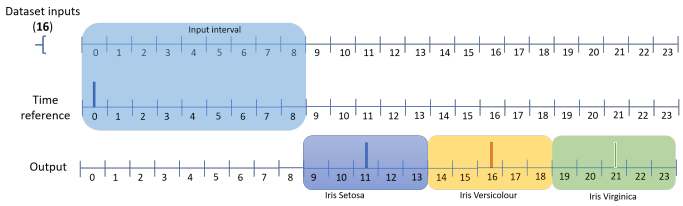


Fig. 4: Network configuration for the Iris dataset.

The dataset is divided into a training (70% of the instances) and a testing set (30%). Both groups are randomly selected, with the same proportion of classes as in the original dataset. The error function is calculated using the accumulated

time-step distances between the actual and expected spikes as its main parameter. However, during the experimental tests, it has been observed that including penalties for non-desired behaviors helps the network to converge faster. The proposed penalty system favors if the spikes for the different classes are correctly ordered, even if they are out of the bounds reserved for the classification. For example, an Iris Virginica instance that spikes after time step 23 (following Fig. 4) is less penalized than an Iris Virginica instance that spikes at time step 16. However, if the output neuron does not spike during the trial time (100 ms), a maximum penalty is applied to the candidate SNN.

The complete list of parameters used to adapt the system for each of the datasets is included in Table 1.

Table 1: Parameters of the Evolutionary Strategy for the different datasets.

Parameter	**Iris**	**Brest Cancer**	**Pima Diabetes**	**Wine**	**Montain Car**
Training trials	105 (70%)	350 (50.07%)	385 (50.13%)	125 (70.22%)	—
Testing trials	45 (30%)	349 (49.93%)	383 (49.87%)	53 (29.78%)	—
Hidden layers	1	1	1	1	1
Neurons per layer	17	37	33	53	17
Hidden neurons	17	37	33	14	17
Input neurons	17	37	33	53	17
Output neurons	1	1	1	1	1
Total neurons	35	75	67	68	35
Population	20	20	20	20	5
Trial time (ms)	100	100	100	100	100

6.2 Breast Cancer Wisconsin Dataset

This dataset uses nine different attributes, resulting in 37 inputs for the network (with 4 Gaussian receptive fields for each feature). A single neuron classifies the data, dividing the output spectrum into two windows. The error function implemented for this dataset also penalizes the case in which the output neuron does not fire during the trial time. If it does fire, the error is computed using the distance to the correct time window for the output spike.

6.3 Pima Indian Diabetes Dataset

The following dataset considered was the Pima Indian Diabetes dataset, whose purpose is to predict whether a patient has diabetes based on diagnostic measurements included as attributes of the dataset. In this dataset, eight features are used to classify the instances into two classes, corresponding to either a positive or a negative prediction. For this purpose, 33 input neurons are used, with just

one output neuron. The output window is split into two parts, representing the two classes. The error function is the same as seen in the Breast Cancer dataset.

6.4 Wine Dataset

The last dataset selected from the UCI repository is the Wine dataset. It includes data about the chemical analysis of wines derived from three different cultivars, which comprise the three possible output classes. The network configuration is derived from the one used for the Iris dataset. An error function favoring the correct order of spikes for the three different classes is applied, as already seen in the case of the Iris dataset. The list of parameters corresponding to this implementation is listed in Table 1, with an initial proposal of 53 neurons inside of the hidden layer. Because of the limited number of instances of the dataset and the large number of neurons and synapses included, the network had difficulty converging within a reasonable time. To solve this problem, some neurons' weights were set to zero to implement only 14 neurons in the hidden layer, which proved more effective, as shown in the experimental results.

6.5 The Mountain Car environment

The mountain car environment is a deterministic Markov Decision Process in which the goal is to accelerate a car placed at the bottom of a valley to reach the top of the right hill. The rendered environment can be observed in Fig. 5. It belongs to Gymnasium, a standard API for RL, created as a maintained version of the original OpenAI's Gym library [15]. The observation space of the mountain car environment consists of two attributes: the position of the car along the x-axis and the car's velocity. Three discrete actions form the action space: accelerating the car in both directions and not accelerating.

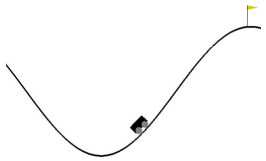


Fig. 5: Mountain car environment.

Eight Gaussian receptive fields are used to encode each one of the attributes of the environment. A multiplying factor is also applied for both attributes (after centering the x-axis position attribute at zero) to improve accuracy for the initial values when the car is at the bottom of the valley with little changes in position. The spike timings of the output neuron dictate the action to be taken. The goal of the mountain car is to reach the flag placed on the right hill as quickly as possible. The agent is then penalized with a reward of -1 for each timestep. After

200 timesteps, the episode ends. This variable number of trials constitutes one of the main differences between this environment and the UCI datasets described before. To help the system converge, the evolution strategy running inside the Zynq device will use this reward information about the time duration of the episode and the maximum x-axis position achieved during it. This allows that, at the first evolution steps, individuals that get closer to the flag are selected. A population of just five individuals is used to accelerate the simulations, as shown in Table 1.

7 Experimental Results

7.1 Resource Utilization

The resource utilization presented in Table 2 covers the implementations for each of the four UCI datasets used for the single IP solution. It can be observed that having only one SNN in the system, the resource occupation is relatively small, with up to 43% of the LUTs used in the worst-case scenario, corresponding to the Wine dataset. This value decreases when implementing smaller networks, such as the one used for the Iris dataset, containing only 17 neurons per layer. In this case, it utilizes 25% of the available LUTs.

Table 2: Resource Utilization with a single SNN (ZYNQ XC7Z020 MPSoC)

Dataset	**LUTs**	**FFs**	**BRAMs**	**DSPs**
Iris	25%	18%	19%	15%
Breast Cancer Dat.	35%	26%	23%	22%
Pima Indian Diabetes	33%	25%	23%	20%
Wine	43%	29%	24%	30%

Table 3: Resource Utilization with two SNNs (ZYNQ XC7Z020 MPSoC)

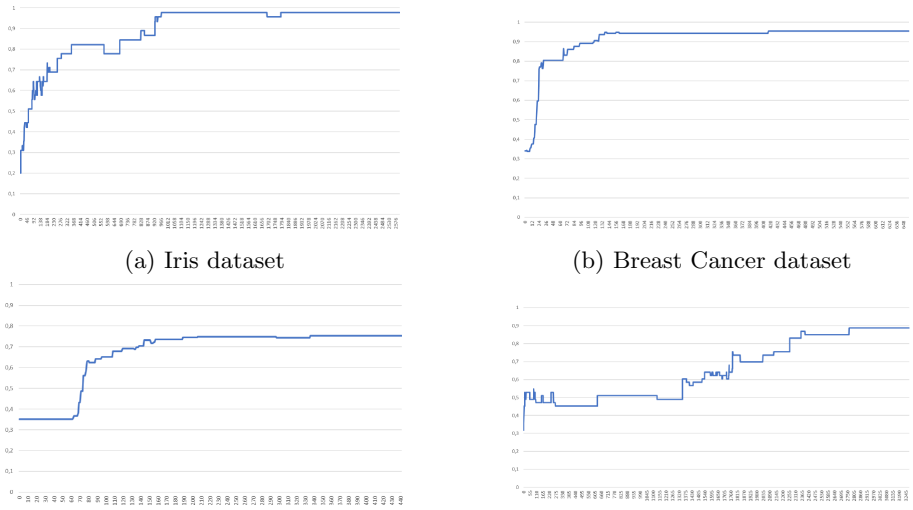
Dataset	**LUTs**	**FFs**	**BRAMs**	**DSPs**
Iris	51%	37%	37%	31%
Breast Cancer Dat.	70%	53%	46%	45%
Pima Indian Diabetes	67%	49%	46%	40%
Wine	98%	59%	47%	59%

When scaling the implementation to two SNN IPs, the resource utilization increases almost by the same factor, as seen in Table 3. All the resource utilization values are approximately doubled, going from a 51% LUT occupation in

the Iris implementation to almost a complete occupation in the case of the Wine dataset.

7.2 Accuracy Results

Fig. 6 shows the maximum, minimum, and average accuracy results achieved for the different datasets during training. The maximum classification accuracy rate of the best individual achieved during training for the Iris dataset is 98.1%. Similar behavior is obtained for the rest of the datasets. The X-axis represents the generation number.



(a) Iris dataset

(b) Breast Cancer dataset

(c) Pima Indian Diabetes dataset

(d) Wine dataset

Fig. 6: Accuracy results for the different training sets.

After training, the goal is to see if the trained networks can extrapolate the knowledge acquired for a given dataset and achieve good results for new input data (during inference with the test dataset). A comparison of the performance in testing achieved over the different datasets against other state-of-the-art solutions that have previously addressed the same problems is provided in Table 4. The comparison table includes results from other machine-learning solutions DoB-SNN [17], SWAT [18], SpikeProp [19], SRESN [20], DANNA and NIDA [21], differential evolution (DE) [22] and Cuckoo Search (CS) [23]. The table presents the accuracy achieved for the testing dataset and neuron type for the solutions considered.

Note that the solution presented in this work is the only one found to use the Izhikevich neuron model in a hardware-accelerated implementation for these datasets. Other solutions use neuron models like LIF or other Integrate-and-Fire

derived models. Additionally, most solutions are tested using software implementations rather than leveraging the parallel capabilities of hardware-accelerated platforms. From the information in the table, it can be seen that the accuracy results for this work are comparable to those obtained by other solutions.

Table 4: Results comparison

Dataset	Algorithm	Neuron type	Hardware impl.	Accuracy
Iris	DANNA	I&F	Yes	99.30%
	NIDA	I&F	No	99.30%
	DE	Izhikevich	No	98.33%
	This work	Izhikevich	Yes	97.78%
	DoB-SNN	LIF	No	97.75%
	SRESN	LIF	No	97.01%
	SpikeProp	LIF	No	96.13%
	CS	Izhikevich	No	94.67%
	SWAT	LIF	No	93.88%
Breast Cancer Wisc.	NIDA	I&F	No	98.60%
	DANNA	I&F	Yes	98.10%
	DoB-SNN	LIF	No	97.35%
	SRESN	LIF	No	97.10%
	SpikeProp	LIF	No	97.04%
	This work	Izhikevich	Yes	95.70%
	SWAT	LIF	No	95.66%
Pima Indian Diab.	NIDA	I&F	No	81.00%
	DANNA	I&F	Yes	78.00%
	SpikeProp	LIF	No	77.38%
	DoB-SNN	LIF	No	76.57%
	CS	Izhikevich	No	74.77%
	DE	Izhikevich	No	73.71%
	SWAT	LIF	No	72.11%
	This work	Izhikevich	Yes	72.06%
	SRESN	LIF	No	70.06%
Wine	NIDA	I&F	No	99,4%
	DANNA	I&F	Yes	97.2%
	CS	Izhikevich	No	90.78%
	This work	Izhikevich	Yes	88.679%
	DE	Izhikevich	No	87.44%

Regarding the mountain car problem, L. Custode and G. Iacca [24] propose evolutionary solutions based on orthogonal and oblique Decision Trees (DT) to solve different reinforcement learning problems. By also computing ten episodes per individual and calculating their mean score, they obtain a maximum score value of 101.72 timesteps for the orthogonal DT method and 106.02 timesteps for

the oblique DT method. For the same problem, Z. Xiao [25] obtains a maximum average score of 102.61 timesteps using a closed-form policy method.

The solution presented in this work solved the mountain car problem with an average episode duration time of 97.3 timesteps, outperforming previous solutions. Three of the five individuals in the original randomly generated population were able to converge and solve this problem in less than 100 timesteps (on average), taking less than 500 generations to do it. This also validates the design for real-time reinforcement learning problems and demonstrates the ability to reliably obtain optimal individuals within a few generations.

8 Conclusions and Future Work

The SNN-based SoC presented in this work has been successfully tested on both supervised learning and reinforced learning problems, achieving positive results for accuracy rates and performance in each case. The results are comparable to other state-of-the-art solutions for the four UCI datasets. For the reinforcement learning test conducted, the results show slightly better performance than other ANN solutions. The tests demonstrate the proposed system's adaptability, flexibility, and real-time capabilities, providing a solid foundation for future development and research. This opens up a wide range of possibilities to improve the system and extend its applications, including the integration of fault models in the neurons to show the fault-tolerance of the proposed integrated SoC, expanding the evolution strategy capabilities by incorporating crossover or more complex selection and mutation algorithms or developing a concurrent solution using several Zynq devices to enable faster computing targeting more complex tasks.

References

1. C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.
2. A. Lines, P. Joshi, R. Liu, S. McCoy, J. Tse, Y.-H. Weng, and M. Davies, "Loihi asynchronous neuromorphic research chip," in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2018, pp. 32–33.
3. S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.
4. C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, p. 2385, 2018.
5. A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.
6. E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

7. R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.
8. F. Ponulak and A. Kasinski, "Introduction to spiking neural networks: Information processing, learning and applications." *Acta neurobiologiae experimentalis*, vol. 71, no. 4, pp. 409–433, 2011.
9. S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, no. 1-4, pp. 17–37, 2002.
10. Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Networks*, vol. 43, pp. 99–113, 2013.
11. F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural computation*, vol. 22, no. 2, pp. 467–510, 2010.
12. I. Sporea and A. Grüning, "Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 25, no. 2, pp. 473–509, 2013.
13. A. Belatreche, L. P. Maguire, M. McGinnity, and Q. X. Wu, "An evolutionary strategy for supervised training of biologically plausible neural networks," in *The sixth international conference on computational intelligence and natural computing*, 2003, pp. 1524–1527.
14. F. G. Sanchez and J. Nunez-Yanez, "Energy proportional streaming spiking neural network in a reconfigurable system," *Microprocessors and Microsystems*, vol. 53, pp. 57–67, 2017.
15. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.
16. A. Asuncion and D. Newman, "Uci machine learning repository," 2007.
17. V. Saranirad, T. M. McGinnity, S. Dora, and D. Coyle, "Dob-snn: A new neuron assembly-inspired spiking neural network for pattern classification," in *2021 International Joint Conference on Neural Networks*, 2021, pp. 1–6.
18. J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, "Swat: A spiking neural network training algorithm for classification problems," *IEEE Transactions on neural networks*, vol. 21, no. 11, pp. 1817–1830, 2010.
19. S. M. Bohte, J. N. Kok, and J. A. La Poutré, "Spikeprop: backpropagation for networks of spiking neurons." in *ESANN*, vol. 48, 2000, pp. 419–424.
20. S. Dora, K. Subramanian, S. Suresh, and N. Sundararajan, "Development of a self-regulating evolving spiking neural network for classification problem," *Neurocomputing*, vol. 171, pp. 1216–1229, 2016.
21. C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 145–154.
22. R. Vazquez, "Izhikevich neuron model and its application in pattern recognition," *Australian Journal of Intelligent Information Processing Systems*, vol. 11, no. 1, pp. 35–40, 2010.
23. R. A. Vazquez, "Training spiking neural models using cuckoo search algorithm," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 679–686.
24. L. L. Custode and G. Iacca, "Evolutionary learning of interpretable decision trees," *IEEE Access*, 2023.
25. Z. XIAO, *Reinforcement Learning: Theory and Python Implementation*. Springer Verlag, Singapor, 2022.