# Comparing Metaheuristic Algorithms on the Training Process of Spiking Neural Networks

**Andrés Espinal, Martín Carpio, Manuel Ornelas, Héctor Puga, Patricia Melin and Marco Sotelo-Figueroa**

**Abstract** Spiking Neural Networks are considered as the third generation of Artificial Neural Networks. In these networks, spiking neurons receive/send the information by timing of events (spikes) instead by the spike rate; as their predecessors do. Spikeprop algorithm, based on gradient descent, was developed as learning rule for training SNNs to solve pattern recognition problems; however this algorithm trends to be trapped in local minima and has several limitations. For dealing with the supervised learning on Spiking Neural Networks without the drawbacks of Spikeprop, several metaheuristics such as: Evolutionary Strategy, Particle Swarm Optimization, have been used to tune the neural parameters. This work compares the performance and the impact of some metaheuristics used for training spiking neural networks.

A. Espinal · M. Carpio · M. Ornelas · H. Puga (✉) · M. Sotelo-Figueroa
Instituto Tecnológico de León, León Gto, Mexico
e-mail: pugahector@yaho.com

A. Espinal
e-mail: andres.espinal@itleon.edu.mx

M. Carpio
e-mail: jmcarpio61@hotmail.com

M. Ornelas
e-mail: mornelas67@yahoo.com.mx

M. Sotelo-Figueroa
e-mail: marco.sotelo@itleon.edu.mx

P. Melin
Instituto Tecnológico Tijuana, Baja California, Mexico
e-mail: pmelin@tectijuana.edu.mx

O. Castillo et al. (eds.), *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, Studies in Computational Intelligence 547,
DOI: 10.1007/978-3-319-05170-3_27, 

## 1 Introduction

Spiking Neural Networks (SNNs) are considered as the third generation of Artificial Neural Networks (ANNs) [1]. These networks are formed by spiking neurons; which deal with information encoded in timing of events (spikes), instead by the spike rate as their predecessors do. It have been proved that spiking neural networks are computationally more stronger than sigmoid neural networks [2]. In fact, there is evidence that fewer spiking neurons are required for solving some functions than neurons of previous generations [1].

As their predecessors, SNNs have been applied for dealing with pattern recognition problems. Spikeprop algorithm is a learning rule for training SNNs, which is based on gradient descent [3]; it is capable of training SNNs to solve complex classification problems. However, this algorithm trends to be trapped in local minima and it has several limitations i.e. it does not allow both, positive and negative synaptic weights. The following list explains some drawbacks about using Spikeprop on the supervised training of SNNs [4]:

- The convergence of the algorithm is vulnerable to being caught in local minima.
- The convergence of the algorithm is not guaranteed as it depends on fine tuning of several parameters before start of the algorithm.
- The structure of the synapse which consists of a fixed number of sub-connections, each of which has a fixed synaptic delay, leads to a drastic increase in the number of connecting synapses and therefore the number of adjustable weights.
- The huge number of synaptic connections makes it difficult for such algorithms to scale up when processing high-dimensional dataset is considered.
- Also having a fixed number of sub-connections with fixed delay vales is not necessity and yields a lot of redundant connections.
- The algorithm entails the problem of 'silent neurons'; i.e. if the outputs do not fire then the algorithm halts, since no error can be calculated.

For overcoming the drawbacks of Spikeprop on the supervised learning of SNNs, some works have proposed to use metaheuristic for tuning the neural parameters (weights and delays) [4–7]. Two similar proposals of the state of the art for training SNNs by using metaheuristics are compared in this work. Both methodologies are tested with classical test benchmarks: XOR Logic Gate and Iris plant dataset (UCI Machine Learning Repository), with different architectures over several experiments.

This chapter is organized as follows: Sect. 2 gives fundamentals for simulating SNNs, including neural network structure, codifier and spiking neuron model. In Sect. 3, the methodology used for training SNNs is explained. The experimental design and results are showed in Sect. 4. Finally, in Sect. 5 conclusions about the work and future work is presented.

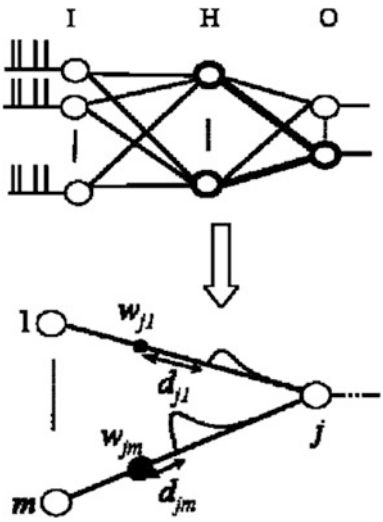


**Fig. 1** *Top* Spiking neural network with three layers: input, hidden and output. *Bottom* Synapse connection (weights and delays) between presynaptic neurons and the postsynaptic neuron *j* (figure taken from [6])

## 2 Spiking Neural Networks

A neural network can be defined as an interconnection of neurons, such that neuron outputs are connected, through weights, to all other neurons including themselves; both lag-free and delay connections are allowed [8]. There are several models or topologies of ANNs, which are defined around three aspects: computing nodes, communication links and message types [9].

For this work were used feed-forward SNNs. They were formed by 3 layers: input, hidden and output (see Fig. 1), their topology was set as follows: the computing nodes (neuron models) are spiking neurons, specifically neurons of the Spike Response Model (SRM) [10]; this model is explained later. The communication links (synaptic connections, each synaptic connection is defined by a synaptic weight and a synaptic delay) were set to form a fully interlayer connected neural network; where all possible interlayer connections are present and the network contains no intralayer (including self-connections), or supralayer connections [11], this kind of pattern connection is also know it as fully connected neural network. The message types are based on the time-to-first-spike encoding scheme; it means that each neuron can produce at most one spike in all the simulation time for each stimuli.

### *2.1 Gaussian Receptive Fields*

The traditional form of patterns as multidimensional raw data, which consist of real values can't be used to feed a SNNs in a simulation process. The patterns need to be transformed into temporal patterns (a set of events in time or spikes as known as spike trains) before being processed by the SNN.

In [12], was proposed an encoding scheme to generate firing times from real values; the Gaussian Receptive Fields (GRFs). This scheme enables the representation of continuously valued input variables by a population of neurons with graded and overlapping sensitivity profiles, such as Gaussian activation functions (the receptive fields). To encode values into a temporal pattern, it is sufficient to associate highly stimulated neurons with early firing times and less stimulated neurons with later (or no) firing times. Each input variable is encoded independently, it means that each input dimension is encoded by an array of one-dimensional receptive fields; the GRFs constitute a biologically plausible and well studied method for representing real-valued parameters [3].

In [13] is given a detailed definition about the construction and use of the GRFs. Each input datum is fed to all the conversion neurons covering the whole data range. For a range $[n_{min},\ n_{max}]$ of a variable $n$, $m$ neurons were used with GRF. For the $i$-th neuron coding for variable $n$, the center of the Gaussian function is set to $C_i$ according to Eq. (1).

$$C_i = n_{min} + \frac{2i-3}{w} \times \frac{n_{max} - n_{min}}{m-2}, \quad m > 2 \tag{1}$$

The width $w$ of each Gaussian function is set according to Eq. 2. Where $1 \leq \gamma \leq 2$ is a coefficient of multiplication which is inversely proportional to the width of the Gaussian functions.

$$w = \frac{n_{max} - n_{min}}{\gamma(m-2)} \tag{2}$$

Using the centers of each neuron/Gaussian function and their width, the magnitude of firing $f(x)$ for each sample points of the input $x$ is calculated using Eq. 3.

$$f_i(x) = e^{-\frac{(x-c_i)^2}{2w^2}} \tag{3}$$

The magnitude values are then converted to time delay values by associating the highest magnitude value to a value close to zero milliseconds and neurons with lower firing magnitude are associated with a time delay value close to the limit maximum of time of simulation. Conversion from magnitude to time delay values is done using Eq. (4). Where $\tau$ is the total time of simulation.

$$T_i = (1 - f_i) \times \tau, \quad i = 1, 2, \ldots, m \tag{4}$$

Time delay values greater than some value $\tau_{threshold} < \tau$ are coded no to fire as they are consider being insufficiently excited. Therefore, neurons with time delay value between zero and $\tau_{threshold}$ milliseconds carry all the encoded information of the input data.

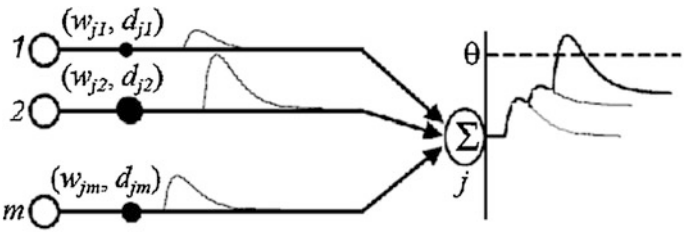


**Fig. 2** Incoming spikes transformed into postsynaptic potentials, they are delayed and weighted by the synapse and finally they are integrated by the postsynaptic neuron (figure taken from [5])

## 2.2 Spike Response Model

The SRM [10, 14] is an approximation of the dynamics of the integrate-and-fire neurclose to the limit maximum of time of simulation.ons. The neuron status is updated through a linear summation of the postsynaptic potentials resulting from the impinging spike trains at the connecting synapses. A neuron fires whenever its accumulated potential reaches a threshold from below (see Fig. 2) [6].

Due to spiking neurons in this work use the time-to-first-spike encoding scheme for sending/receiving messages, a reduced version of the SRM is implemented; this has been used in [3–7, 12].

The reduced SRM is defined according [6] as follows. Let us consider that a neuron $j$ has a set $\Gamma_j$ of immediate predecessors called presynaptic neurons and receives a set of spikes with firing times $t_i, i \in \Gamma_j$. Neurons fire when their state variable $x(t)$, called membrane potential, reaches a certain threshold $\Theta$. The internal state of a neurons is determined by (5), where $w_{ji}$ is the synaptic weight to modulate $y_i(t)$, which is the unweighted postsynaptic potential of a single spike coming from neuron $i$ and impinging on neuron $j$.

$$x_j(t) = \sum_{i \in \Gamma_j} w_{ji} y_i(t) \tag{5}$$

The unweighted contribution $y_i(t)$ is given by Eq. (6), where $\varepsilon(t)$ is a function that describes the form of the postsynaptic potential.

$$y_i(t) = \varepsilon\left(t - t_i - d_{ji}\right) \tag{6}$$

The form of the postsynaptic potential is given by Eq. (7), and it requires the next parameters: $t$ is the current time, $t_i$ is the firing time of the presynaptic neuron $i$ and $d_{ji}$ is the associated synaptic delay. Finally the function has a $\tau$ parameter, it is the membrane potential time constant defining the decay time of the postsynaptic potential.

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{1-\frac{t}{\tau}} & if\, f > 0 \\ 0 & else \end{cases} \tag{7}$$

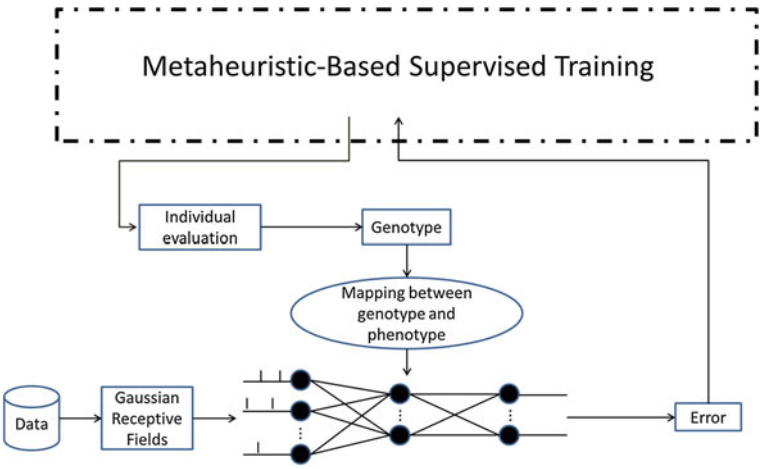


**Fig. 3** Generic scheme for training SNNs with metaheuristic algorithms

The firing time $t_j$ of neuron $j$ is determined as the first time the state variable crosses the threshold from below. The threshold $\Theta$ and $\tau$ are constants and equal for all neurons in the network.

## 3 Metaheuristic-Based Supervised Learning

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [15]. In this case, the learning is driven by some metaheuristic algorithm (see Fig. 3 the free parameters of the SNN are all weights and delays of each synapse.

Several works have used metaheuristics for training SNN [4–7]; each of them defines a particular methodology, next is presented the methodology followed based on similar aspects of the works revised for this work.

In Metaheuristic-Based Supervised Learning, each individual contains all the free parameters of a previously structured SNN. Every individual is evaluated by means of a fitness function, the first step of the fitness function makes a mapping process; this sets the individuals parameter as weights and delays in the SNN. The second step of the fitness function uses the batch training as learning protocol, where all patterns are presented to the network before the learning takes place [16]. The third step of the fitness function is calculated an error (to be minimized) according Eq. (8) (equation taken from [4]); where $T$ are all training patterns, $O$ are all output spiking neurons, $t_O^a(t)$ is the current timing output of the SNN and $t_O^t(t)$ is the desired timing output. The error calculated in the fitness function

determines the fitness value of each individual and drives the supervised learning based on metaheuristic algorithms.

$$E = \sum_{t}^{T} \sum_{o}^{O} \left( t_0^a(t) - t_o^t(t) \right)^2 \tag{8}$$

Next are presented two metaheuristic used for training SNNs in the state of the art.

## 3.1 Evolutionary Strategy

The Evolutionary Strategies (ES) [17], deal natively with problems in real domain. In [4] was designed a Self-Adaptive ES for training SNNs, in this ES each population member consists of $n$-dimensional vectors, where $n$ is the total number of tuneable network parameters within input, hidden and output layer. The population at any given generation $g$ is denoted as $P(g)$. Each individual is taken as a pair of real-valued vectors, $(x_i, \eta_i)$, where $x_i$'s are objective variables representing the synaptic free parameters, and $\eta_i$'s are standard deviations for mutations. Each individual generates a single offspring $(x'_i, \eta'_i)$, where each variable $x'_i(j)$ of the offspring can be defined by either Eq. 9 (local search) or Eq. 10 (global search) and the standard deviation is defined by Eq. (11).

$$x'_i(j) = x_i(j) + \eta_i(j) N_j(0,1) \tag{9}$$

$$x'_i(j) = x_i(j) + \eta_i(j) \delta_j \tag{10}$$

$$\eta'_i(j) = \eta_i(j) \exp\left( \tau' N(0,1) + \tau N_j(0,1) \right) \tag{11}$$

where:

- $N(0,1)$ denotes a normally distributed one dimensional random number with $\mu = 0$ and $\sigma = 1$.
- $N_j(0,1)$ indicates that the random number is generated anew for each value of $j$.
- $\delta_j$ is a Cauchy random variable, and it is generated anew for each value of $j$ (Scale $= 1$).
- Factor $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$
- Factor $\tau' = \frac{1}{\sqrt{2n}}$

The Self-adaptive ES is presented in the Algorithm 1.

**Algorithm 1** Self-adaptive ES 1

1: Generate the initial population of $\mu$ individuals.
2: Evaluate the fitness score for each individual $(x_i, \eta_i)$, $i = 1, \ldots, \mu$ of the population based on $E = \sum_t^T \sum_o^O (t_o^a(t) - t_o^t(t))^2$
3: **while** the maximum iteration is not reached **do**
4: Each parent $(x_i, \eta_i)$ generates a single offspring $(x_i', \eta_i')$
5: Calculate the fitness of each offspring $(x_i', \eta_i')$, $i = 1, \ldots, \mu$
6: Generate a new population $P(g)$ using tournament selection and elitism to keep track of the best individual at each generation
7: **end while**

## 3.2 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) [18] is an algorithm originally used for continuous optimization problems, it is inspired in the behavior of flocks of birds or schools of fish. A version of PSO called Cooperative PSO (CPSO) has been used in [7] for training SNNs. In this metaheuristic, each individual is defined by the vector velocity $v_i$ and the vector position $x_i$.

The vector $v_i$ is updated by Eq. (12) where $\rho_1$ and $\rho_2$ are acceleration coefficients. $\chi$ is a constriction factor, it can be defined according Eq. (14). $\gamma_1$ and $\gamma_2$ are two independent uniform numbers randomly distributed in the range [0,1]. And $\omega$ is the inertia weight, which controls the impact of the previous velocity; it is defined by Eq. (13) where $\omega_0$ is the maximum of $\omega$, $\omega_0 - \omega_1$ is the minimum of $\omega$, $t$ is the current iteration and $T_{max}$ is the maximal iteration of PSO.

The vector $x_i$ is updated by Eq. (12).

$$v_{id} = \chi(\omega v_{id} + \rho_1 \gamma_1 (p_{id} - x_{id})) \tag{12}$$

$$\omega = \omega_0 - \omega_1 \frac{t}{T_{max}} \tag{13}$$

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}; \varphi = \rho_1 + \rho_2, \varphi > 4 \tag{14}$$

$$x_{id} = x_{id} + v_{id} \tag{15}$$

The CPSO is presented in the Algorithm 2.

**Algorithm 2** CPSO-based supervised learning

1: Generate the initial swarm.
2: **for all** particle $i$ **do**
3: $x_{id} \leftarrow rand(x^{min}, x^{max})$

(continued)

(continued)

**Algorithm 2** CPSO-based supervised learning

4: $v_{id} \leftarrow rand(v^{min}, v^{max})$
5: Its past best solution can be set to the same as $x_i$ initially.
6: **end for**
7: **while** the maximum iteration is not reached **do**
8: Compute the fitness of each particle for its current position
9: Compute for each input pattern by Spiking neural network, and obtain the spike times of the output neurons
10: Calculate the square error usign $E = \sum_t^T \sum_o^O (t_o^a(t) - t_o^t(t))^2$ as the fitness of the particle
11: Find the global best particle g with the minimal fitness from the swarm of the current iteration
12: $v_{id} \leftarrow \chi\left(\omega v_{id} + p_1\gamma_1(p_{id} - x_{id}) + p_2\gamma_2\left(p_{gd} - x_{id}\right)\right)$
13: $x_{id} \leftarrow x_{id} + v_{id}$
14: **end while**

## 4 Experiments and Results

Classical benchmarks of pattern recognition were used for experimentation: XOR Logic Gate and Iris Plant dataset. For each benchmark, 5 SNNs with different configurations on their layers were used; each of them was trained (using both metaheuristics) 35 times. For each benchmark, the training by using ES and CPSO of each SNN configuration is contrasted using the fitness values by calculating the minimum, the maximum, median, mean and standard deviation.

The common parameters of the metaheuristics through all experiments and trainings are: 30 individuals and 15,000 function calls.

### *4.1 XOR Logic Gate*

For the XOR logic gate, it was not necessary to use GRFs. Inputs values were manually encoded, the encoded inputs and the desired timing output are showed in Table 1.

The values of CPSO parameters were empirically set, they were set as $\rho_1 = 0.8$ and $\rho_2 = 3.3$. The results for the training of each configuration are showed in Table 2. For this benchmark the testing phase was not necessary, due that any fitness value greater than 0 produces a poor performance of the SNN. Taking the median value as representative error of training for each metaheuristic over all configurations, can be inferred that ES outperforms to CPSO for the training phase in this problem.

**Table 1** Configuration used for solving XOR

| Original XOR | | | | Encoded XOR | |
|---|---|---|---|---|---|
| $x_1$ | $x_2$ | y | $x'_1$ | $x'_2$ | Output firing time (ms) |
| 0 | 0 | 0 | 0 | 0 | 20 |
| 0 | 1 | 1 | 0 | 6 | 10 |
| 1 | 0 | 1 | 6 | 0 | 10 |
| 1 | 1 | 0 | 6 | 6 | 20 |

**Table 2** Results of fitness values for training SNNs by using ES and CPSO to solve XOR Logic Gate

| Network | ES Optimization | | | | CPSO Optimization | | | |
|---|---|---|---|---|---|---|---|---|
| | Minimum | Median | Maximum | $\mu \pm \sigma$ | Minimum | Median | Maximum | $\mu \pm \sigma$ |
| 2-3-1 | 0.0 | 8.0 | 51.0 | 15.4 ± 16.8 | 1.0 | 22.0 | 52.0 | 25.8 ± 15.6 |
| 2-5-1 | 0.0 | 1.0 | 50.0 | 5.7 ± 9.9 | 0.0 | 5.0 | 43.0 | 10.3 ± 12.0 |
| 2-7-1 | 0.0 | 0.0 | 18.0 | 1.3 ± 3.3 | 0.0 | 2.0 | 24.0 | 5.9 ± 7.0 |

**Table 3** Results of fitness values for training SNNs by using ES and CPSO to solve Iris plant dataset (1st fold)

| Network | ES Optimization (Fitness) | | | | CPSO optimization (Fitness) | | | |
|---|---|---|---|---|---|---|---|---|
| | Minimum | Median | Maximum | $\mu \pm \sigma$ | Minimum | Median | Maximum | $\mu \pm \sigma$ |
| 16-1-1 | 35.0 | 99.0 | 218.0 | 101.4 ± 40.1 | 83.0 | 182.0 | 267.0 | 181.2 ± 47.8 |
| 16-5-1 | 10.0 | 67.0 | 167.0 | 69.7 ± 30.1 | 33.0 | 117.0 | 192.0 | 121.2 ± 40.6 |
| 16-10-1 | 16.0 | 62.0 | 207.0 | 64.9 ± 34.0 | 55.0 | 111.0 | 202.0 | 115.6 ± 35.5 |
| 16-15-1 | 20.0 | 64.0 | 123.0 | 65.9 ± 23.1 | 13.0 | 86.0 | 215.0 | 97.4 ± 42.8 |
| 16-20-1 | 29.0 | 66.0 | 111.0 | 66.3 ± 23.5 | 16.0 | 88.0 | 204.0 | 92.6 ± 41.4 |

## *4.2 Iris Plant Dataset*

The Iris plant dataset contains 3 classes of which 2 are not linearly separable, each class is formed by 50 patterns each of them described by 4 features. For this benchmark, each feature was encoded by GRFs using 4 encoding. The desired timing outputs for setosa, versicolor and virginica classes are respectively 6,10 and 14 ms. The dataset was divided into 2 equal parts (2-Folds Cross Validation). Then 2 tests were performed.

The values of CPSO parameters were empirically set, they were set as $\rho_1 = 2.1$ and $\rho_2 = 2.1$. For the first test instance Tables 3 and 4 show the optimization phase and classification phase respectively. And for the second test instance Tables 5 and 6 show the optimization phase and classification phase respectively. The classification phase was performed by using the configuration given by the

**Table 4** Results of performance of SNN trained with ES and CPSO (1st fold)

| Network | Classification (ES Training) | | | | Classification (CPSO Training) | | | |
|---|---|---|---|---|---|---|---|---|
| | Training | | Testing | | Training | | Testing | |
| | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) |
| 16-1-1 | 4.0 | 69.3 | 13.3 | 81.3 | 9.3 | 22.7 | 12.0 | 34.7 |
| 16-5-1 | 41.3 | 85.3 | 38.7 | 90.7 | 20.0 | 68.0 | 21.3 | 80.0 |
| 16-10-1 | 4.0 | 89.3 | 8.0 | 98.7 | 1.3 | 32.0 | 2.7 | 37.3 |
| 16-15-1 | 41.3 | 92.0 | 41.3 | 97.3 | 36.0 | 66.7 | 32.0 | 82.7 |
| 16-20-1 | 54.7 | 86.7 | 58.7 | 86.7 | 34.7 | 78.7 | 33.3 | 89.3 |

**Table 5** Results of fitness values for training SNNs by using ES and CPSO to solve Iris plant dataset (2nd fold)

| Network | ES Optimization (Fitness) | | | | CPSO Optimization (Fitness) | | | |
|---|---|---|---|---|---|---|---|---|
| | Minimum | Median | Maximum | $\mu \pm \sigma$ | Minimum | Median | Maximum | $\mu \pm \sigma$ |
| 16-1-1 | 16.0 | 67.0 | 218.0 | $65.5 \pm 16.3$ | 69.0 | 133.0 | 312.0 | $134.7 \pm 32.2$ |
| 16-5-1 | 10.0 | 39.0 | 167.0 | $37.3 \pm 11.4$ | 33.0 | 95.0 | 220.0 | $86.7 \pm 23.1$ |
| 16-10-1 | 16.0 | 32.0 | 207.0 | $34.5 \pm 11.0$ | 37.0 | 83.0 | 202.0 | $79.3 \pm 20.2$ |
| 16-15-1 | 7.0 | 39.0 | 123.0 | $37.6 \pm 11.3$ | 13.0 | 62.0 | 215.0 | $62.3 \pm 18.4$ |
| 16-20-1 | 16.0 | 39.0 | 111.0 | $38.0 \pm 8.8$ | 16.0 | 52.0 | 204.0 | $53.5 \pm 13.8$ |

**Table 6** Results of performance of SNN trained with ES and CPSO (2nd fold)

| Network | Classification (ES Training) | | | | Classification (CPSO Training) | | | |
|---|---|---|---|---|---|---|---|---|
| | Training | | Testing | | Training | | Testing | |
| | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) | Minimum (%) | Maximum (%) |
| 16-1-1 | 4.0 | 89.3 | 13.3 | 80.0 | 1.3 | 53.3 | 4.0 | 52.0 |
| 16-5-1 | 41.3 | 85.3 | 38.7 | 90.7 | 33.3 | 68.0 | 36.0 | 80.0 |
| 16-10-1 | 4.0 | 89.3 | 8.0 | 98.7 | 1.3 | 78.7 | 2.7 | 68.0 |
| 16-15-1 | 41.3 | 90.7 | 41.3 | 85.3 | 36.0 | 66.7 | 32.0 | 82.7 |
| 16-20-1 | 54.7 | 98.7 | 58.7 | 93.3 | 34.7 | 78.7 | 33.3 | 89.3 |

minimum and maximum fitness values on the training phase, the best performances correspond to trainings with low fitness and worst performances correspond to trainings with high fitness. Same as the XOR logic gate, taking the median value as representative error of training for each metaheuristic over all configurations, can be inferred that ES outperforms to CPSO for the training phase in this problem. In general the best performances on classification tasks, for both: known patterns and unknown patters, can be observed in the trainings made by ES.

## 5 Conclusions and Future Work

This work compares two metaheuristics on the training of SNNs. The few ES's parameters made easier to implement. It as training algorithm than CPSO. Moreover, in general ES gets better fitness values than CPSO and the SNNs trained with ES get better classification performance too.

The results obtained show evidence that there is a relationship between the neural architecture and the capability of the metaheuristic to train it. Due that good results in phases of training and testing can be achieved by different architectures, but some neural configurations are easier to train and show more stability through several experiments. Even when a metaheuristic can achieved low fitness values, it doesn't ensure the good performance of the SNN for classifying unseen data; this is an important aspect that needs to be analyzed when using metaheuristic-based supervised learning on SNNs.

As future work, authors propose try to improve the CPSO performance on the training of SNNs by tuning its parameters. It is interesting to use other metaheuristic algorithms as learning rules to analyze their behavior. Finally it is necessary to use statistical tests for a best comparison of these algorithms in this task.

**Acknowledgments** Authors thanks the support received from Consejo Nacional de Ciencia y Tecnologia (CONACyT).The authors want to thank to *Instituto Tecnológico de León* (ITL) for the support to this research. Additionally they want to aknowledge the generous support from the *Mexican National Council for Science and Technology* (CONACyT) for this research project.

## References

1. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural Networks **10**(9), 1659–1671 (1997)
2. Maass, W.: Noisy Spiking Neurons with Temporal Coding Have More Computational Power Than Sigmoidal Neurons, pp. 211–217. MIT Press, Cambridge (1996)
3. Bohte, S.M., Kok, J.N., LaPoutre, H.: Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing **48**, 17–37 (2002)
4. Belatreche, A.: Biologically Inspired Neural Networks: Models, Learning, and Applications. VDM Verlag Dr. Müller, Saarbrücken (2010)
5. Belatreche, A., Maguire, L.P., McGinnity, M., Wu, Q.X.: An evolutionary strategy for supervised training of biologically plausible neural networks. In: The Sixth International Conference on Computational Intelligence and Natural Computing (CINC), Proceedings of the 7th Joint Conference on Information Sciences. pp. 1524–1527 (2003)
6. Belatreche, A., Maguire, L.P., McGinnity, T.M.: Advances in design and application of spiking neural networks. Soft. Comput. **11**(3), 239–248 (2007)
7. Shen, H., Liu, N., Li, X., Wang, Q.: A cooperative method for supervised learning in spiking neural networks. In: CSCWD. pp. 22–26. IEEE (2010)
8. Zurada, J.M.: Introduction to Artificial Neural Systems. West (1992)
9. Judd, J.S.: Neural Network Design and the Complexity of Learning. Neural Network Modeling and Connectionism Series, Massachusetts Institute Technol (1990)

10. Gerstner, W.: Time structure of the activity in neural network models. Phys. Rev. E **51**(1), 738–758 (1995)
11. Elizondo, D., Fiesler, E.: A survey of partially connected neural networks. Int. J. Neural Syst. **8**(5–6), 535–558 (1997)
12. Bohte, S.M., La Poutre, H., Kok, J.N.: Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. Neural Networks IEEE Trans. **13**, 426–435 (2002)
13. Johnson, C., Roychowdhury, S., Venayagamoorthy, G.K.: A reversibility analysis of encoding methods for spiking neural networks. In: IJCNN. pp. 1802–1809 (2011)
14. Gerstner, W., Kistler, W.: Spiking Neuron Models: Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge (2002)
15. Haykin, S.: Neural Networks: Comprehensive Foundation. Prentice Hall (1999)
16. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, New York (2012)
17. Rechenberg, I.: Evolutions Strategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog (1973)
18. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. IEEE Int. Conf. Neural Netw. **4**, 1942–1948 (1995)