Neurocomputing 148 (2015) 187–197

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Tuning the parameters of an integrate and fire neuron via a genetic algorithm for solving pattern recognition problems

Aleister Cachón [*], Roberto A. Vázquez [**]

Intelligent Systems Group, Universidad La Salle, Benjamín Franklin No. 47, Col. Hipódromo Condesa, 06140, D.F., Mexico

ARTICLE INFO

Article history:
Received 7 April 2012
Received in revised form
6 November 2012
Accepted 13 November 2012
Available online 30 July 2014

Keywords:
Spiking neuron
Genetic algorithm
Pattern recognition
Integrate and fire

ABSTRACT

Recently, it has been proven that spiking neurons can be used for some pattern recognition problems. Nonetheless, the spiking neurons models have many parameters that have to be manually adjusted in order to achieve the desired behavior. This paper has the purpose of showing an optimization method for one such model, the Integrate & Fire spiking model (I&F). A genetic algorithm (GA) is proposed to automatically adjust the parameters, removing the need of manual tuning and increasing efficiency. Initial experimentation is done by tuning the I&F model parameters by hand, to confirm the importance and relevance of determining the best parameter values. The GA is then used to automatically tune different parameter combinations of the pattern recognition model, which uses the I&F neuron as core, to determine which parameters are worth including in the GA. The proposed method was tested with five different datasets, where no change was required to apply the model to each. Very good results were achieved in all test cases, but experiments where parameters of the neuron model were included converged faster.

1. Introduction

In the last few years, neural networks based on spiking neuron models have gained importance. This has partly because their behavior is closest to the one of a biological neuron, thus making simulations more realistic [1]. Spiking neuron models have been broadly used in computational neuroscience [2] such as auditory processing [3], visual processing [4,5], robotics [6,7], and brain region modeling [8].

In contrast with perceptron based networks, the result is not a coded value within the output of the network in a single propagation cycle; it is obtained by leaving the model reacting to and processing the input during a time T, so that it generates a *spike train*. The spike train is then interpreted according to its characteristics, with different ways to code and understand the results [9]; we can then relate the outputs to different classes as required by the pattern recognition problem.

Spiking neural models have been applied in pattern recognition problems. During the training phase, they adjust their synaptic weights using several techniques such as the back-propagation algorithm [10,11]. Evolutionary Computation methods have been also applied during training. In [12], the authors described how a genetic algorithm can be used during the learning process of a spiking neural network; however, the model was not applied to perform a pattern recognition problem. In [13], the authors show how a spiking neural network can be trained by a quantum PSO and then applied to a string pattern recognition problem. In [14] the authors trained a spiking neural network using a PSO algorithm, and then it was applied in three pattern recognition problems.

In [9,15,16], the authors shown how only one spiking neuron such as Leaky-Integrate-and-Fire and Izhikevich models [17,18] can be applied to solve different linear and non-linear pattern recognition problems.

For example in [9], an I&F model is fed different currents which correspond to different input patterns. Each input pattern belongs to a different class; this shows the neuron's ability to generate different outputs which can be interpreted as different classes. We have to remark that this method uses a single neuron; this may lead to think that non-linear class separation is not possible, nonetheless it has been proven [9] that non-linear separation is within the scope of the I&F model's capabilities.

In spite of the usefulness of the method, the parameters for the neuron model had to be manually adjusted taking into account the behavior of the model. By means of experimenting with the parameters, better results can be achieved; but doing so manually can be a tiresome task and even better results could be achieved with an automatic adjustment method.

In this paper, a genetic algorithm is proposed for exploring the parameters of the I&F model, and those of the proposed method, to yet obtain better results. First, different combinations for the parameters are tested; this with the purpose of explaining how the

[*] Corresponding author.
[**] Principal corresponding author.
E-mail addresses: a.cachon@lasallistas.org.mx (A. Cachón), ravem@lasallistas.org.mx (R.A. Vázquez).

http://dx.doi.org/10.1016/j.neucom.2012.11.059
0925-2312/© 2014 Elsevier B.V. All rights reserved.

changes in the impulse train may affect pattern classification. After that, many tests are done in which we apply the genetic algorithm to the model, including/excluding different parameters from it. We may also be able to determine which parameters of the model, if any, should be subjected to the genetic algorithm's domain search and optimization. To determine the performance of the proposed method, five different datasets were used, which vary in complexity and data categories.

2. Problem statement

The paper aims to improve the pattern recognition model presented in [9] by providing automated tuning to its parameters, including those of the spiking neuron model. The improvements will be reflected as ease of use, since no specific tuning for different datasets will be required, and better classification rates.

Given that the model should have an automated adjustment, which may apply to a variety of parameters, a genetic algorithm is used for this purpose.

2.1. Genetic algorithm

A genetic algorithm (GA) is an optimization strategy which performs a heuristic search over the domain of a problem [19]. The GA tries to replicate the process of evolution where *the aptest individual survives*. It utilizes operators for that purpose, such as *elitism*, *crossover*, *inheritance* and *mutation*, simulating evolutionary and genetic processes within a population. The techniques mentioned are not the only ones used in GAs, but belong to the most important. Each of the techniques has different ways to be implemented, ranging from random to orderly strategies.

The objective of the GA is to find a good, if not the best, solution over the course of many generations. The problem is represented as an objective function, called a *fitness function*, which the algorithm intends to optimize. Each individual (candidate solution) in the population (set of individuals) is evaluated by the function, thus obtaining a fitness. The obtained values allow the GA to choose the best individuals in the population and to generate a new population, improving its general aptness. Each cycle where this happens is called a generation. Using a specified criterion, we stop at a last generation. The stop criterion can be, for example, a number of generations, or achieving a certain fitness. From the final population the best individual is chosen as the optimal solution for the problem.

An algorithm for a standard GA is shown here:

```
begin
  maxGenerations := N;
  generation := 1;
  population := InitializePopulation();
  repeat
    EvaluateIndividuals(population);
    SelectFittestIndividuals(population);
    population := ApplyOperators(population);
    generation := generation + 1;
  until generation = maxGenerations;
  solution := BestIndividual(population);
end.
```

2.2. Integrate & fire spiking neuron model

Within the spiking neuron models, the *Integrate* & *Fire* is one of the most popular in computer neuroscience, it is also one of the easiest to implement. This model is so simple that its computational cost is negligible compared to more complex models. Given its simplicity it lacks flexibility, as it can only simulate *Class 1 excitable neurons*, as classified according to the spike train. This type of neuron emits a continuous spike train where the interval between spikes is inversely proportional to the input magnitude, in other words, the greater the input potential, the smaller the spike interval.

The mathematical model is represented by the membrane's potential dynamic:

$$\tau \frac{dv_i}{dt} = -g_{leak}(v_i - E_{leak}) + I(t) \tag{1}$$

where g_{leak} and E_{leak} are the conductance and reversal potential of the leak current, τ is the membrane time constant and $I(t)$ is the current injected into the neuron. Another simpler representation of the model [9], which is used by the pattern recognition method, is shown here:

$$v' = I + a - bv$$
$$\text{if } v \geq v_{threshold} \text{ then } v \leftarrow c \tag{2}$$

where I is the neuron's input current (directly injected into it), a and b are parameters to vary the model's behavior, $v_{threshold}$ is the threshold for the neuron's spike (firing), and c is the reset or steady state voltage. Another parameter which is important once the equation is solved by numerical methods is the initial condition v_0.

2.3. Analysis of the I&F Model's parameters when related to pattern recognition

For the analysis of the parameters, it is important to remember how the I&F model reacts to the input. The I&F neuron generates a spike train of type *Class 1 excitable*, meaning, it emits a constant spike train whose interval between spikes is inversely proportional to the input's magnitude. According to this it would be expected that as the current rises, the quantity of spikes in the train, and thus the firing rate, also will.

A good method to determine different classes is to relate each class to a number (or number range) of spikes an input pattern generates in a given time [9]. The output measuring is done according to the number of spikes within a time T for which the neuron is stimulated. This is basic to justify the importance of the neuron's parameters, as they have a direct effect in the behavior of the spike train. Depending on the values of a, b, c, $v_{threshold}$ and v_0, the behavior of the model and thus the output will vary, modifying the firing rate (FR).

Just one neuron with different values for the parameters a and b is used for the analysis. Variations in parameters c, $v_{threshold}$ and v_0 are left pending, but it is assumed they will affect the input processing in a more uniform way which may not be relevant enough for pattern recognition.

For each of the parameter values' combination, four different input currents generated from different patterns are tested; each represents a different class. The input currents used are: $\{0.4, 1.6, 3, 7\}$ pA. The analysis of the behavior is done experimentally, instead of mathematical analyzing the dynamic of the equation. The purpose of this is to show that the dynamic of the model does not need to be known to utilize the algorithm for choosing the parameters.

In a time T, if the number of spikes $\approx$ zero or T, the spike train is not considered *significant* as it is not giving any information on the output, as they respectively represent the cases where the neuron is *under-stimulated* and *over-stimulated*. In such cases, there's no distinction in the outputs generated by fairly different inputs.

2.3.1. *a fixed to 0.1, b with* $[-0.1, 1.1]$ *range and 0.4 increments*

For this value range (Table 1) it was observed that the lowest of the input currents, 0.4 pA, did not generate a significant output (the number of spikes approximates 0 or T, which is 100), as only when $b = 0.3$ a reaction was present (Fig. 1).

From this initial result it can be theorized that values given to a and b will *affect the input range that will present a significant output* for pattern recognition.

For a negative value of b, the input must be great to generate an output, as observed for 7 pA. We can also observe that the shape of the spikes changes (Fig. 2), having a closer resemblance to a real neuron's output as opposite to positive b values. Nevertheless we must remember that *the shape of the spike does not carry any information*.

Another observed behavior is that for $b > 1$ the neuron is saturated and does not stop generating spikes, and does not have the absolute refractory period which should be found on biological neurons, as mentioned in Section 2.2.

For $b = 0.3$, clear differences in the spike quantity for each input could be observed. This behavior may be the one we would expect for pattern recognition problems, where we aim to have *different input patterns belonging to different classes generating different enough outputs*.

Table 1
Spike quantity for all the inputs and the parameter combination in Section 2.3.1 for $T = 100$ ms.

b	Input currents (pA)			
Value	0.4	1.6044	3	7
−0.1	0	0	0	7
0.3	7	14	16	24
0.7	0	32	33	49
1.1	99	99	99	99

2.3.2. *a with* $[-1, 19]$ *range and 5 increments, b fixed to* -0.1

It seems interesting to examine the possibilities for $b = -0.1$ given that the spike shape is different to that of the other values. A wide range for the values of a is used, with a big increment, expecting to get good results useful for pattern recognition.

In Table 2, we can observe that small a values cause the model to have no output for any of the inputs. The next value, 4, causes significant results only in half of the inputs. From 9 and to the maximum value in the range, significant outputs are generated for all of the inputs.

The chosen b value seems to *confer sturdiness to the model* given that even for high a values and the tested inputs, only the spike number varies (Fig. 3). This sturdiness may prove important in pattern recognition problems as it may make the model *respond to a greater range of inputs*. Reducing this sturdiness would then result in the contrary effect, making the model *respond to a lesser range of inputs*. Both behaviors may be relevant for pattern recognition as they could make the model include important inputs for classification and discard unimportant ones.

In spite of the apparent low importance a may have, if we pay enough attention to the results we can notice that as its value increased, *the difference between outputs corresponding to different inputs decreased greatly*. This may indicate that a is an important

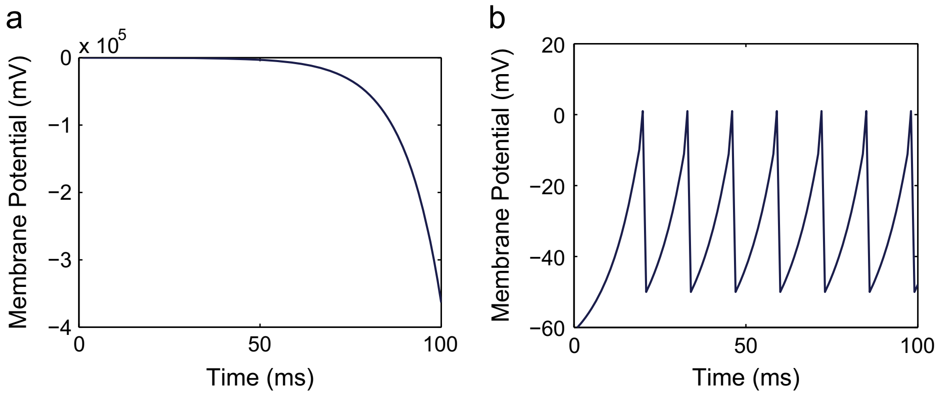

Fig. 1. Graphics of the spike train with $a = 0.1$, $b = -0.1$; for $I = 0.4$ pA (a) and $I = 7$ pA (b).

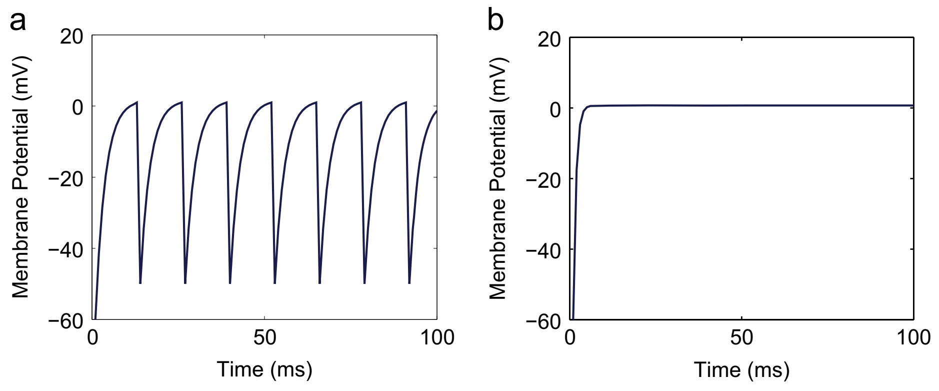

Fig. 2. Graphics of the spike train with $a = 0.1$, $I = 0.4$ pA; for $b = 0.3$ (a) and $b = 0.7$ (b).

adjustment factor to *include or exclude patterns from one class*. This is confirmed when $a=19$, as two different inputs generate the same number of spikes and a third one is barely differentiated.

Up to this point, the relationship between the increase of a and the size of the input set which belongs to the same pattern class seems to be linear. We can conclude from this that the increase or decrease of a could be important to pattern recognition, when wanting to group or separate the patterns in classes.

2.3.3. *a with* $[-1, 19]$ *range and 5 increments, b fixed to 0.1*

The same range of a is now tested for a positive value of b, namely $b=0.1$. With this we make this test directly comparable to Section 2.3.3. We can observe that for small I values and negative a we still do not generate a significant output. In a similar fashion to Section 2.3.3, it can be observed that increasing a results in a greater spike quantity and a lesser output difference between different input patterns.

Nonetheless, there is another detail which can be appreciated from the increase in a, which is more visible in the last 3 values of Table 3.

For $a=9$, two inputs generated the same spike quantity which, according to our classification criterion, means that they belong to the same class. The two other inputs can be related to a second and a third class respectively.

For $a=14$, we can observe that the outputs generated by the four inputs are strikingly similar. We could determine two different classes which only differ by one spike. This could lead us to think that for the next value of a, said difference will not exist.

For $a=19$, contrary to what was expected, the difference between classes increased (Table 3 and Fig. 4).

From this behavior, we can assume that by increasing a, we do not only increase the number of inputs in the set belonging to a class, but we also move the range of said set.

A more thorough analysis of the input universe for the combinations tested in this section would confirm all said assumptions, but up to this point it seems that the parameters of the I-F neuron model are indeed important for pattern classification.

2.4. *Pattern recognition method*

Pattern recognition will be carried out with the procedure described in [9–16]. The classification is made by counting the

Table 2
Spike quantity for all the inputs and the parameter combination in Section 2.3.2 for $T=100$ ms.

a	Input currents (pA)			
Value	0.4	1.6044	3	7
−1	0	0	0	0
4	0	0	6	13
9	10	13	16	24
14	19	19	24	32
19	24	32	32	33

Table 3
Spike quantity for all the inputs and the parameter combination in Section 2.3.3 for $T=100$ ms.

a	Input currents (pA)			
Value	0.4	1.6044	3	7
−1	0	4	7	16
4	12	14	16	24
9	19	24	24	32
14	32	32	33	33
19	33	33	49	49

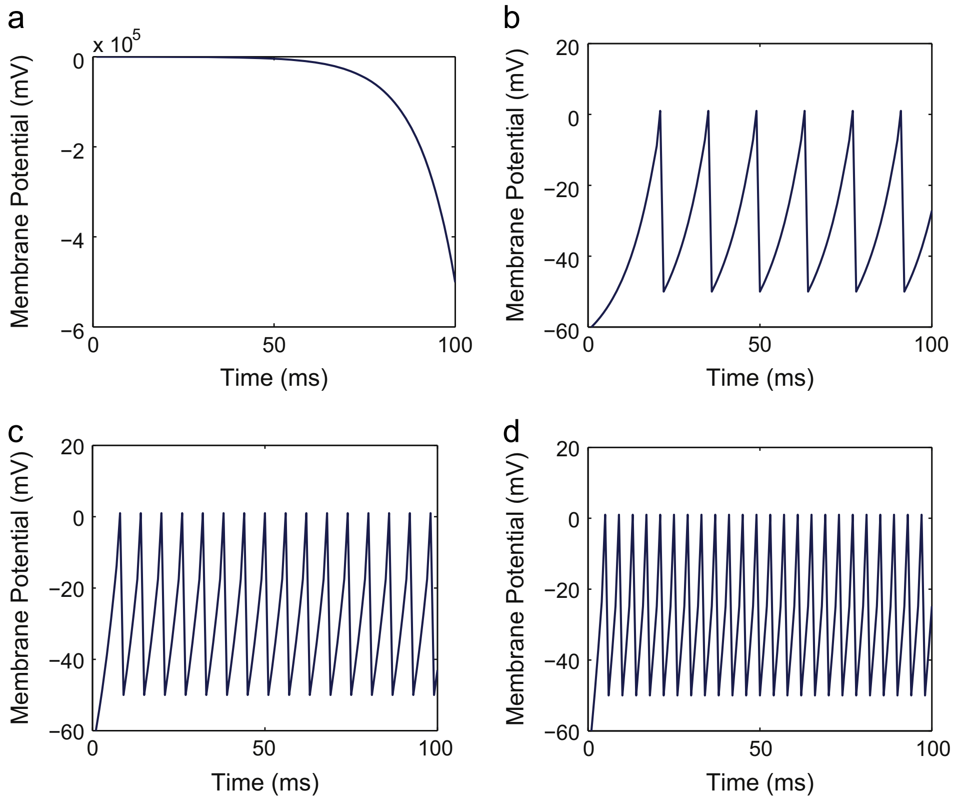

Fig. 3. Graphics of the spike train with $I=3$ pA, $b=-0.1$; for $a=-1$ (a) $a=4$, (b), $a=9$ (c) and $a=14$ (d).

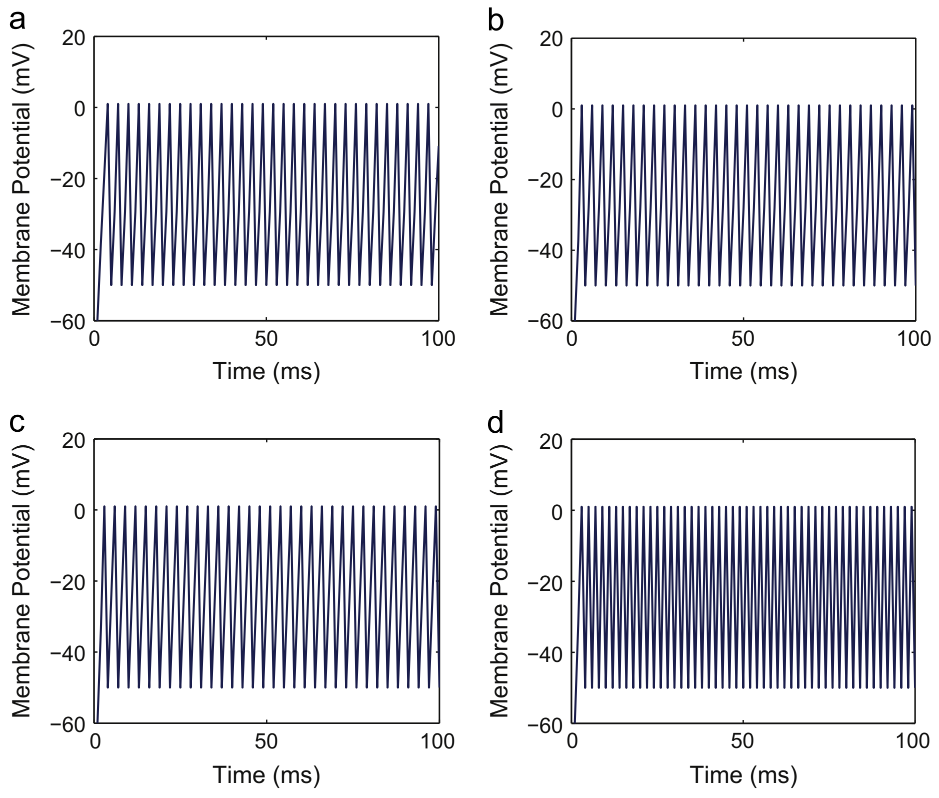

Fig. 4. Graphics of the spike train with $b = 0.1$; for $I = 1.6044$ pA, $a = 14$ (a), $I = 3$ pA, $a = 14$ (b), $I = 1.6044$ pA, $a = 19$ (c) and $I = 3$ pA, $a = 19$ (d).

number of spikes generated, by each of the input patterns, when processed by the neuron. To input them into the neuron they must first be converted into an equivalent current, to this end they are pre-processed with the following formula:

$$I = \vec{x} \cdot \vec{w} \cdot \theta \tag{3}$$

where $\vec{x}$ is the input pattern vector, $\vec{w}$ is the synaptic weights vector and θ is a gain factor. Before being input to the neural model, they are normalized to values between 0 and 1. This formula in conjunction with (2) are the whole input pattern to output pattern conversion process. After this, the firing rate produced by the model gives us the class of the input pattern by means of the equation

$$cl = \arg \min_{k=1}^{K} (|AFR_k - fr|) \tag{4}$$

where fr is the firing rate generated by the input pattern and AFR_k is the average firing rate for class k. This formula determines that an input pattern belongs to the class whose average firing rate is closer to the firing rate generated by the input.

The parameters which affect the behavior model up to this point can be summarized in the following set: $\{a, b, c, v_{threshold}, v_0, \vec{w}, \theta\}$. The set then contains parameters from both the LIF model and the input pre-processing into an equivalent current. To obtain good results during the classification task, the parameters must generate firing rates similar enough for patterns belonging to the same class and dissimilar enough when belonging to different classes. For that purpose a genetic algorithm could be used; having as evolving values any combination of the above summarized six parameters. The weight vector is never excluded from the evolving parameters.

An objective function, or fitness function, is proposed for the evolutionary algorithm. The function aims to minimize the number of classification errors and maximize the distance between the average firing rates:

$$f = \min\left(EQ \cdot \beta + \frac{1}{dist(\mathbf{AFR})} + PF\right) \tag{5}$$

where EQ is the number of classification errors for input patterns during training, β is the gain factor to make EQ more or less important in the function, $dist(\mathbf{AFR})$ is the sum of the euclidian distances between the classes' average firing rates and PF is the punishment factor to prevent meaningless outputs.

The classification errors are calculated by classifying the input patterns with (4). When the class defined by the equation does not match the real class of the input pattern, it is considered as an error and added to the error quantity.

To maximize the distance between the average firing rates within the same function, we do the equivalent operation of minimizing the inverse: $1/dist(\mathbf{AFR})$.

To obtain significant results, a punishment factor is added to the function for firing rates too low or too high which may indicate the neuron is under/over excited. This behavior may result in input information being lost in the neuron's processing, hence the importance in giving a bad evaluation value to individuals which cause this. Justification and proof of this is given in Section 2.3 of this paper.

The formula for calculating the punishment factor is

$$PF = \begin{cases} \delta, & \text{if } 0-\gamma < AFR < 0+\gamma \\ 0, & \text{if } 0+\gamma \le AFR \le T-\gamma \\ \delta, & \text{if } T-\gamma < AFR < T+\gamma \end{cases} \tag{6}$$

where δ is the selected punishment value, γ is the AFR tolerance range and T is the time the spiking neuron is left processing the input pattern.

3. Results

For experimentation, four datasets were chosen from the UCI machine learning benchmark repository [20]: the Iris plant dataset, the Wine dataset and the SPECT heart dataset and the Glass dataset. The last dataset is an object recognition problem denoted by the Screw dataset [21].

The five datasets are different from each other in the number of inputs, input attributes, classes and attributes data types. These characteristics are summarized in Table 4.

Different combinations for the genetic algorithm parameters were tested based on their meaning and on common literature recommended values [19–22], but the best results were achieved by the parameters described here.

For the training phase 50% of the data is used, and the other 50% is used for testing. Both phases have approximately the same number of inputs belonging to each of the classes, where, if they are not evenly divisible by 2, the remaining pattern is assigned to the testing phase.

The genetic algorithm has a stop criterion of 1000 generations, each generation with 50 individuals, this means that a maximum of $1000 \times 50 = 50\,000$ individuals will be tested. The number of possible individuals, which represents the domain size of the problem, varies depending on the codification, and the number of parameters being optimized. Although the domain size changes from one problem to the other, it was decided to leave these values fixed so that the proposed method proves it can be applied to different problems without much care for redefining the parameters.

The individuals are binary encoded, and each value within the individual has a length of 8 binary characters, this value was selected to keep computational cost low and as a byte multiple to ease C language implementation. With 8 symbols and 2 possible values, we have a total of 256 (2^8) possible permutations.

An elitism factor of 0.1 is used, meaning that the top 10 individuals from the population of 50, will be passed to the next generation without changes. A crossover chance of 0.9 is used, ensuring that most of the chosen individuals will breed. The two parents are chosen randomly from the whole population, and breed two offsprings, this process is repeated until the new population has a size of 100. The crossover algorithm uses a single cross-point chosen randomly from the length of the individual.

Mutation occurs to the offspring before being added to the new population. The mutation algorithm traverses the whole individual encoding, toggling each of the binary symbols (0 and 1) from one value to the other. This toggling occurs with a probability of the mutation factor, chosen as 0.2. This value was selected to estimate that a fifth of the individual's code, 1.6 characters, changes in average. In this way we ensure that at least one symbol changes, and another one changes more than half of the time.

All random functions within the genetic algorithm, such as parent and cross-point selection, have a uniform distribution.

For each of the datasets, four different configurations of the model's parameters will be optimized by the genetic algorithm:

0. First, only $\vec{w}$ will be optimized.
1. In a second combination, $\vec{w}$ and θ will be optimized.
2. The third combination optimizes $\vec{w}$, θ, a, b and c.
3. And the last combination optimizes all 7 parameters, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

The values of the parameters when not evolving, and their ranges when optimized by the genetic algorithm can be found in Table 5, where one can also observe that the fixed values are included within the optimization ranges.

For each of the parameter combinations, 10 experiments are done to statistically prove the results obtained. For each of the experiments, a different random set, with the specified size of 50%, is chosen for both the training and testing phases.

Two graphics are shown for each of the datasets. The first set of graphics shows the evolution of the fitness function's evaluation over generations, for the best individual and for each of the four different parameter combinations. This is only presented for the training phase, since it is the only stage where the fitness function, $f(x)$, is used. This function is applied as defined in (5), where it is presented as a minimization problem.

The values plotted are the average of the ten experiments (Figs. 5, 6, 7, 8 and 9). Here one can observe and compare the convergence speed and improvement rate of the algorithm.

For the cases where only $\vec{w}$ was optimized, one can see that the initial error is normally smaller, this is because the fixed values for the parameters were chosen from already known values which worked well for pattern recognition [9–16]. It would also seem that the function is optimized faster when including more parameters, particulary combination 1, since the other two more complete configurations greatly increase the problem domain size.

Table 4
Characteristics of the datasets used for testing.

Dataset	Inputs	Attributes	Classes	Data types
Iris	150	4	3	Real
Wine	178	13	3	Integer, real
SPECT	267	22	2	Binary
Glass	214	10	6	Integer, real
Screws	100	7	5	Real

Table 5
Values of the parameters when not evolving (fixed), and ranges when optimized by the genetic algorithm.

Parameter	Fixed value	Genetic algorithm	
		Min. value	Max. value
$\vec{w}$	N/A	−1	1
θ	0.4	0.1	1
a	0.5	−1	10
b	−0.001	−0.5	0.5
c	−50	−50	0
$v_{threshold}$	50	20	80
v_0	−60	−80	−40

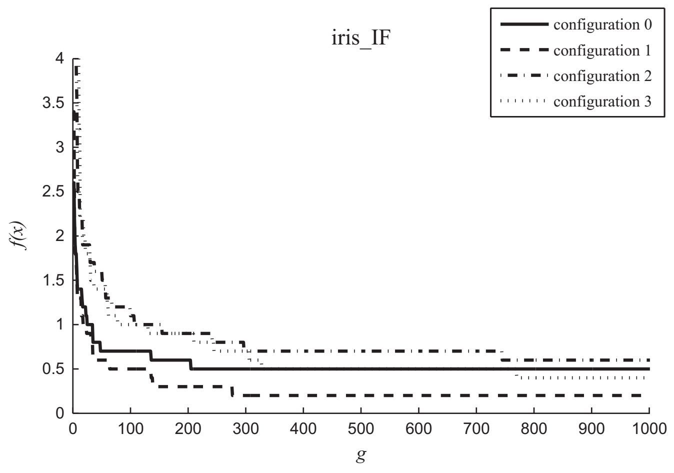

Fig. 5. Iris dataset, evolution of the fitness function evaluation for the best individual in a generation. Optimizing parameter combinations 0 through 3; for 0, $\vec{w}$; for 1, $\vec{w}$ and θ; for 2, $\vec{w}$, θ, a, b and c; and for 3, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

The Wine dataset is the only one that struggles between the first and second parameter configurations, since both are pretty close to one another. Both the third and fourth parameter configurations fall rapidly behind the first two.

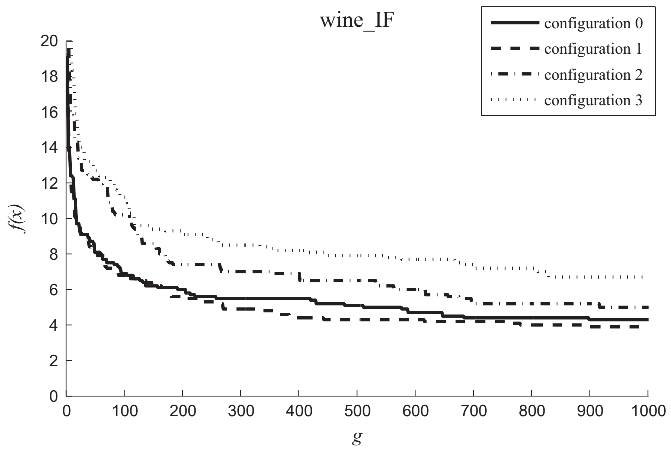

Fig. 6. Wine dataset, evolution of the fitness function evaluation for the best individual in a generation. Optimizing parameter combinations 0 through 3: for 0, $\vec{w}$; for 1, $\vec{w}$ and θ; for 2, $\vec{w}$, θ, a, b and c; and for 3, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

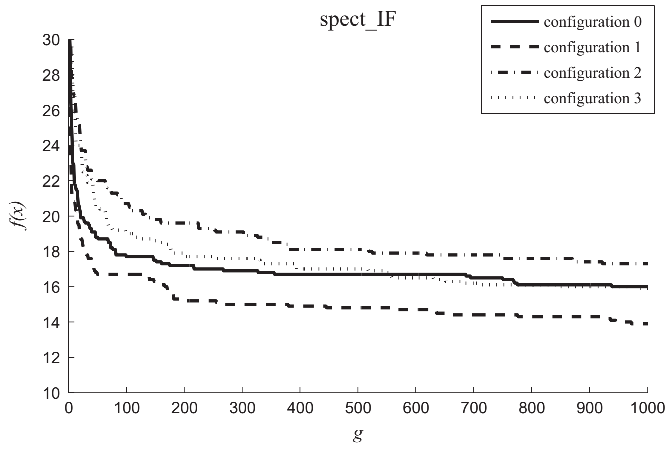

Fig. 7. SPECT dataset, evolution of the fitness function evaluation for the best individual in a generation. Optimizing parameter combinations 0 through 3: for 0, $\vec{w}$; for 1, $\vec{w}$ and θ; for 2, $\vec{w}$, θ, a, b and c; and for 3, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

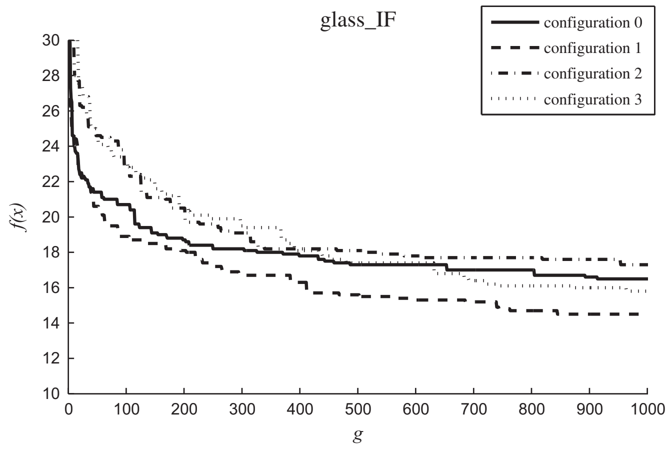

Fig. 8. Glass dataset, evolution of the fitness function evaluation for the best individual in a generation. Optimizing parameter combinations 0 through 3: for 0, $\vec{w}$; for 1, $\vec{w}$ and θ; for 2, $\vec{w}$, θ, a, b and c; and for 3, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

Another observation is that for the Screw dataset, the evaluation of the fitness function improved greatly when more parameters were included (Fig. 9).

The second set of graphics shows all of the inputs' spike trains, both from the training and testing phases (Figs. 10, 11, 12, 13 and 14). It shows a raster type graphic with the spike trains which had the highest classification efficiency, within the ten experiments for each combination. In most of them, one is able to easily differentiate graphically between classes, clearly showing how the firing rates help solving pattern recognition problems.

It is also appreciated how each problem was solved differently, as the graphics do not look similar to one another, such is the case of Fig. 13, where instead of having clearly differentiated classes with very similar firing rates, the classes are differentiated by firing rate ranges.

From the results obtained, it is apparent that including the weight factor θ improves the results in the training phase, although this does not always translate into better testing efficiency.

For all datasets, including more parameters resulted in lower minimum efficiencies, but in some even greater maximum efficiencies were achieved, this is understandable as the problem domain was greatly incremented.

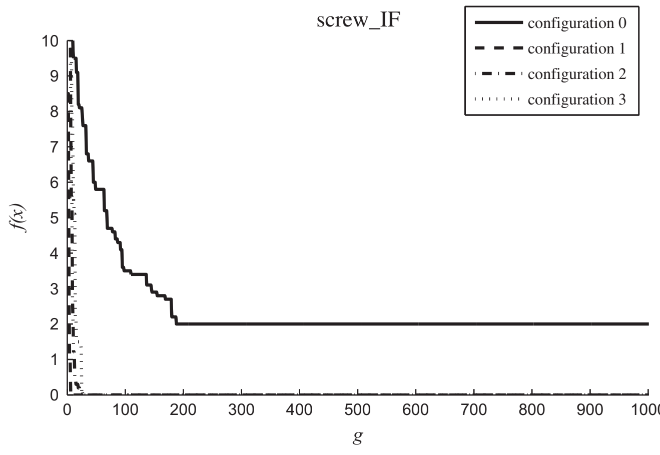

Fig. 9. Screw dataset, evolution of the fitness function evaluation for the best individual in a generation. Optimizing parameter combinations 0 through 3: for 0, $\vec{w}$; for 1, $\vec{w}$ and θ; for 2, $\vec{w}$, θ, a, b and c; and for 3, $\vec{w}$, θ, a, b, c, $v_{threshold}$ and v_0.

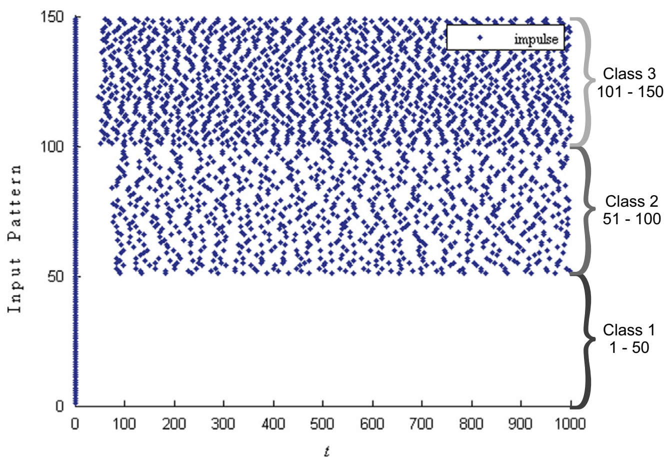

Fig. 10. Raster for the best individual of the Iris dataset, where we can appreciate the spike train (x-axis) for each of the inputs (y-axis). A dot is shown whenever a spike was present.

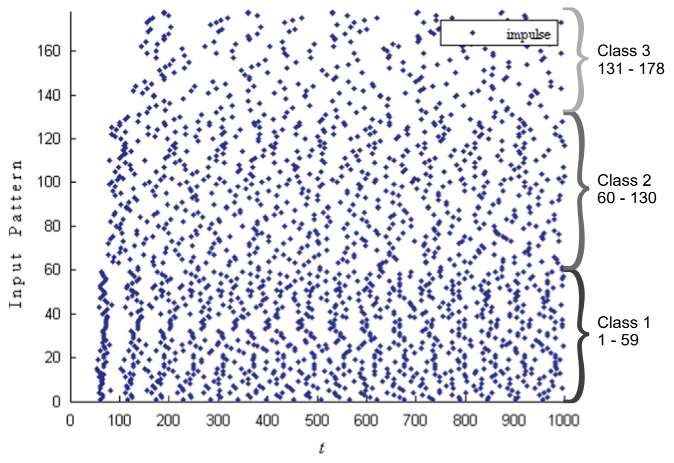

Fig. 11. Raster for the best individual of the Wine dataset, where we can appreciate the spike train (*x*-axis) for each of the inputs (*y*-axis). A dot is shown whenever a spike was present.

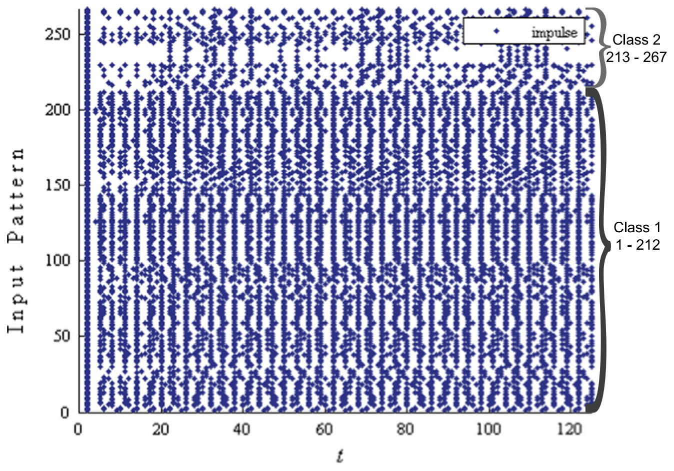

Fig. 12. Raster for the best individual of the SPECT dataset, where we can appreciate the spike train (*x*-axis) for each of the inputs (*y*-axis). A dot is shown whenever a spike was present.

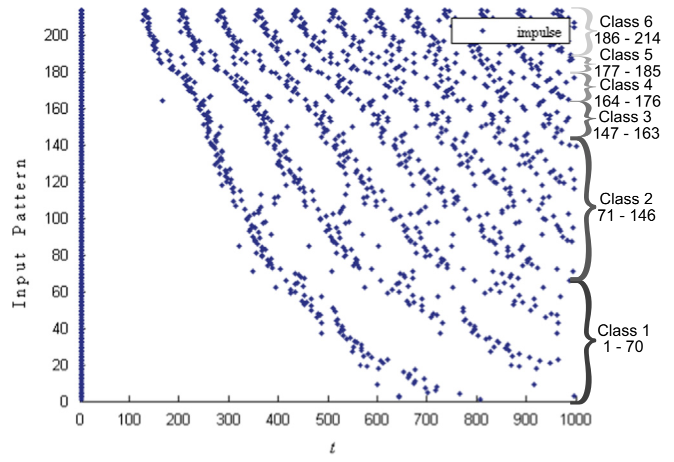

Fig. 13. Raster for the best individual of the Glass dataset, where we can appreciate the spike train (*x*-axis) for each of the inputs (*y*-axis). A dot is shown whenever a spike was present.

Further clarification on the classification efficiencies is provided by a table for each dataset (Tables 6, 7, 8, 9 and 10), which contains the average, maximum and minimum efficiencies for the training

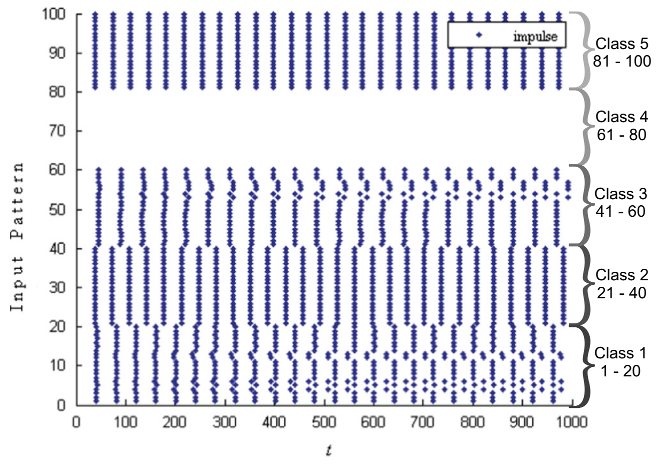

Fig. 14. Raster for the best individual of the Screw dataset, where we can appreciate the spike train (*x*-axis) for each of the inputs (*y*-axis). A dot is shown whenever a spike was present.

Table 6
Classification efficiencies for the Iris dataset.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.973	1.000	0.993	0.920	1.000	0.959
1	0.987	1.000	0.997	0.920	0.973	0.945
2	0.973	1.000	0.992	0.933	0.987	0.960
3	0.987	1.000	0.995	0.907	0.987	0.955

Table 7
Classification efficiencies for the Wine dataset.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.944	0.966	0.952	0.809	0.899	0.861
1	0.933	0.978	0.956	0.820	0.944	0.893
2	0.910	0.955	0.944	0.742	0.933	0.864
3	0.888	0.955	0.925	0.798	0.921	0.872

Table 8
Classification efficiencies for the SPECT heart dataset.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.835	0.902	0.880	0.784	0.888	0.832
1	0.872	0.925	0.895	0.769	0.843	0.806
2	0.842	0.902	0.870	0.784	0.866	0.820
3	0.842	0.902	0.880	0.724	0.843	0.793

and testing phases, and for each of the parameter combinations optimized.

3.1. Efficiency comparison with other methods

In this section, we compare our approach against previously reported results with other methods.

In addition, we perform a comparison against the well-known feedforward neural network (FNN). Two different topologies (1 and 2 hidden layers) and two different training algorithms (Descendant Gradient and Levenberg–Marquardt) [23] were used to design the FNN. The number of neurons in hidden layers depends on the

Table 9
Classification efficiencies for the Glass dataset.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.819	0.876	0.843	0.706	0.826	0.768
1	0.838	0.886	0.862	0.697	0.835	0.775
2	0.810	0.886	0.835	0.734	0.817	0.765
3	0.829	0.895	0.850	0.679	0.807	0.758

Table 10
Classification efficiencies for the Screw dataset.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.920	0.980	0.960	0.920	1.000	0.956
1	1.000	1.000	1.000	0.960	1.000	0.986
2	1.000	1.000	1.000	0.960	1.000	0.994
3	1.000	1.000	1.000	0.960	1.000	0.992

classification problem. The parameters for training algorithms were set as: 0.1 for the learning rate, 2000 epochs for the stop criterion. For each combination (topology-learning algorithm), 10 experiments are done to statistically prove the obtained results. For each experiment, we selected a different random set with the specified size of 40% and 10%, for both training and validation phases. The remaining 50% is selected for the testing phase.

Finally, instead of using the genetic algorithm to adjust the parameters of the spiking neuron, we compare our approach using the well-known differential evolution technique [24], with the same parameters and combinations described in the beginning of this section. To statistically prove the obtained results, 10 experiments were carried out.

From the maximum efficiencies achieved in the testing phase by each of the parameter combinations, we choose the least and most efficient cases. These efficiencies are then compared to previously reported results with other methods.

3.1.1. *Iris dataset*

In [25], using multi-class support vector machines and tenfold cross-validation, a 99.33% efficiency was achieved. In [26], on the section where they summarize previously achieved efficiencies, the maximum reported is 95.53%, which was managed by a Bagging simple Bayes (NB) method, again statistically validated by tenfold-cross validation.

Table 11 shows the classification efficiencies for the iris dataset using feedforward neural networks (FNN). Each row contains the obtained results using a combination of topology and learning algorithms during training and testing phases. Label DG1 means that descendant gradient method and one hidden layer was used to design the FNN, label LM2 indicates that Levenberg–Marquardt method and two hidden layers were used to design de FNN, and so on.

For this dataset, we designed a topology of one hidden layer composed of four neurons. On the other hand, we designed a topology of two hidden layers composed of two and two neurons, respectively.

Table 12 shows the accuracy of our approach using the differential evolution method instead of genetics algorithms.

The proposed method achieved 97.3% and 100% for the least and most efficient cases respectively. When DE is used in combination with our approach, we observed that the classification efficiency slightly decreases. Although other methods have already achieved very high classification rates, getting a 100% efficiency speaks for the method capabilities.

Table 11
Classification efficiencies using feedforward neural networks for the iris dataset.

Training	Training phase			Testing phase		
Algorithm	Min	Max	Avg	Min	Max	Avg
DG1	0.73	0.986	0.912	0.60	1.000	0.878
DG2	0.32	1.000	0.722	0.34	1.000	0.713
LM1	0.986	1.000	0.996	0.960	1.000	0.985
LM2	0.346	1.000	0.754	0.253	1.000	0.701

Table 12
Classification efficiencies for the Iris dataset using DE.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.973	1.000	0.975	0.920	0.960	0.944
1	0.960	1.000	0.980	0.920	0.973	0.951
2	0.947	1.000	0.983	0.920	0.973	0.949
3	0.853	1.000	0.968	0.827	0.973	0.928

Table 13
Classification efficiencies using feedforward neural networks for the wine dataset.

Training	Training phase			Testing phase		
Algorithm	Min	Max	Avg	Min	Max	Avg
DG1	0.966	1.000	0.988	0.91	0.988	0.961
DG2	0.662	1.000	0.944	0.573	0.988	0.909
LM1	1.000	1.000	1.000	0.932	1.000	0.966
LM2	0.966	1.000	0.994	0.91	1.000	0.967

3.1.2. *Wine dataset*

Again in [25],[1] a 90% efficiency was achieved for this dataset. Many efficiencies for this dataset are reported in [26], the highest of them is 98.71% which was achieved by a lazy Bayesian rule learning algorithm (LBR).

Table 13 shows the classification efficiencies for the Wine dataset using feedforward neural networks (FNN). For this dataset, we designed a topology of one hidden layer composed of 16 neurons. On the other hand, we designed a topology of two hidden layers composed of 10 and six neurons, respectively.

Table 14 shows the accuracy of our approach using the differential evolution method instead of genetics algorithms.

The proposed method achieved 89.9% and 94.4% for the least and most efficient cases respectively. Though it did not achieve a better classification rate, the efficiency achieved is very acceptable. When DE is used in combination with our approach, we observed that the classification efficiency slightly decreases.

3.1.3. *SPECT dataset*

The SPECT heart dataset is the only dataset included which has a train and testing subset already defined. In [27], the TAN algorithm is reported to have achieved an efficiency of 81.25%; the whole dataset was randomly divided into three sets, using two for training and one for testing. An ensemble of CLIP4 classifiers is

[1] In the referred article, two efficiencies are reported when a pattern classification is tied for two or more classes. They correspond to the cases when patterns are assigned to the right and the wrong classes, respectively. Since the method proposed in the present paper considers this particular scenario to be an error, the lower of the two efficiencies is considered for comparison.

Table 14
Classification efficiencies for the Wine dataset using DE.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.854	0.966	0.919	0.787	0.899	0.829
1	0.865	0.944	0.911	0.787	0.910	0.849
2	0.820	0.978	0.896	0.719	0.910	0.828
3	0.809	0.944	0.893	0.685	0.921	0.812

Table 15
Classification efficiencies using feedforward neural networks for the SPECT dataset.

Training	Training phase			Testing phase		
Algorithm	Min	Max	Avg	Min	Max	Avg
DG1	0.843	0.932	0.901	0.751	0.827	0.785
DG2	0.82	0.932	0.88	0.744	0.864	0.796
LM1	0.768	0.925	0.859	0.616	0.834	0.773
LM2	0.746	0.925	0.814	0.714	0.842	0.778

Table 16
Classification efficiencies for the SPECT heart dataset using DE.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.812	0.917	0.883	0.701	0.821	0.781
1	0.759	0.895	0.828	0.687	0.813	0.741
2	0.865	0.925	0.894	0.754	0.828	0.797
3	0.820	0.917	0.877	0.716	0.851	0.795

reported to have achieved 90.4% accuracy [28], using the original train and testing subsets of 80 and 187 patterns respectively (roughly 30% and 70%).

Table 15 shows the classification efficiencies for the SPECT dataset using feedforward neural networks (FNN). For this dataset, we designed a topology of one hidden layer composed of 24 neurons. On the other hand, we designed a topology of two hidden layers composed of 15 and nine neurons, respectively.

Table 16 shows the accuracy of our approach using the differential evolution method instead of genetics algorithms.

The proposed method, with the evenly randomly selected train and test sets (50% each), obtained 84.3% and 88.8% for the least and most efficient cases. This is very competitive considering the simplicity of the model. When DE is used in combination with our approach, we observed that the classification efficiency slightly decreases.

3.1.4. *Glass dataset*

In [25], a 70.48% efficiency was achieved for this dataset. The highest results reported were found in [29], for the Boost nearest neighbor classifier (Boost NN), with a 75.6% efficiency.

Table 17 shows the classification efficiencies for the Glass dataset using feedforward neural networks (FNN). For this dataset, we designed a topology of one hidden layer composed of 16 neurons. On the other hand, we designed a topology of two hidden layers composed of 10 and six neurons, respectively.

Table 18 shows the accuracy of our approach using the differential evolution method instead of genetics algorithms.

The proposed method achieved 80.7% and 83.5% for the least and most efficient cases respectively, which is a high improvement. When DE is used in combination with our approach, we observed that the classification efficiency slightly increases to almost 90%. This is very competitive considering the simplicity of the model.

Table 17
Classification efficiencies using feedforward neural networks for the glass dataset.

Training	Training phase			Testing phase		
Algorithm	Min	Max	Avg	Min	Max	Avg
DG1	0.57	0.906	0.713	0.504	0.878	0.678
DG2	0.439	0.831	0.656	0.439	0.887	0.653
LM1	0.794	0.953	0.865	0.616	0.897	0.754
LM2	0.504	0.925	0.818	0.411	0.85	0.725

Table 18
Classification efficiencies for the Glass dataset using DE.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.724	0.895	0.815	0.633	0.817	0.726
1	0.438	0.895	0.778	0.385	0.844	0.717
2	0.467	0.819	0.695	0.339	0.771	0.632
3	0.457	0.905	0.763	0.349	0.899	0.688

Table 19
Classification efficiencies using feedforward neural networks for the screw dataset.

Training	Training phase			Testing phase		
Algorithm	Min	Max	Avg	Min	Max	Avg
DG1	0.48	0.86	0.734	0.34	0.82	0.67
DG2	0.22	0.84	0.56	0.18	0.80	0.456
LM1	0.78	1.000	0.978	0.78	1.000	0.974
LM2	0.60	1.000	0.864	0.56	1.000	0.848

Table 20
Classification efficiencies for the Screw dataset using DE.

Param.	Training phase			Testing phase		
Comb.	Min	Max	Avg	Min	Max	Avg
0	0.960	1.000	0.994	0.940	1.000	0.980
1	1.000	1.000	1.000	0.980	1.000	0.996
2	1.000	1.000	1.000	0.960	1.000	0.994
3	1.000	1.000	1.000	1.000	1.000	1.000

3.1.5. *Screw dataset*

In [21], a 100% efficiency was achieved for this dataset.

Table 19 shows the classification efficiencies for the Screw dataset using feedforward neural networks (FNN). For this dataset, we designed a topology of one hidden layer composed of 12 neurons. On the other hand, we designed a topology of two hidden layers composed of seven and five neurons, respectively.

Table 20 shows the accuracy of our approach using the differential evolution method instead of genetics algorithms.

The proposed method achieved 100% efficiency for all cases.

4. Conclusions

It has been demonstrated that the I&F model parameters are important to the pattern recognition model proposed. When varying these parameters with the evolutionary algorithm, the results changed significantly, improving the converging speed on most cases but for the Wine dataset. Although the fixed values for the Wine dataset were within the optimization range for the genetic algorithm, such values were never found. From this, we can conclude that improvements can be done to the evolutionary algorithm chosen, be it to choose another kind or change its parameters and/or functions.

When including all parameters, the tendency was to generate more consistent results across experiments, and across the training and testing phases. This, combined with a better evolutionary algorithm may prove to obtain better results consistently among different datasets. A point worth mentioning is that none of the model's parameters were varied from one dataset to another, by any other means but the evolutionary algorithm. The intent of this is to show the robustness of the model and its application to several domains.

The classification efficiencies achieved by the method are very acceptable, improving for most cases on other methods. A more flexible spiking neuron model, such as proposed by Eugene Izhikevich in [30], may further improve results, since it can vary its spiking behavior depending on its parameters.

Acknowledgments

The authors thank Universidad La Salle for the economical support under grant no. ULSA I-054/11 and ULSA I-061/12. Aleister Cachón thanks CONACyT for the scholarship provided.

References

[1] W. Maass, C.M. Bishop, Pulsed Neural Networks, , 2001.
[2] F. Rieke, D. Warland, Robert, W. Bialek, Spikes—exploring the neural code, 1997.
[3] S. Loiselle, J. Rouat, D. Pressnitzer, S. Thorpe, Exploration of rank order coding with spiking neural networks for speech recognition, in: IEEE International Joint Conference on Neural Networks, IJCNN '05, Proceedings, vol. 4, 2005, pp. 2076–2080.
[4] H. Azhar, K. Iftekharuddin, R. Kozma, A chaos synchronization-based dynamic vision model for image segmentation, in: IEEE International Joint Conference on Neural Networks, 2005, IJCNN '05, Proceedings.
[5] S.J. Thorpe, R. Guyonneau, N. Guilbaud, J.-M. Allegraud, R. Van Rullen, Spikenet: real-time visual processing with one spike per neuron, Neurocomputing 58–60 (2004) 857–864, Computational Neuroscience: Trends in Research 2004.
[6] E. Di Paolo, Spike-timing dependent plasticity for evolved robots, Adapt. Behav. 10 (2002) 243–263.
[7] D. Floreano, J.-C. Zufferey, J.-D. Nicoud, From wheels to wings with evolutionary spiking neurons, Artif. Life 11 (2005) 121–138.
[8] M.E. Hasselmo, C. Bodelón, B.P. Wyble, A proposed function for hippocampal theta rhythm: separate phases of encoding and retrieval enhance reversal of prior learning, Neural Comput. 14 (2002) 793–817.
[9] R. Vazquez, A. Cachon, Integrate and fire neurons and their application in pattern recognition, in: 7th International Conference on Electrical Engineering Computing Science and Automatic Control (CCE), 2010, pp. 424–428.
[10] S.M. Bohte, J.N. Kok, H.L. Poutre, Error-backpropagation in temporally encoded networks of spiking neurons, Neurocomputing 48 (2002) 17–37.
[11] A. Belatreche, L.P. Maguire, T.M. McGinnity, Pattern recognition with spiking neural networks and dynamic synapse, in: International FLINS Conference on Applied Computational Intelligence, pp. 205–210.
[12] S. Kamoi, R. Iwai, H. Kinjo, T. Yamamoto, Pulse pattern training of spiking neural networks using improved genetic algorithm, in: IEEE International Symposium on Computational Intelligence in Robotics and Automation, Proceedings, vol. 2, 2003, pp. 977–981.
[13] H.N. Abdull Hamed, N. Kasabov, Z. Michlovský, S.M. Shamsuddin, String pattern recognition using evolving spiking neural networks and quantum inspired particle swarm optimization, in: Proceedings of the 16th International Conference on Neural Information Processing: Part II, ICONIP '09, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 611–619.
[14] S. Hong, L. Ning, L. Xiaoping, W. Qian, A cooperative method for supervised learning in spiking neural networks, in: 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2010, pp. 22–26.
[15] R. Vazquez, Izhikevich neuron model and its application in pattern recognition, Aust. J. Intell. Inf. Process. Syst. 11 (2010) 35–40.
[16] R.A. Vázquez, Pattern recognition using spiking neurons and firing rates, in: Proceedings of the 12th Ibero-American Conference on Advances in Artificial Intelligence, IBERAMIA'10, LNAI, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 423–432.
[17] E.M. Izhikevich, Simple model of spiking neurons, IEEE Trans. Neural Netw. 14 (2003) 1569–1572.
[18] E.M. Izhikevich, Dynamical Systems in Neuroscience: the Geometry of Excitability and Bursting, Computational Neuroscience, MIT Press, Cambridge, MA, USA, 2007.
[19] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Boston, MA, USA, 1989.
[20] P. Murphy, D. Aha, UCI Repository of Machine Learning Databases, Technical Report, University of California, Department of Information and Computer Science, Irvine, CA, US, 1994.
[21] R.A. Vazquez Espinoza De Los Monteros, J.H. Sossa Azuela, A new associative model with dynamical synapses, Neural Process. Lett. 28 (2008) 189–207.
[22] T.M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, USA, 1996.
[23] M.I.A. Lourakis, A brief description of the Levenberg–Marquardt algorithm implemented by Levmar, Foundation for Research and Technology – Hellas, Vassilika Vouton, P.O. Box 1385, GR 711 10 Heraklion, Crete, Greece, 2005.
[24] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution a Practical Approach to Global Optimization, Natural Computing Series, Springer-Verlag, Secaucus, NJ, USA, 2005.
[25] P. Zhong, M. Fukushima, Regularized nonsmooth Newton method for multi-class support vector machines, Optim. Methods Softw. 22 (2007) 225–236.
[26] S.B. Kotsiantis, P.E. Pintelas, Logitboost of simple Bayesian classifier, Informatica 29 (2005) 53–59.
[27] M.G. Madden, Evaluation of the performance of the Markov blanket Bayesian classifier algorithm, CoRR cs.LG/0211003, 2002.
[28] L. Kurgan, K.J. Cios, Ensemble of classifiers to improve accuracy of the clip4 machine-learning algorithm, in: Proceedings of SPIE International Conference on Sensor Fusion: Architectures, Algorithms,and Applications, vol. VI, pp. 22–31.
[29] V. Athitsos, S. Sclaroff, Boosting Nearest Neighbor Classifiers for Multiclass Recognition, Technical Report, Boston University, 2004.
[30] E.M. Izhikevich, Which model to use for cortical spiking neurons? IEEE Trans. Neural Netw. 15 (2004) 1063–1070.

Aleister Cachón received his M.Sc. in Cybernetics from La Salle University, Mexico City, Mexico, 2012. From both Mexican and Spanish heritage, he is happily married and has a passion for all computer related fields, always eager to learn something new.

Roberto A. Vázquez received his B.Sc. degree from the School of Computer Sciences, National Polytechnic Institute (ESCOMIPN), Mexico City, Mexico, 2003. He received his M.Sc. degree from the Center for Computing Research, National Polytechnic Institute (CIC-IPN), Mexico City, Mexico, 2005 and his Ph.D. degree at the Center for Computing Research, National Polytechnic Institute (CIC-IPN), Mexico City, Mexico, 2009. He is a full time professor at Faculty of Engineering, La Salle University, Mexico City, Mexico. His main research interests are Artificial Intelligence, Neurocomputing, Computational Neuroscience, Associative Memories, Pattern Recognition and Image Analysis.