Soft Comput
DOI 10.1007/s00500-011-0793-1

ORIGINAL PAPER

Improved Izhikevich neurons for spiking neural networks

Stylianos Kampakis

Abstract Spiking neural networks constitute a modern neural network paradigm that overlaps machine learning and computational neurosciences. Spiking neural networks use neuron models that possess a great degree of biological realism. The most realistic model of the neuron is the one created by Alan Lloyd Hodgkin and Andrew Huxley. However, the Hodgkin–Huxley model, while accurate, is computationally very inefficient. Eugene Izhikevich created a simplified neuron model based on the Hodgkin–Huxley equations. This model has better computational efficiency than the original proposed by Hodgkin and Huxley, and yet it can successfully reproduce all known firing patterns. However, there are not many articles dealing with implementations of this model for a functional neural network. This study presents a spiking neural network architecture that utilizes improved Izhikevich neurons with the purpose of evaluating its speed and efficiency. Since the field of spiking neural networks has reinvigorated the interest in biological plausibility, biological realism was an additional goal. The network is tested on the correct classification of logic gates (including XOR) and on the iris dataset. Results and possible improvements are also discussed.

Keywords Izhikevich neuron · Spiking neural network · Rebound spiking · Classification · Receptive fields · Biologically plausible

S. Kampakis (✉)
Department of Computer and Electrical Engineering,
Aristotle University of Thessaloniki, Ano Tzoumagias 1,
Thessaloniki, Greece
e-mail: stylianos.kampakis@gmail.com

1 Introduction

Spiking neural networks (Maass and Bishop 2001) are a machine learning paradigm that emerged during the 1990s. They focus on the use of biologically realistic neuron models and information coding schemes. Spiking neural networks are categorized as the third generation of neural networks (Maass 1997), the other two being neural networks using analog neurons (second generation) and binary neurons (first generation).

Spiking neural networks are closely related to computational neuroscience. Computational neuroscience aims at the realistic representation of the brain. The official creation of this discipline can be traced to the year 1952, when Alan Hodgkin and Andrew Huxley published their seminal work based on findings they derived from the study of a squid giant axon (Hodgkin and Huxley 1952). Since then, many more neuron models have been created, such as integrate-and-fire neurons and the FitzHugh–Nagumo model (Gerstner and Kistler 2002). Each model faces a tradeoff between biological plausibility and computational efficiency. Izhikevich (2004) provides an account of the differences in computational efficiency among existing neuron models.

Spiking neural networks are so called because they encode information by using spikes. There is substantial research that supports the view that the brain uses spikes to encode information (Shadlen and Newsome 1994; Rieke et al. 1996). However, there is no single accepted best coding scheme. It is very likely that the brain uses many coding schemes in different regions at any given point in time. Many coding schemes have been proposed for spiking neural networks, such as rank order coding, timing coding, and binary coding (Thorpe et al. 2001).

Different neuron models have been used for machine learning tasks. For example, Lanella and Back (2001) have

Published online: 27 November 2011

used a spiking neural network for function approximation. Wu and (2007) have used a spiking neural network architecture for edge detection. Spiking neural networks have also been used for clustering (Bohte et al. 2001; Xiaoli and Howard 2004). Thus far, the neuron model created by Eugene Izhikevich (2003) is not very popular for spiking neural network implementations, even though it combines biological realism, with computational efficiency. This research addresses this issue and examines the potential of the model for machine learning tasks.

1.1 Izhikevich neuron model

The neuron model created by Eugene Izhikevich (Izhikevich 2003) is defined by the following equations:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{1}$$

$$u' = a(bv - u) \tag{2}$$

$$\text{If } v\,\varepsilon\,30 \text{ then } (v = c \text{ and } u = u + d) \tag{3}$$

Here a, b, c, and d are parameters, I is the input, v is the voltage of the neuron's membrane, and u is the recovery variable. The parameters are important in terms of maintaining the biological realism of the model. Some combinations of values for the parameters a, b, c, and d could lead to biologically unrealistic results.

It is evident that the Izhikevich neuron uses voltage as its modeling variable. However, this is not a completely accurate representation of the neuron's function. Even though neurons utilize current, their function is not purely electrical, but electro-chemical, being based on ion channels and neurotransmitters. The Hodgkin–Huxley model allows explicit modeling of these variables, since it implements many more variables, including sodium and potassium currents. Nevertheless, the Izhikevich neuron model remains biologically plausible because it is able to capture every known firing pattern. Furthermore, the model is computationally very efficient, and is close to the integrate-and-fire neuron model in terms of implementation cost. According to a graph in Izhikevich (2004), this model represents the best compromise between speed and biological realism.

Wang (2009) has proposed an improved version of the model, which limits the voltage of the spike to 30 mV. In the network discussed in this study, the Wang–Izhikevich neuron is used, instead of the original neuron proposed by Eugene Izhikevich for two reasons. First, the Izhikevich neuron could at times lead to unrealistically high voltages in the neuron (sometimes more than 100 mV). Second, in practice, limiting the spike to 30 mV provided better visualization of the spikes, which helped greatly as the network was being coded and tested.

In practice, real neurons can spike over 30 mV, something which can be reproduced by the Hodgkin–Huxley model. However, limiting the upper threshold to 30 mV allows for an easier implementation of the network, since a spike can be considered to occur only whenever a neuron reaches this threshold. The model proposed by Wang (2009) does not impose a hard-limit on the voltage, but, creates a biologically realistic spike instead.

2 Network architecture and function

The network's first layer is the input layer, which comprises numerous improved Izhikevich neurons. Since the improved Izhikevich neurons accept as an input only a voltage variable I, as it was demonstrated in Eq. 1, an information coding scheme for the input had to be used. For this purpose Gaussian receptive fields were utilized described by Eq. 4.

$$f(x; \sigma, c) = e^{\frac{-(x-c)^2}{2\sigma^2}} \tag{4}$$

In Eq. 4, σ is the standard deviation of the function, and c is the center. Receptive fields have been used before in spiking neural networks (Eurich and Wilke 2000; Glackin et al. 2011). They also contain a high degree of biological realism, since it has been documented that such a coding scheme might be used by the brain to code quantities (Nieder and Dehaene 2009). In order to preserve the biological realism, the author will also use the term "sensor neurons" to describe the Gaussian receptive fields of the input layer.

The Gaussian receptive fields separate the input space into different regions and the activation of each receptive field is translated into input voltage for each improved Izhikevich neuron (Fig. 1). Note that, for each neuron, only a single Gaussian receptive field was used. The sensor neurons yielded an output in the [0, 1] range; however, this proved to be a very small value for the improved Izhikevich neurons to have any important effect.

In the original demonstration file (Izhikevich, n.d.) Izhikevich used a random input for each neuron from the range [−5, 2]. This input resulted in a random number of neurons firing each time, depending not only on the intensity of the stimulus, but also on their randomly initialized parameters, as well. By setting the demonstration input to the range [0, 1] only a small subset of the neurons fired.

The original demonstration file of Eugene Izhikevich was modified to incorporate the improved Izhikevich neuron model. 1,000 neurons with randomly initialized parameters were tested for various intensities. The improved Izhikevich neurons failed to respond to intensities lower than 4.2 for a number of 100 simulated ms. These results are demonstrated in Fig. 2.

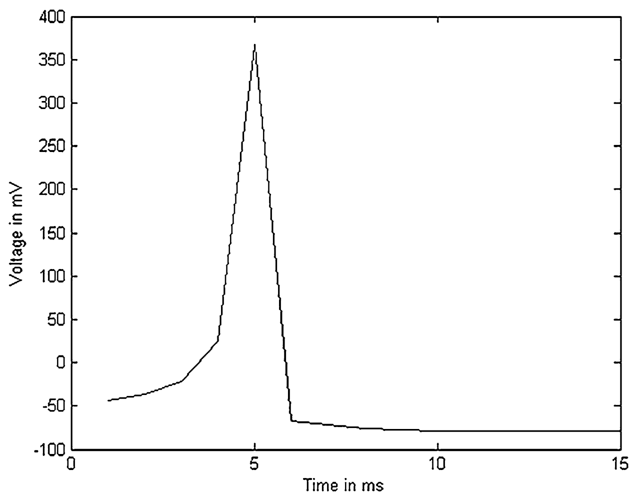

Fig. 1 An input is coded with two different Gaussian receptive fields

There is a possibility that the improved Izhikevich neuron model could fire in even lower intensities by changing the neuron parameters. However, changing the parameters beyond certain values can provide biologically unrealistic results. In addition, it could utterly change the behavior of the neuron making the simulation very difficult. Therefore, the solution implemented was to use a variable called *amplifier* multiplies the output of every Gaussian field before it reaches the input neurons. If the user does not want to let this variable affect the result, then this variable can be simply set to 1. The effect of the variable is shown in Eq. 5 and is also illustrated in Fig. 3. The Greek letter α in Eq. 5 stands for the amplifier.

$$f(x;\sigma,c,a) = \left(e^{\frac{-(x-c)^2}{2\sigma^2}}\right) \times \alpha \tag{5}$$

After the input layer, one or more layers are connected in a feed-forward fashion. A spike occurs anytime the voltage reaches 30 mV. While the neurons communicate with spikes, the improved Izhikevich neuron accepts voltage as an input. A new variable called *voltage_sent* was also added that indicated the voltage transmitted from one neuron to the next. This variable represents, in biological terms, the voltage that a firing neuron transmits to the receiving neurons. This variable is multiplied by the weight of connection between two neurons to calculate the total effect. In this implementation the variable *voltage_sent* is a global variable (the value is the same for all neurons). The variable was chosen to be global, because it was considered that a training algorithm that needed only to change the weights would face less complexity than if it needed to change *voltage_sent*, as well. After all, by setting the *voltage_sent* as a constant, Eq. 6 shows clearly that voltage received can be purely manipulated as a function of the weight between the two neurons.

$$\text{Voltage}_{\text{received by } i} = \text{Voltage}_{\text{sent by } j} \times \text{weight}_{j \text{ to } i} \tag{6}$$

The final layer is considered by default the output layer.

Eugene Izhikevich in the Matlab demonstration of his model (Izhikevich, n.d.) preferred to simulate the model by iterating over a specified timestep[1] instead of using a numerical method. This was the approach taken in this paper, as well. The model is iterated for a specified amount of times with a specified a timestep of 1 ms for each iteration. The timestep of 1 ms was chosen, because a smaller timestep changed the behavior of the improved Izhikevich neuron in biologically unrealistic ways. The default implementation of the improved Izhikevich neuron does not allow the neuron spike to exceed 30 mV. Using a timestep of 0.5 ms this threshold could be surpassed.

3 Tests

The network was used in two tests. The first one was to simulate all logic gates: AND, OR, NOT, XOR, NOR, NAND, and XNOR. The XOR problem is, probably, the most famous problem in neural networks, owing to the inability of a single-layered perceptron to approximate it, because it is non-linearly separable (Minsky and Papert 1972). Since then, the XOR problem has become a benchmark for any new type of neural network. The rest of the logic gates are tested to have a clearer picture of the speed of training.

The second test was to classify the irises in the iris dataset. The iris dataset was introduced by Fisher (1936) and has become a standard benchmark for many classification studies.

3.1 Logic gates architecture

For the logic gates test two different networks were used. The first network represents the same implementation strategy that was used for the iris classification task. There are two input neurons, with each one having a respective receptive field. The receptive field of the first neuron is centered on 0, and the second receptive field is centered on 1. There are two output neurons, with the first neuron corresponding to the answer "1(TRUE)" and the second one to the answer "0(FALSE)". So, in each case, only one neuron has to be active. The pattern {first neuron = 1, second neuron = 0} indicates a TRUE response, and the pattern {first neuron = 0, second neuron = 1} indicates a FALSE response. The network uses a winner-takes-all strategy, where the output is received when the first neuron

[1] In the default implementation found in (Izhikevich, n.d.) the timestep is 0.5 ms.

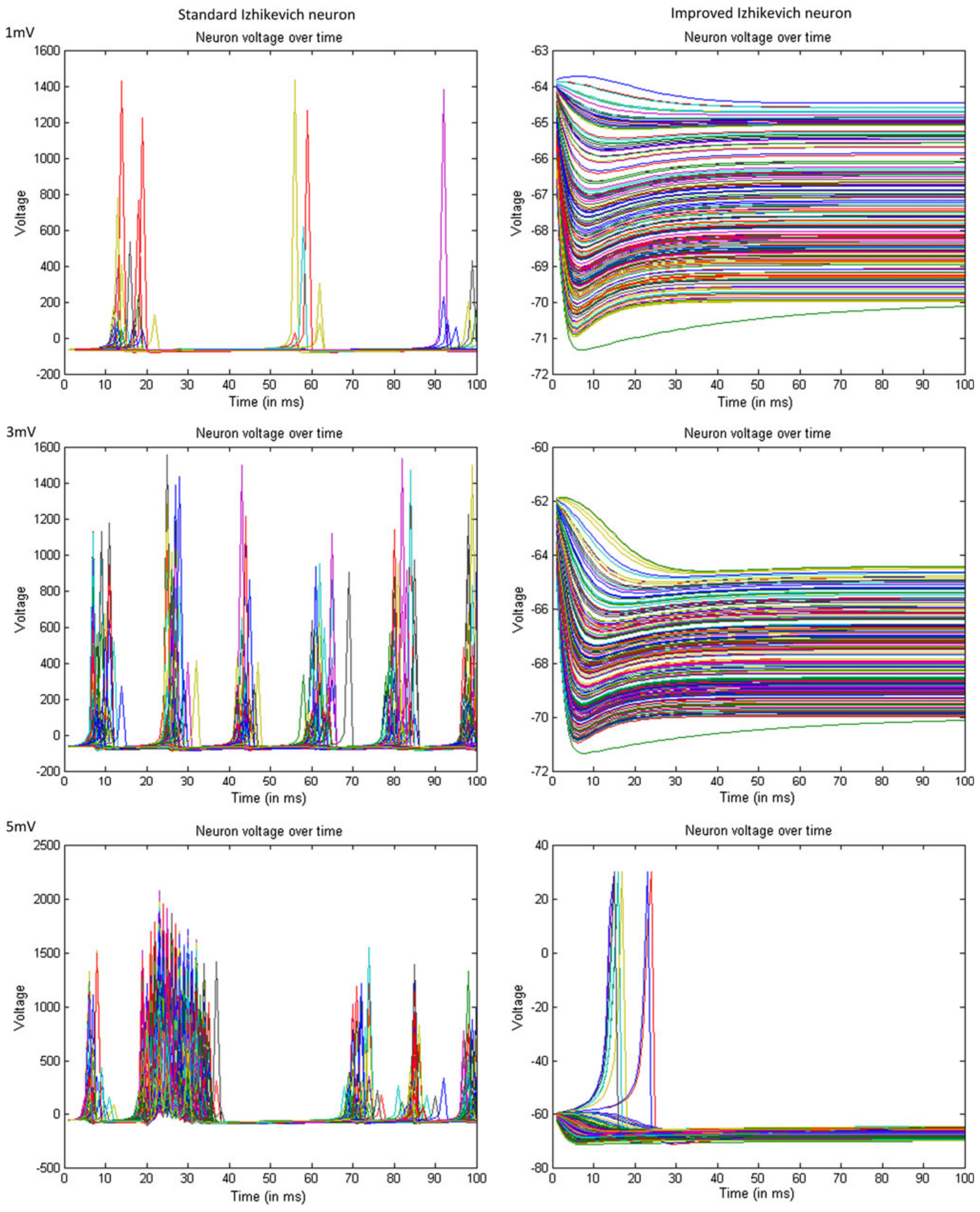

Fig. 2 Comparison of standard Izhikevich neuron and improved Izhikevich neuron

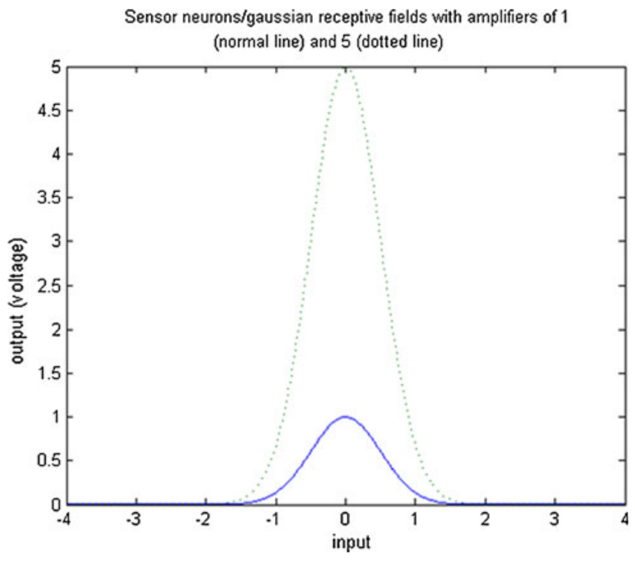

Fig. 3 Sensor neuron without amplifier (amplifier = 1) and with amplifier (amplifier = 5)

fires. So, it is possible that both neurons might fire, but since neuron 1 fired first, the pattern is considered to indicate the response TRUE. Therefore, the absolute timing does not matter, but rather it is the relative timing of the output spikes that matters. Note that the case where the two neurons fire at the same time is erroneous and does not correspond to any output, since it would correspond to an answer being TRUE and FALSE at the same time.

The second network takes advantage of the fact that the output of the logic gates is binary. In this case, there is only one output neuron and firing during the simulation time corresponds to TRUE, while not firing (during the simulation time) corresponds to FALSE. The two architectures are graphed in Fig. 4.

3.2 Iris classification architecture

For the iris data set only two variables were used (sepal width and sepal length), since the other two (petal length, petal width) have been found to be less important for the classification task (Valko et al. 2005). The network consisted of six sensor neurons, three for each input variable. The data were normalized and using *k*-means clustering, the centers of the three classes for each variable were found. The center of each sensor neuron was matched to the respective value for an input variable-class pair. Table 1 illustrates this. A similar approach towards the centering of receptive fields via clustering was taken in, even though Glackin et al. used fuzzy c-means clustering.

For the iris classification task, each output neuron represents different iris species. The network implements a winner-takes-all strategy where the first neuron to fire indicates to which class the input belongs, like it was the case for the logic network with two output neurons. Figure 5 shows the network architecture.

The parameters of the model were $a = 0.045$, $b = 0.2$, $c = -65$, and $d = 5$. Izhikevich, in his original paper (Izhikevich 2003), presents a set of parameters for different kind of real neurons. For this implementation, parameters *b* and *d* correspond to a *spike frequency adapting* neuron. Parameter *a* corresponds most closely to a neuron for subthreshold oscillations (the *a* of the subthreshold oscillations neuron is 0.5). Parameter *d* does not correspond directly to a neuron proposed in the original paper; however, many neurons have the paremeter *d* set to 4 or 6.

Despite not corresponding directly to a neuron presented in the original article, the model, nevertheless, presents realistic spiking behavior. Figure 6 shows voltage plots and raster plots for three different sets of neuron parameters from the original paper by Izhikevich (2003) and for the parameters used in this implementation. The neurons used in the experiments were improved Izhikevich neurons, since these were the neurons that were used for the neural network implementations, as well. For each demonstration only one neuron was used. The initial input voltage was 0, and it was increased by 1 for each iteration up to iteration 5. This is illustrated in Table 2. The timestep was 1 ms and the total time of simulation was 1,000 ms.

3.3 Training

The network was trained using a genetic algorithm in both cases. Genetic algorithms have been used in the past for spiking neural network optimization by (Bellatreche et al. 2006) and Glackin et al. (Glackin et al. 2011). A comprehensive review of training algorithms for spiking neural networks is provided by Kasinski and Ponulak (Kasinski and Ponulak 2006). Some choices beyond genetic algorithms/evolutionary strategies include SpikeProp (Bohte et al. 2002), statistical approaches (Pfister et al. 2003), linear algebra methods (Carnell and Richardson 2005), synfire chains (Sougne 2000), spike-based supervised Hebbian learning (Legenstein et al. 2005), and ReSuMe (remote supervision) (Ponulak 2005). A problem that genetic algorithms face is that they are very time consuming, making them unsuitable for online learning.

The algorithm comprised two populations that ranged from 50 to 100 members each, with a crossover ratio 0.6 and 1 elite. The algorithm terminated in 150 generations. Algorithm inputs were the weights of the neurons and none of the parameters of the model. Different tests were run with different parameters. The fitness criterion was the minimization of the classification error of the network.

For every logic gate, the network was provided with the four inputs and the corresponding outputs and the algorithm's success was counted as the percentage of outputs

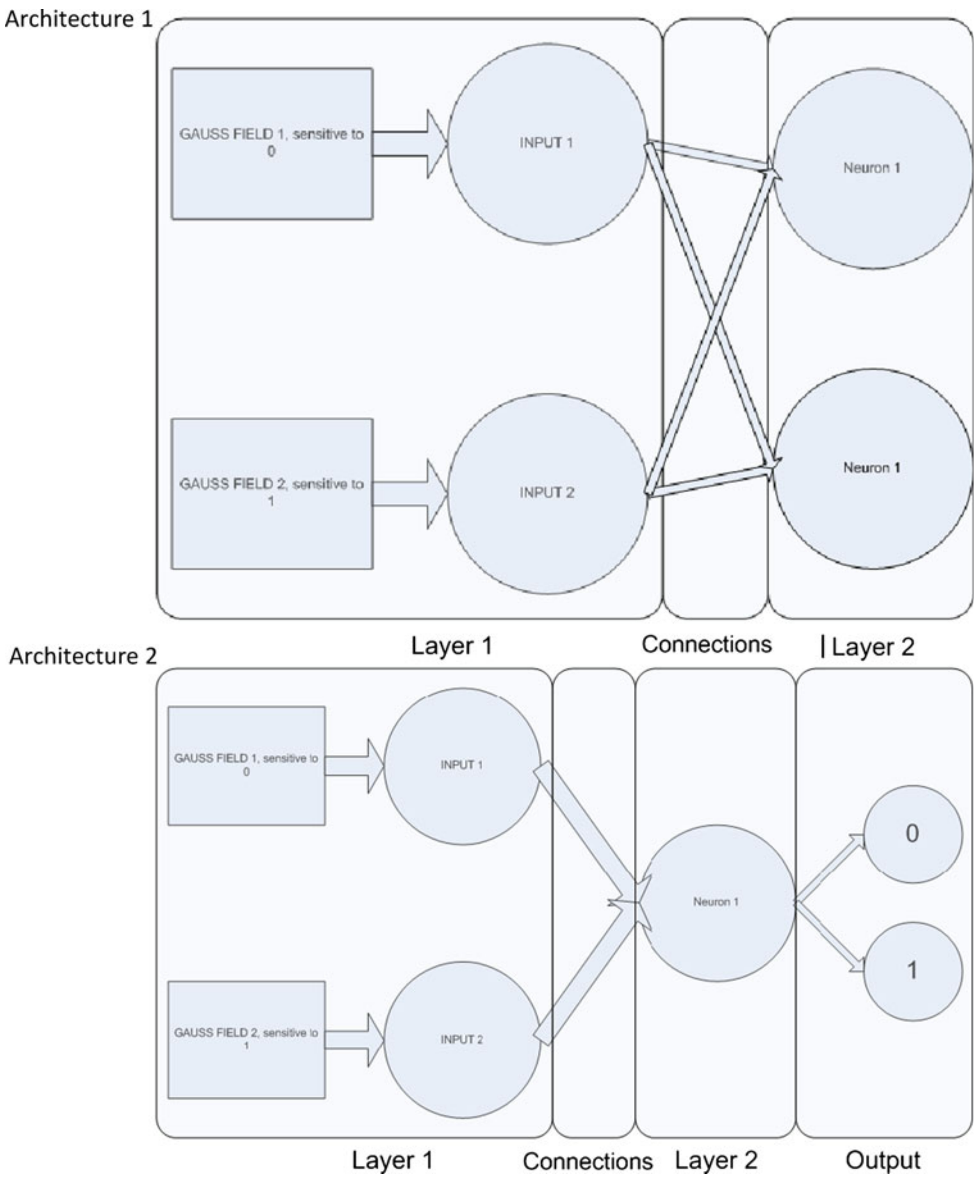

Fig. 4 Logic gates architectures

Table 1 Centers of sensor neurons

	Petal length	Petal width
Class 1	Sensor neuron 1 = 0.5587	Sensor neuron 4 = 0.5104
Class 2	Sensor neuron 2 = 0.0786	Sensor neuron 5 = 0.0600
Class 3	Sensor neuron 3 = 0.7740	Sensor neuron 6 = 0.8151

classified correctly. For the iris dataset 66% of the instances were sampled randomly (with resampling) for the training, while the network was then tested on 100% of the instances (to assess general accuracy) and on the unseen instances (in order to assess generalization).

All the tests were run on a PC with Intel Core Duo 2.66 MHz processor, 4 GB RAM, and Windows XP 32-bit Service Pack 3. The network was programmed in MATLAB 2009Rb. The genetic algorithm was implemented through Matlab's genetic algorithm toolbox.

4 Results and discussion

4.1 Logic gates results

Both network architectures specified for the logic gates managed to simulate them. However, there were many fluctuations in its performance, depending on the voltage transmitted through a spike and the amplifier. For the network with an *amplifier* of 30 and a *voltage_sent* of 60 mV, the networks could successfully simulate all logic gates. However, by setting both variables to 100, for example, the

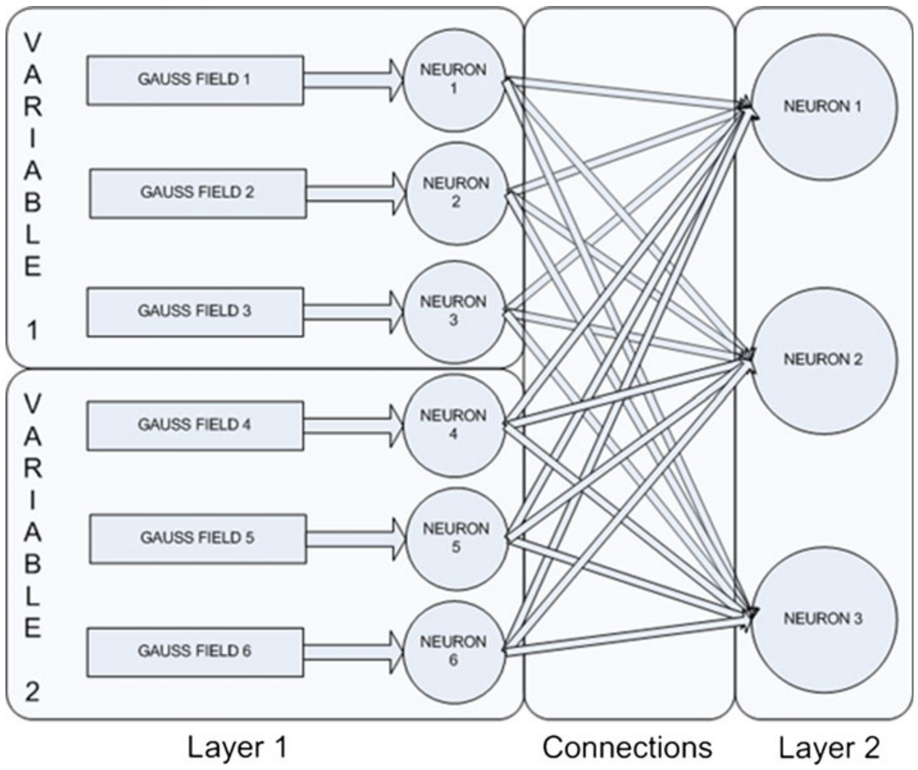

Fig. 5 Iris classification network

Table 2 Voltage versus time as used in the experiment

Time (ms)	0	1	2	3	4	5…1000
Voltage (mV)	1	2	3	4	5	5

networks took longer to train and it occasionally failed completely in the task. The training time required was usually less than ten generations, although in some cases the network did not manage to reach the global minimum.

A very interesting result was the use of rebound spiking by the network with one output neuron to correctly simulate the XOR gate for the input vectors [0, 1] and [1, 0]. Rebound spiking is a phenomenon in which a hyperpolarizing pulse leads to membrane depolarization. It has been accounted as the mechanism behind a variety of neural phenomena (Miltra and Miller 2007; Person and Perkel 2005; Super and Romeo 2011).

The two input neurons were connected to the output neuron with negative weights. Figure 7 shows the two neurons depolarizing simultaneously (in reality, there are two distinct curves but only one is visible because they happen to be equal at all points).

Figure 8 shows the negative input received by the output neuron that causes a small hyperpolarization leading to a rebound spike.

In further experiments, it was impossible to find a way to simulate the XOR gate with two input neurons and one output neuron, without using rebound spiking. This is a very interesting result because it demonstrates that increased biological realism can enhance the power of a network. Figure 9 illustrates this effect.

4.2 Iris dataset results

Concerning the iris dataset, the network achieved a total success rate of 96.44% (tested on all instances of the set after the training) with a 97% success during training and 96.3% success in the unseen data. Figure 10 shows an example of the evaluation process of the network. The result in this case was the output vector $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ in response to the input vector $\begin{pmatrix} 0.5085 \\ 0.5 \end{pmatrix}$, which indicates that the input belongs to the second class. The upper 3 neurons in the raster plot (numbers 7, 8 and 9) are the output neurons. As we see, the second neuron fires first from the output neurons, at the same time as the input neuron 4. In the first voltage plot, we see the voltage for the input layer. The first line corresponds to input neuron 4. In the second voltage plot the line that fires first corresponds to neuron 8. It seems that the input neurons sent an inhibitory signal to the other neurons, since they fall too much below their resting state voltage, before rebound spiking at about the 2nd ms. However, their spikes have no effect on the decision for the

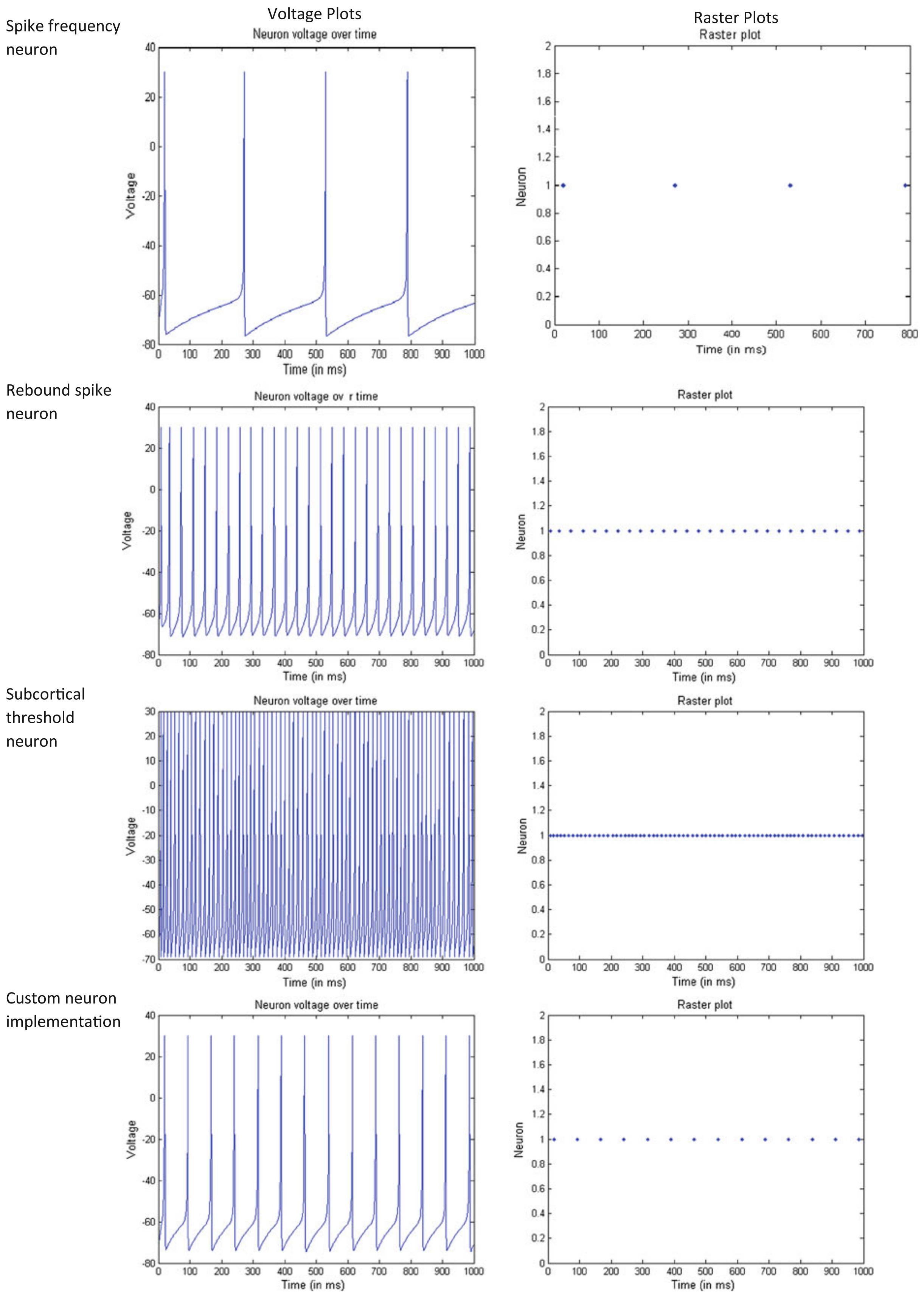

Fig. 6 Voltage plots and raster plots for various Izhikevich neuron types

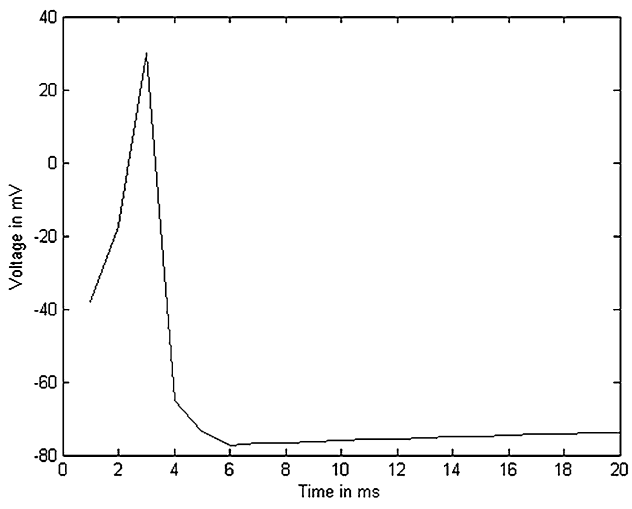

Fig. 7 Depolarizing neurons in XOR [0, 1] case

output since only the first output neuron to spike is significant for this implementation.

4.3 Speed

The network was also tested for speed of execution. Spiking neural networks are speculated to be more powerful than second-generation neural networks (Maass 1997) but speed remains always an issue, especially for real-time systems. While the network does not have a dedicated training algorithm and was trained using a genetic algorithm which is unsuitable for online applications, the speed of processing could be nevertheless important in real time applications.

A comparison test was made on the speed of execution between the set of networks trained for the logic gates and a set of multilayer perceptrons trained on the same inputs. The multilayer perceptrons were trained in the same version of Matlab as the one that was used for the coding of the Izhikevich network using the Levenerg–Marquadt algorithm. Each network had a hidden layer of two neurons since this was the minimum needed to converge to a solution.

The average time of simulation for 100 runs was calculated for both the network and the multilayer perceptrons. Their difference was approximated to 0.02 s in favor of the Izhikevich neural network with one output neuron. For the network with two output neurons, the difference was about the same (approximately 0.02 s).

A standard method of improving the speed of a neural network is pruning. In pruning the objective is to remove hidden neurons as long as the performance does not drop. Since the Izhikevich network does not make use of a hidden layer, there are two possibilities for improving performance. The first one is to use fewer sensor neurons for the coding of the input, something which is infeasible for the logic gates networks, since they use the minimum possible of two neurons.

The second possibility for improving the performance of an Izhikevich network lies in the way that the simulation is performed. The network is evaluated by running the model for a number of iterations, each iteration representing a millisecond of simulation. The highest number of iterations required to run the network is the number of iterations that the first output neurons needs in order to spike. Any iteration beyond that is useless for evaluation purposes. Since no network trained in the logic gates needed more than ten iterations for its output neuron to fire, the time variable was reduced from 20 to 10 and the test same test as above was conducted again for 100 runs. The difference was now

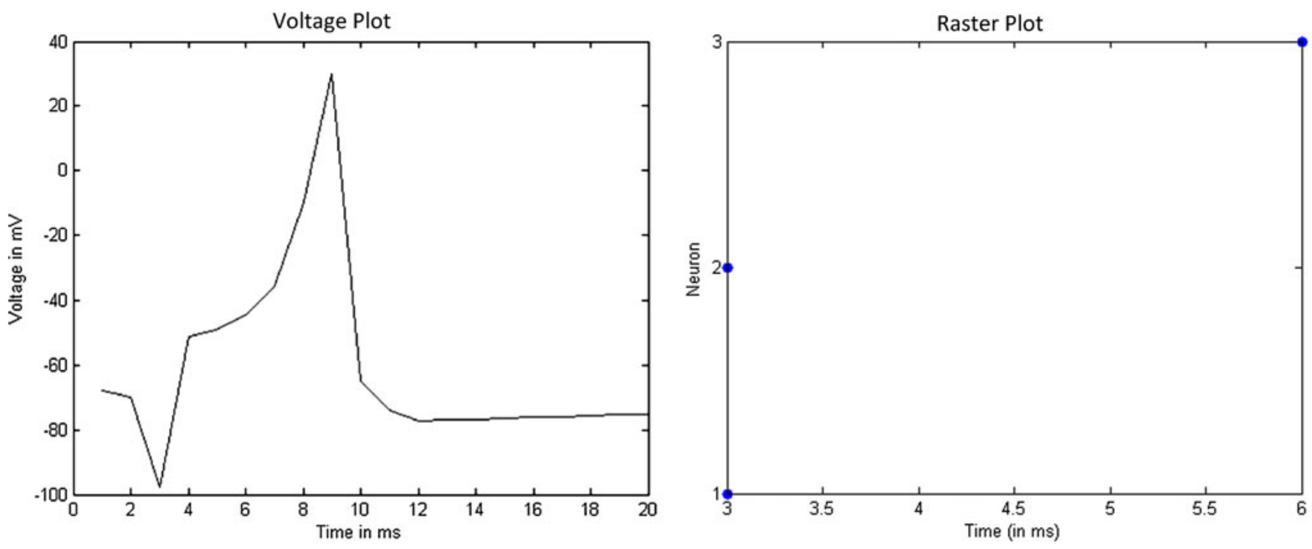

Fig. 8 Rebound spiking in output neuron of XOR network and Raster plot for XOR [0, 1] case

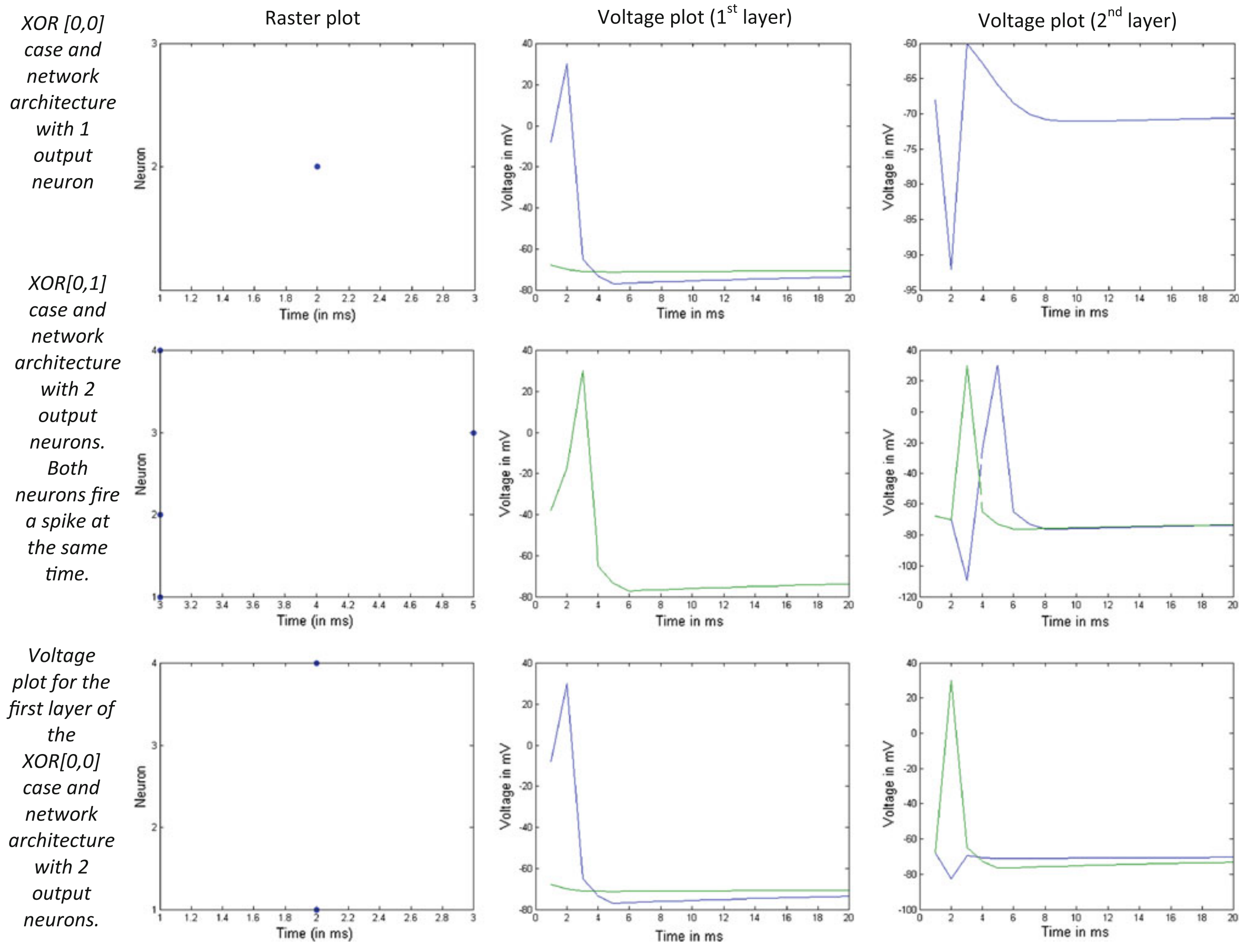

Fig. 9 Rebound spiking in XOR network

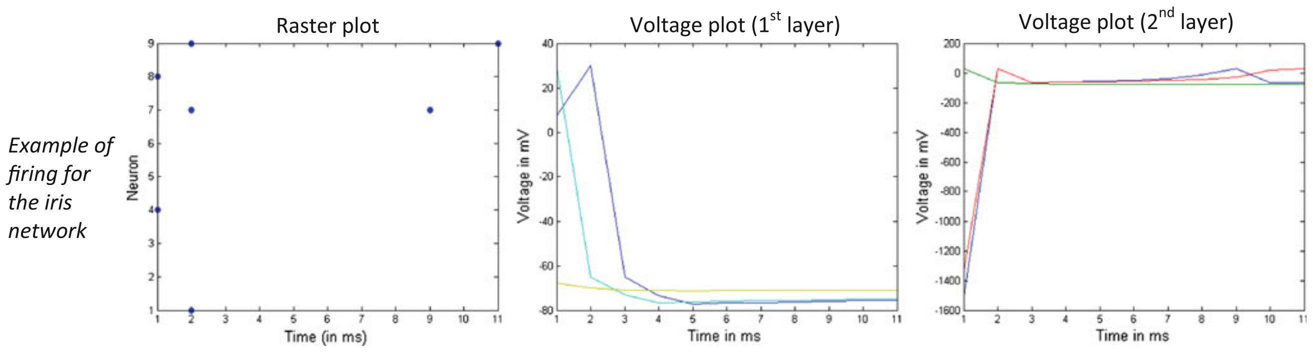

Fig. 10 Iris network depolarizing

0.0213 s in favor of the Izhikevich network for the network with one output neuron and about 0.02 s in favor of the network with two output neurons.

A speed test was also conducted for the iris dataset. A hundred multilayer perceptrons were trained to classify the iris dataset and their speed was measured on all of the instances of the dataset and an average time of execution was found. The multilayer perceptrons used a hidden layer of four neurons, since this was the lowest number of neurons that showed good (>90% accuracy) results. Similarly, the best Izhikevich network was run a hundred times in order to calculate its average speed of execution. Its iterations were set to 11. The final difference was 0.3930 s in favor of the Izhikevich network.

5 Conclusions

The functionality of a neural network with two layers of improved Izhikevich neurons that encodes information using Gaussian receptive fields was tested in this study. The tests proved that it is truly functional for classification tasks. The logic gates test proves that the network can simulate all gates, including nonlinearly separable ones (such as XOR), using only three neurons, as well as four neurons. The results in the iris classification task are close to that reported in past research. This network reached a total rate of success of 96.44%, while Bohte et al. (2001) have reported a successful classification rate of 96.1% on the iris dataset using a spiking neural network. Bellatreche et al. (2006) reported a success rate of 97.33%. It must be stated, however, that the network does not have a dedicated training algorithm, which could, possibly, further boost its performance, although a dedicated training algorithm might be more important for the speed of the training process and uses in real-time systems.

Note that the neuron model used is biologically realistic. This indicates that the network's use is not limited to machine learning tasks, but could also be used for research in computational neuroscience. Further research could indicate whether rebound spiking is indeed an important feature of the real neural networks used for classification in the mammal brain.

One more positive point was the speed of the network compared to a multilayer perceptron. Its speed can be further improved using automatic time pruning optimization coded in the evaluation or the training function and tested again.

Finally, future study could also indicate ways in which the proposed network could become more efficient for machine learning tasks. Possible ideas include the use of other clustering algorithms by the sensor neurons, or the creation of a training algorithm. Further improvements in its speed of execution could be made and possible applications for real-time systems could be tested, as well.

References

Bellatreche A, McGuire LP, McGiniity M, Wu Xiang Q (2006) Evolutionary design of spiking neural networks. New Math Nat Comput 2(3):237–253

Bohte S, Kok J, Poutre HL (2002) Error backpropagationin temporally encoded networks ofspiking neurons. Neurocomputing 48:17–37

Bohte SM, Poutre HL, Kok JN (2001) Unsupervised clustering with spiking neurons by sparse temporal coding and multi-layer RBF networks. IEEE Trans Neural Netw, XX

Carnell A, Richardson D (2005) Linear algebrafor time series of spikes. In Proceedings European Symposium on Artificial Neural Networks

Eurich CW, Wilke SD (2000) Multidimensional encoding strategy of spiking neurons. Neural Comput 12(7):1519–1529

Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7(2):179–188

Gerstner W, Kistler WM (2002) Spiking neuron models. Cambridge University Press, Cambridge

Glackin C, Maguire L, McDaid L, Sayers H (2011) Receptive field optimisation and supervision of a fuzzy spiking neural network. Neural Netw 24:247–256

Hodgkin AL, Huxley A (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500–544

Izhikevich EM (2003) Simple model of spiking neurons. IEEE Trans Neural Netw 14(6):1569–1572

Izhikevich EM (2004) Which model to use for cortical spiking neurons? IEEE Trans Neural Netw 15(5):1063–1070

Izhikevich E, n.d. Simple model of spiking neurons. http://www.izhikevich.org/publications/spikes.htm. Accessed 8 May 2011

Kasinski A, Ponulak F (2006) Comparison of supervised learning methods for spike time coding in spiking neural networks. Int J Appl Math Comput Sci 16(1):101–113

Lanella N, Back AD (2001) A spiking neural network architecture for nonlinear function approximation. Neural Netw 14(2001):933–939

Legenstein R, Naeger C, Maass W (2005) What can a neuron learn with spike-timing-dependent plasticity? Neural Comput 17(11):2337–2382

Maass W (1997) Networks of spiking neurons: the third generation of spiking neural networks. Neural Netw 10(9):1659–1671

Maass W, Bishop CM (2001) Pulsed neural networks. MIT Press, Cambridge

Miltra P, Miller R (2007) Normal and rebound impulse firing in retinal ganglion cells. Vis Neurosci 24(1):79–90

Minsky M, Papert S (1972) Perceptrons: an introduction to computational geometry, 2nd edn. The MIT Press, Cambridge

Nieder A, Dehaene S (2009) Representation of number in the brain. Ann Rev Neurosci 32:185–208

Person AL, Perkel DJ (2005) Unitary IPSPs drive precise Thalamic spiking in a circuit required for learning. Neuron 46(1):129–140

Pfister JP, Barber D, Gerstner W (2003) Optimal Hebbian learning: a probabilistic point of view. In: ICANN/ICONIP, 2003. Springer, Berlin

Ponulak F (2005). http://d1.cie.put.poznan.pl/~fp/. Accessed 25 Aug 2010

Rieke F, Warland D, Bialek W, de Ruyter van Steveninck R (1996) Spikes, exploring the neural code. The MIT Press, Cambridge

Shadlen MN, Newsome WT (1994) Noise, neural codes and cortical organization. Curr Opin Neurobiol 4(4):569–579

Sougne JP (2000) A learning algorithm for synfire chains. In: Connectionist models of learning, development and evolution, Liege, Belgium, 2000. Springer, Berlin

Super H, Romeo A (2011) Rebound spiking as a neural mechanism for surface filling-in. J Cogn Neurosci 23(2):491–501

Thorpe SJ, Delorme A, VanLurren R (2001) Spike-based strategies for rapid processing. Neural Netw 14(6–7):715–726

Valko M, Marques NC, Castellani M (2005) Evolutionary feature selection for spiking neural network pattern classifiers. In: Artificial intelligence, 2005. epia 2005. portuguese conference on. Covilha, 2005. IEEE Press, New York

Wang H (2009) Improvement of Izhikevich's neuronal and neural network model. In: International conference on information engineering and computer science, Wuhan, China, 2009. IEEE Press, New York

Wu Q et al (2007) Edge detection based on spiking neural network model. In: Lectures notes in computer science, 4682/2007, pp 26–34

Xiaoli T, Howard ME (2004) Data clustering via spiking neural networks through spike timing-dependent plasticity. In: International conference on artificial intelligence, 2004