The Role of Random Spikes and Concurrent Input Layers in Spiking Neural Networks

Janis Zuters

Faculty of Computing, University of Latvia, Raina bulvaris 19,
LV-1586 Riga, Latvia
janis.zuters@lu.lv

Abstract. In spiking neural networks (SNNs), unlike traditional artificial neural networks, signals are propagated by a ‘pulse code’ instead of a ‘rate code’. This results in incorporating the time dimension into the network and thus theoretically ensures a higher computational power. The different principle of operation makes learning in SNNs complicated. In this paper, a Hebbian-learning based learning algorithm, introduced by the author earlier, is further developed through applying random spikes in order to guarantee frequent firings of neurons for each layer. In addition, the impact of concurrent input layers to the learning ability is shown. The introduced ideas have been illustrated with experimental results.

Keywords: Spiking Neural Networks, Unsupervised Learning, Randomness.

1 Introduction

A neural network is a computational model which is made up of relatively simple interconnected processing units which are put into operation through a learning process. Neural networks are useful tools for solving many types of problems. These problems can be characterised as mapping (including pattern association and pattern classification), clustering, and constraint optimisation. There are several neural network models which are available for each type of problem [3].

Typically, the neurons of a network communicate via a ‘rate code’, i.e., by transmitting “pure” continuous values irrespective of the timings of the signals. Work with traditional models is usually arranged in steps – during each step exactly one pattern is classified or recognized. Considering this, any temporal information, if present in the problem, or interconnectedness of patterns is out of area of responsibility of a neural system, so it should be processed by a superior system.

When spiking neural networks appeared, the neural network models became “ready” to be classified into three generations.

The first generation of artificial neural networks consisted of McCulloch-Pitts threshold neurons [9], where a neuron sends a binary signal if the sum of its weighed incoming signals rises above a threshold value.

The second generation neurons use continuous activation functions instead of threshold functions to compute the output signals.

In contrast to classical models, spiking neural networks, or the third generation neural networks, are designed to make use of the temporal dynamics of neurons. Here, communication between neurons involves a ‘pulse code’ – the exact timing of the transmitted signal (i.e., the spike) is of key importance, but the strength of the signal is not considered. SNNs are believed to be very powerful, but their operation is more complicated and their application is not as obvious as is the case with “ordinary” neural networks. Nevertheless SNNs have been focused as an area of much research.

In [12], the method of ‘adaptive layer activation’ has been introduced for Hebbian-like learning in SNNs. The current paper further develops this approach (a) by adding injections of random spikes to induce the learning process, and (b) by extending network structure with concurrent input layers of neurons in order to enrich presence of temporal templates in the input structure represented by axonal delays.

2 Spiking Neural Networks

Spiking neural network is a model of artificial neural networks, in which signals are propagated via spikes instead of rates (i.e., values). SNNs are potentially very powerful, still complicated to operate.

2.1 General Description

Each neuron in an SNN produces a train of spikes (i.e., it fires), and the operation of the neuron *i* is fully characterized by the set of firing times [4] (Fig. 1):

$$F_i = \left\{ t_i^{(1)}, \ldots, t_i^{(n)} \right\} , \tag{1}$$

where $t_i^{(n)}$ is the most recent spike of neuron *i*.

One of the most frequently applied spiking neuron models is the ‘integrate-and-fire’ model [1][2]. In this model, input signals (both actual and recent) from a neuron are combined together in order to achieve enough activation for the neuron to fire.

Neuron *i* is said to fire if its activation state u_i reaches the threshold ϑ. At the moment when the threshold is crossed, a spike with the firing time $t_i^{(f)}$ is generated. Thus Eq. (2) can be clarified with Eq. (3) [4]:

$$F_i = \left\{ t_i^{(f)} ; 1 \le f \le n \right\} = \left\{ t \mid u_i(t) = \vartheta \right\} , \tag{2}$$

where the activation state u_i is given by the linear superposition of all contributions:

$$\mathrm{u}_i(t) = \eta_i (t - \hat{t_i}) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in F_j} w_{ij}\, \varepsilon_{ij} (t - t_j^{(f)}) , \tag{3}$$

where Γ_i is the set of presynaptic neurons to neuron i; $\hat{t}_i$ is the last spike of neuron i; the function η_i takes care of refractoriness after a spike is emitted; the kernels ε_{ij} model the neurons response to presynaptic spikes.

$$\eta_i(s) = \begin{cases} -\vartheta \exp(-\frac{s}{\tau}); & s > 0 \\ 0; & s \le 0 \end{cases}, \tag{4}$$

where τ is a time constant.

$$\varepsilon_{ij}(s) = \begin{cases} \exp(-\frac{s-\Delta_j^{ax}}{\tau_m}) - \exp(-\frac{s-\Delta_j^{ax}}{\tau_s}); & s-\Delta_j^{ax} > 0 \\ 0; & s-\Delta_j^{ax} \le 0 \end{cases}, \tag{5}$$

where τ_s and τ_m are time constants, and Δ_j^{ax} is the axonal transmission delay for presynaptic neuron j.

Axonal transmission delay (or simply, axonal delay) is an essential part of a spiking neuron (Fig. 1). It delays transmitted signals for the subsequent neurons, therefore it realizes a kind of short-time memory. Axonal delays are usually initialized in random with creation of an SNN.

Eqs. (4) and (5) have been adapted from [4], [8]. In Fig 2, the nature of activation of spiking neurons is visualized. The algorithm in Fig. 3 gives a general overview of operation of a spiking neuron.

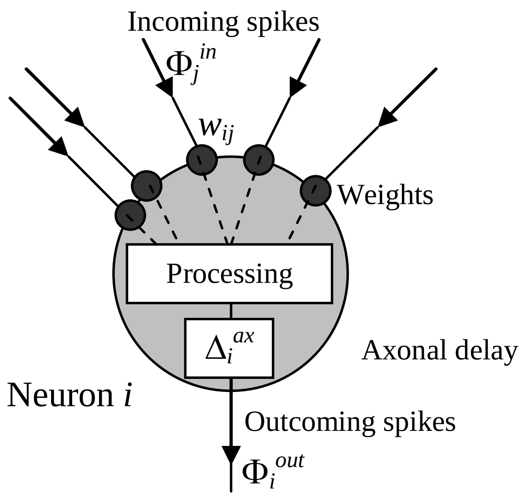
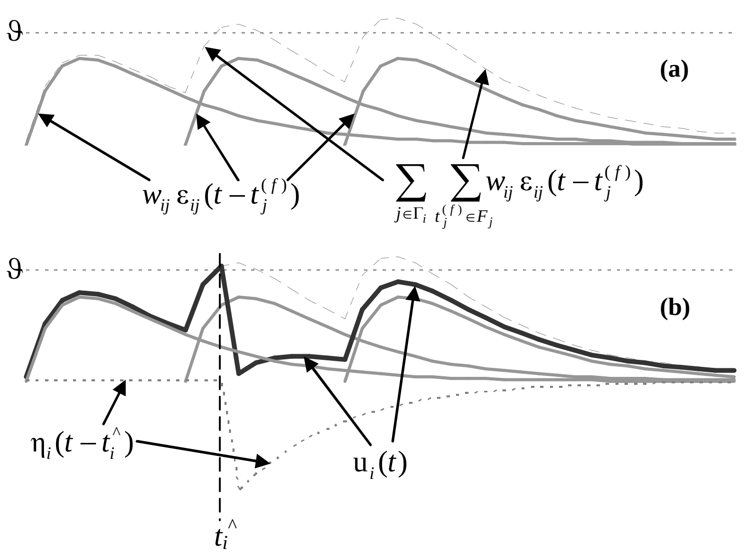


Fig. 1. A single spiking neuron. The small filled circles w_{ij} denote synaptic weights. Φ_j^{in} denotes an incoming spike train j. The neuron produces the spike train Φ_i^{out}, delayed by axonal delay Δ_i^{ax}. (Adapted from [7])



ϑ
(a)
$w_{ij}\,\varepsilon_{ij}(t-t_j^{(f)})$
$\sum_{j\in\Gamma_i}\sum_{t_j^{(f)}\in F_j} w_{ij}\,\varepsilon_{ij}(t-t_j^{(f)})$
ϑ
(b)
$\eta_i(t-\hat{t}_i)$
$u_i(t)$
$\hat{t}_i$

Fig. 2. Graphic interpretation of activation state u_i (Eq. (3)) a) Activation $u_i(t)$ is a superposition of responses of incoming spikes and the weights w. The kernels ε_{ij} describe the response of u_i to presynaptic spikes. (b) Activation $u_i(t)$ reaches the threshold ϑ at time $t_i^{(f)}$. The reset of activation is immediately performed in order to avoid continuation of spiking over a certain period. This refractoriness is achieved by adding a kernel $\eta_i(t–t_i^{(f)})$.

The pure integrate-and-fire model is very simple, but the fire sequences appear oversimplified and unable to reproduce rich and irregular nature of spiking of biological systems. In order to avoid regularities and deterministicities, numerous extensions of the model, as well as intermediate ones with other models have been proposed [6][10].

2.2 Learning in Spiking Neural Networks

If supervised learning methods greatly predominate in “non-spiking” neural networks, then the situation with SNNs is quite different.

Hebbian learning ([5]) is a typical learning method with SNNs. The simplest way to describe Hebbian learning is that the weight modifications depend on the level how pre- and postsynaptic activities match (Eq. (6)).

Operation of traditional neural networks is carried out through propagating of values, and the process of propagation is occurring without any constraints – the neural activities influence value strengths, while the propagation is occurring regardless these values.

$$\Delta w_{ij} = c_{ij} y_i y_j \ . \qquad (6)$$

With SNNs, spikes are the ones, which are being propagated over the network. Unlike traditional neural networks, there are conditions required for the process to occur, since spikes are generated only at certain conditions. This is a noticeable practical problem.

```
Algorithm run_spiking_neuron (t)
    t – time step of operation
    $\vartheta$ – threshold to be reached by activation of the neuron
    $\Delta^{ax}$ – axonal delay for the neuron
Begin
    Compute activation u by processing timings of incoming spikes and weights (Eq. (3))
    If $u \geq \vartheta$ Then  % activation reaches threshold
        Generate spike at time step t
        Delay the spike by $\Delta^{ax}$ to become 'incoming' spike for subsequent neurons (See Fig. 1)
```

Fig. 3. Operation of a spiking neuron at fixed time step.

2.3 The Learning Method Using Adaptive Layer Activation

In [12], a Hebbian rule-based method for learning in SNNs is proposed. It is built considering the following predefined principles (not mandatory for SNNs in general):

1) Weight changes are performed only after the neuron has just fired. Thus an additional mechanism is required to ensure spikes to be generated;
2) Weight changes are performed only upwards (as an award for a correct timing of a spike, Eq. (7)). Thus an additional mechanism is required to regulate weight values to avoid them all to become too big.

$$\Delta w_{ij} = \alpha(w' - w_{ij}) \sum_{t_j^{(f)} \in F_j} \varepsilon_{ij}(t - t_j^{(f)}) \ , \tag{7}$$

where α is the learning rate; w' is a constant representing maximum allowed weight value, ε_{ij} – the neurons response to presynaptic spikes.

```
Algorithm run_spiking_neuron_adaptive (t, β)
    t – time step of operation
    $\beta$ – activation acceleration factor of the layer
    $\vartheta$ – threshold to be reached by activation of the neuron
    $\Delta^{ax}$ – axonal delay for the neuron
Begin
    Compute activation u by processing timings of incoming spikes and weights (Eq. (3))
    If $\beta \times u \geq \vartheta$ Then  % (accelerated) activation reaches threshold
        Generate spike at time step t
        Delay the spike by $\Delta^{ax}$ to become 'incoming' spike for subsequent neurons
```

Fig. 4. Operation of a spiking neuron with 'adaptive acceleration factor'. Activation acceleration factor β determines 'sensitivity' of the neuron.

These conditions were obeyed through introducing the following key innovations (See the algorithm in Figs. 4 and 5):

1) Adjustment of the average activation state of neurons in each layer in order to ensure a determined average rhythm of spikes within a layer;
2) Maintenance of the fixed average volume of synaptic weight in each neuron in order to avoid a situation in which all the weights converge to maximum possible value.

```
Algorithm learning_adaptive_layer_activation (t)
    t – time step of operation
    β – activation acceleration factor of the layer (in the beginning is set to 1.0)
    ϑ – threshold to be reached by activation of the neuron
    run_spiking_neuron_adaptive – algorithm to operate one neuron with ‘adaptive acceleration factor’
Begin
    Forall neurons i in the layer
        Call run_spiking_neuron_adaptive (t, β)  % run neuron (See the algorithm in Fig. 4)
        If neuron i has just fired Then
            Forall weights j in the neuron
                Increase weight j proportionally to the respective incoming activation (Eq. (7))
            Decrease uniformly all the weights to keep a fixed average weight value of the neuron
    Update layer statistics about neuron activity in the layer (frequency of spikes)
    If neurons activity is “strong” (according to the statistics) Then
        Decrease β
    Else
        Increase β
```

Fig. 5. Learning with method of ‘adaptive layer activation’ for one layer. In addition to neuron weights, also the activation acceleration factor β undergoes learning.

3 Random Spikes and Concurrent Input Layers

Although spiking neural networks are believed to be very powerful, they are complicated to operate, and training them is still a challenging issue.

In this section, two different ideas have been proposed to operate a kind of SNNs (built according to the principles stated in the previous section):

1) Injecting random spikes to induce learning of an SNN;
2) Extending network structure with concurrent input layers.

The proposed methods help to realize a way, how an SNN is set to operate. The experimental results show these methods to help to successfully overcome various problems to build SNNs.

3.1 The Learning Method Using Random Spikes

Applying effects of randomness is a method already used to extend integrate-and-fire neural networks [11].

This section introduces the approach of using random spikes in learning with SNNs. The current research is continuation of the previous work, where the following principle in operation of an SNN was assumed: learning within a neuron is performed only after the neuron has just fired, i.e., has just generated a spike. Since normally firing is accomplished only when activation of the neuron reaches a certain threshold value, the problem to overcome is – how to provide a neuron with conditions to frequently fire in order to undergo learning.

In the previous research, this problem was proposed to solve by using the ‘activation acceleration factor’. The main idea of it is that the requirements of firing are being turned down, if neural activity in the layer is insufficient. Such a situation is typical at the very beginning of the learning process.

In the new approach, this principle is substituted by generating random spikes with a certain probability. This principle ensures the minimum activity of a neuron, and by this, also the learning process to happen.

Roughly speaking, the new method has the effect similar to the one of the method of ‘adaptive layer activation’, yet it is simpler and so much easier to implement.

The algorithm in Fig. 6 unveils the principle of random spikes used in operation of a neuron.

The algorithm in Fig. 7 shows the learning algorithm for a whole layer using this principle. Adjusting weights of neurons remains the same as in the previous approach.

```
Algorithm run_spiking_neuron_random_spikes ($t$, $\rho$)
    $t$ – time step of operation
    $\rho$ – random spike probability ($0 \le \rho << 1$)
    $\vartheta$ – threshold to be reached by activation of the neuron
    $\Delta^{ax}$ – axonal delay for the neuron
    get_random_value – generates a random value from interval 0..1
Begin
    Compute activation $u$ by processing timings of incoming spikes and weights (Eq. (3))
    fire ← False
    If u $\ge \vartheta$ Then  % activation reaches threshold
        fire ← True
    Elseif get_random_value < $\rho$ Then  % probability to generate random spike
        fire ← True
    If fire $\ge \vartheta$ Then
        Generate spike at time step $t$
        Delay the spike by $\Delta^{ax}$ to become ‘incoming’ spike for subsequent neurons
```

Fig. 6. Operation of a spiking neuron with ‘random spikes’. Spikes are generated also in random to guarantee a certain frequency.

```
Algorithm learning_random_spikes ($t$, $\rho$)
    $t$ – time step of operation
    $\rho$ – random spike probability ($0 \le \rho << 1$)
    $\vartheta$ – threshold to be reached by activation of the neuron
    run_spiking_neuron_random_spikes – algorithm to operate one neuron with ‘random spikes’
Begin
    Forall neurons $i$ in the layer
        Call run_spiking_neuron_random_spikes ($t$, $\rho$)  % run neuron (See Fig. 5 for the algorithm)
        If neuron $i$ has just fired Then
            Forall weights $j$ in the neuron
                Increase weight $j$ proportionally to the respective incoming activation (Eq. (7))
            Decrease uniformly all the weights to keep a fixed average weight value of the neuron
```

Fig. 7. Learning with the method of ‘random spikes’ for one layer.

3.2 Using Concurrent Input Layers in Network Structure

This section proposes an SNN architecture with concurrent input layers to enhance the ability of a neural network to detect regularities in an input.

Commonly a neural network has a linear feed-forward structure. This also holds for SNNs (Fig. 8). For technical reasons, ‘input layers’ are usually not “true” layers: input values to the whole neural network are directly put to the outputs of the input layers, so they don’t compute anything (See the algorithm in Fig. 9).

With SNNs, the input layer can additionally have the role of delaying input signals. As axonal delays are predefined in random, input layers of SNNs provide with a kind of spatial representation of temporal input information.

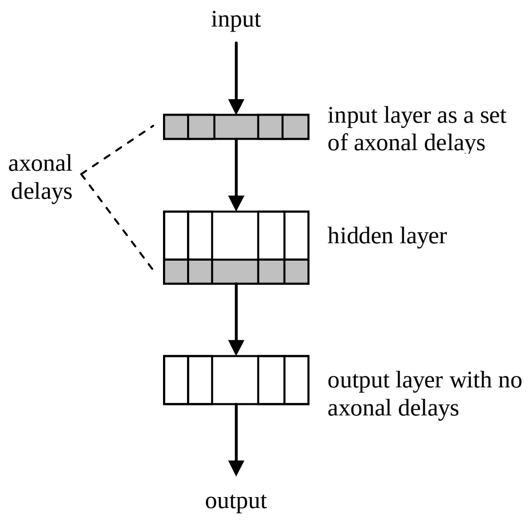

input
input layer as a set of axonal delays
axonal delays
hidden layer
output layer with no axonal delays
output

Fig. 8. A simple example of a feed-forward linear architecture of a spiking neural network. Connections between layers are depicted as arrows, and they denote all-to-all connection between the neurons of both layers.

```
Algorithm run_neural_network
Begin
    Do many times
        Fetch next input into $X$
        Set $X$ to outputs of the input layer (in SNNs, directly generate spikes according to $X$)
        Process the rest of layers in a row
```

Fig. 9. Running a neural network with linear structure and a single input layer.

Instead of a single input layer, several concurrent input layers are proposed by the author. Since axonal delays of neurons are predefined random values, bunch of input layers provide with several kinds of spatial representations of input signals. (Fig. 10)

The idea is very simple: at each iteration, the input is duplicated for all the input layers (see the algorithm in Fig. 11).

By this, a network receives several varieties of partly delayed input signals, so concurrent input layers serve as a grid of first level perception.

As the experimental results show, this structural feature leads to faster and better learning in an SNN.

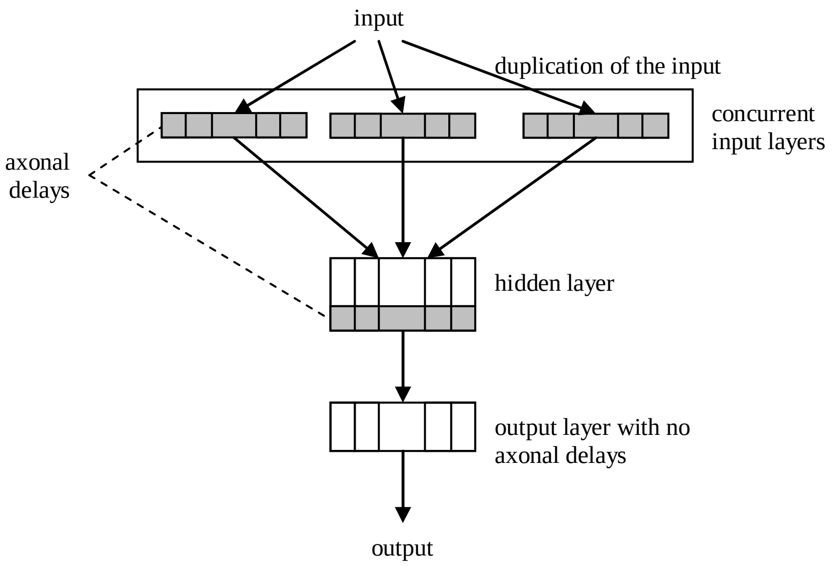


Fig. 10. A spiking neural network architecture with concurrent input layers.

```
Algorithm run_neural_network_concurrent_input_layers
Begin
    Do many times
        Fetch next input into X
        Forall L in input layers Do
            Set X to L
        Process the rest of layers in a row
```

Fig. 11. Running a network with concurrent inputs.

4 Experimental Work

To show the introduced methods working, a large experimental work was accomplished using original software for that. Each experiment was run to solve a task of detecting one kind of temporal patterns in an input stream. Experiments were carried out using several configurations of SNNs to be able to compare them in order to mark out the effect of the two ideas introduced in this paper.

4.1 Setup of the Experiments

The proposed learning method was tested on the detection of reiterative temporal patterns in an input stream. A trained neural network should be able to detect such patterns by giving a spike in output after the pattern is encountered.

The testing environment produces three binary signals at each discrete time moment *t* and passes them to the input of the neural network. In general, signals are generated at random, but sometimes patterns from a predefined store are interwoven into them. There are six possible temporal patterns in the store (Figure 12). The proportion of 1s and 0s in the random part equals the one in the patterns – 20%.

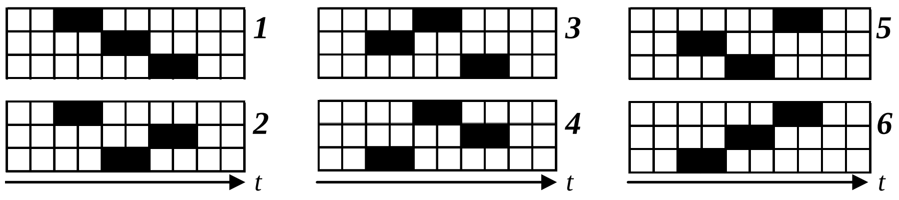


Fig. 12. The 6 temporal patterns used in experiments: width=3 signals, length=10 times; black squares denote ‘1’ or ‘spike’, but white squares – ‘0’ or ‘no spike’

At each separate experiment, one of the six patterns was occasionally put into input stream (7 patterns of size 10 within 200 time steps in average), so the proportion of patterns in the input stream was 7×10÷200=35%.

Output spikes of trained networks were analyzed of how they match patterns in the input stream. Pattern recognition rate was measured as a rate of responses of a network to input patterns at a fixed position with regard to the end of a pattern. Positions tried were of the interval [-5, +5]. As for instance, pattern recognition rate of 90% means that we have a position *p*, one for which is true that for 90% of patterns of the input stream, a spike of the neural network is encountered at the position *p* with regard to patterns. In addition, the rest of spikes, false positive ones, also were counted (Fig. 13.). False positive rate refers to the proportion between false positive spikes and actual patterns in the input stream.

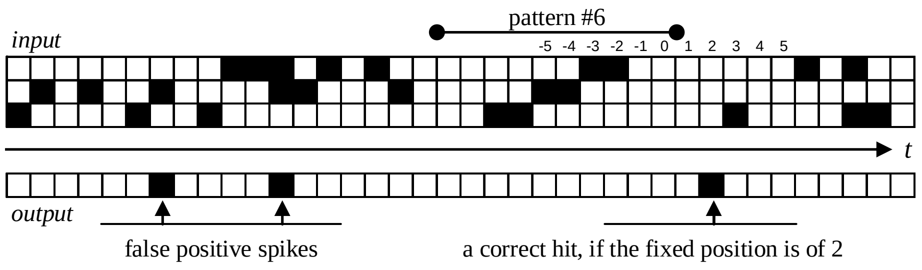


Fig. 13. An excerpt of an input stream, and examples of responses of the network to it. Here the value of position can be chosen as 2, so the pattern recognition rate is computed as 100% (as the only pattern is recognized), and the false positive rate is 2 (two false positive spikes against one pattern).

To show the introduced methods working, a total of 15,000 experiments was conducted. Each experiment consisted of 20,000 iterations in the learning phase, and then 200 iterations in the recall phase, the results of which were recorded and further analyzed.

Five different configurations of SNNs were used (Table 1) according with the general structure shown in Fig. 10 and the algorithm in Fig 7. Different configurations of SNNs were used to be able to compare them in order to mark out the effect of the two ideas introduced in this paper: random spikes and concurrent input layers.

The functionality of SNNs was built as described in sections 2.1 and 3; with random spike probability – 4%, threshold – 0.3, and axonal delays randomly set from interval [0, 4].

Along analyzing direct results of the experiments, also side affects were determined, those of ‘false positives’ and position of pattern detection.

Table 1. Network configurations used in the experimentation.

Name of the configuration	Usage of random spikes	Network architecture description (for all types, there are 3 inputs and 1 output)
A. Random=N 4×3-12-1	No	4 concurrent inputs, 12 hidden neurons
B. Random=N 8×3-24-1	No	8 concurrent inputs, 24 hidden neurons
C. Random=Y 4×3-12-1	Yes	4 concurrent inputs, 12 hidden neurons
D. Random=Y 1×3-24-1	Yes	no concurrent inputs, 24 hidden neurons
E. Random=Y 8×3-24-1	Yes	8 concurrent inputs, 24 hidden neurons

4.2 Analysis of the Results

Fig. 14 shows the general overview of experimental results for the five configurations of SNNs.

The configurations C, D, and E show the positive effect of using the random spikes to recognition.

The configuration E shows the impact of additional concurrent input layers, while D and E show the effect of both concurrent input layers and a sufficiently sized hidden layer.

The configuration D (in contrast to C) might signal a large hidden layer to bring the same effect as concurrent input layers do, however it suffers from instability of detection (see Fig. 16).

The best results were obtained by obeying all the following conditions:

1) Using random spikes;
2) Using more concurrent input layers;
3) Using a hidden layer with a sufficient size.

The main results demonstrate SNNs to be successfully trained to detect temporal patterns. During experiments, there also were undesirable side effects, which should be reduced during further research.

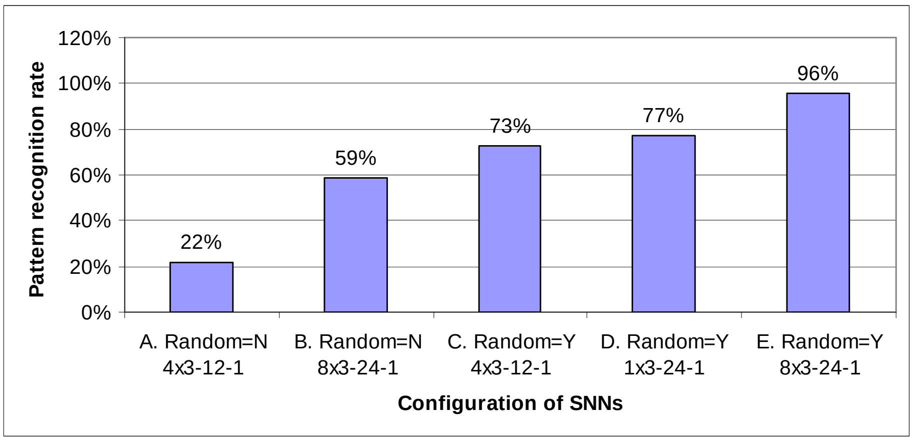


Fig. 14. Pattern recognition rate obtained for various configurations of SNNs. For the best results, random spikes and concurrent input layers should be used.

The most noticeable one is that of false positives (Fig. 15). The results show that, along successful detection of input patterns, a lot of redundant spikes are additionally generated by a neural network.

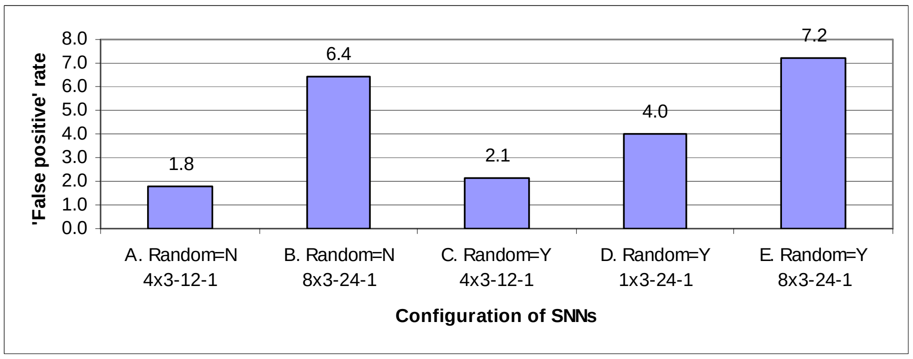


Fig. 15. High ‘false positive’ rates signal the algorithms to be further improved and/or better configured/setup.

Fig. 16 shows how the architecture of the network influences the length of detected patterns.

The networks of the configurations C and D detect most patterns at earlier positions (See also Fig. 13), while the stronger networks (E) typically detect patterns almost at the end position (-1), moreover the position of detection was much more determined (50% for the positions of highest occurrence). The configuration D is especially unstable, so it has such a good recognition rate (Fig. 14) just because of the methodology used.

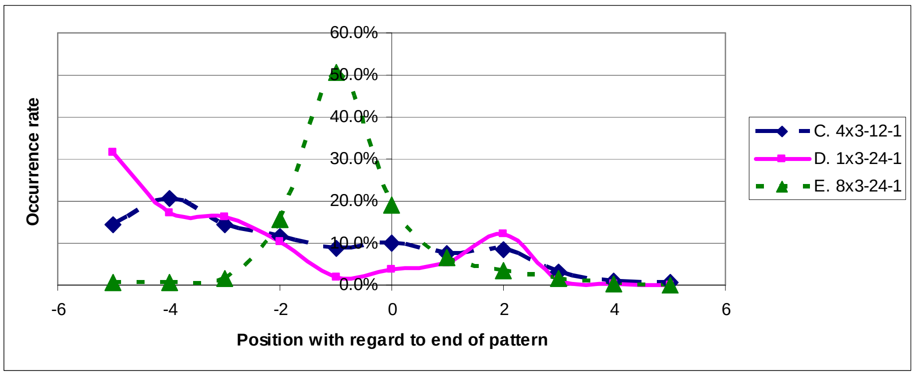


Fig. 16. Occurrence distribution of positions (with regard to the ends of patterns) at which the best pattern recognition rate was obtained (only for the three configurations with random spikes). The configuration E shows the most stable results.

5 Conclusion

Spiking neural networks are believed to be very powerful; for all that they are comparatively complicated, especially in terms of learning.

In this paper, two ideas have been introduced to enhance learning and operation of a variety of SNNs (characterized by the principle that learning within a neuron is performed only after the neuron has just fired):

1) Using random spikes to additionally stimulate the neuron;
2) Using several concurrent input layers to better preprocess the input.

The principle of random spikes ensures the minimum activity of a neuron, and by this, also the learning process to happen. Experimental results show it essential for a good recognition rate.

Although the experiments confirm the significance of hidden layers with sufficient sizes, yet for the best results, concurrent input layers are very important.

The proposed approach furnishes an effect to learning of SNNs similar to one in the previous research; nevertheless the new methods are less complicated and have demonstrated to be more stable.

References

1. Burkitt, A.N.: A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. In: Biological Cybernetics, Volume 95, Number 1, 2006, pp. 1-19. Springer (2006)
2. Burkitt, A.N.: A review of the integrate-and-fire neuron model: I. Inhomogeneous synaptic input and network properties. In: Biological Cybernetics, Volume 95, Number 2, 2006, pp. 97-112. Springer (2006)
3. Fausett, L.: Fundamentals of Neural Networks. Prentice-Hall, Inc. (1993)

4. Gerstner, W.: Spiking Neurons. In: Maass, W., Bishop, C.M. (Eds.) Pulsed neural networks. MIT Press (1999)
5. Hebb, D.: The organization of behavior. John Wiley and Sons, New York (1949).
6. Izhikevich, E.M.: Simple Model of Spiking Neuron. In: IEEE Transactions on Neural Networks, Volume 14, Number 6, November 2003. IEEE (2003)
7. Kempter, R., Gerstner, W., van Hemmen, L.: Hebbian learning and spiking neurons. Physical Review E, 59:4498-4514. (1999)
8. Maass, W.: Computing with Spiking Neurons. In: Maass, W., Bishop, C.M. (Eds.) Pulsed neural networks. MIT Press (1999)
9. Maass, W.: Networks of spiking neurons: the third generation of neural network models. In: Transactions of the Society for Computer Simulation International, vol. 14, issue 4 (December 1997), pp. 1659-1671. (1997)
10. Salerno, M., Casali, D., Constantini, G., Carota, M.: Fully Asynchronous Neural Paradigm. In: Proceedings of the 15th IEEE Mediterranean Electrotechnical Conference (MELECON 2010), pp. 1039-1043. IEEE (2010)
11. Seung, H.S.: Learning in Spiking Neural Networks by Reinforcement of Stochastic Synaptic Transmission. In: Neuron, Volume 40, Number 6, 18 December 2003, pp. 1063-1073. Elsevier (2003)
12. Zuters, J.: Spiking Neural Networks to Detect Temporal Patterns. In: Haav, H.M., Kalja A. (eds.) Frontiers in Artificial Intelligence and Applications, vol. 187, Databases and Information Systems V - Selected Papers from the Eighth International Baltic Conference, DB&IS 2008, pp. 131-142. (2009)