Learning with Adaptive Layer Activation in Spiking Neural Networks

Janis ZUTERS[1]
Computing Department, University of Latvia, Latvia

Abstract. Spiking neural network (SNN) is a neural model, where the communication between two neurons is performed by a 'pulse code', i.e., computation considers timings of the signals (spikes) whilst ignoring the strengths. It has been theoretically proven that a neural network composed of spiking neurons is able to learn a large class of target functions. In this paper, an unsupervised learning algorithm for the spiking neural network is introduced. The algorithm is based on the Hebbian rule, additionally using the principle of adaptation of layer activation level in order to guarantee frequent firings of neurons for each layer. The proposed algorithm has been illustrated with experimental results on detection of temporal patterns in an input stream.

Keywords. spiking neural networks, unsupervised learning, adaptive activation, temporal patterns

1. Spiking Neural Networks

1.1. Pulse Code Instead of Rate Code

Neural network refers to a computational model made up of relatively simple interconnected processing units, which are put into operation through a learning process. Typically, neurons of a network communicate by a 'rate code', i.e. through transmitting "pure" continuous values irrespective of the timings of the signals. Work with such models is usually arranged in steps: during each step exactly one pattern is classified or recognized, and no temporal information is directly processed by the neural network.

In contrast to classical models, SNNs are designed to get use of temporal dynamics of neurons. Communication between neurons there is realized by a 'pulse code': exact timing of the transmitted signal (i.e., spike) is of key importance, while the strength of the signal is not considered.

Each neuron produces a train of spikes (i.e., it fires), and the operation of a neuron i is fully characterized by the set of firing times [2] (see Figure 1):

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

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

[1] Janis Zuters, Computing Department, University of Latvia, Raina bulvaris 19., LV-1050 Riga, Latvia; E-mail: janis.zuters@lu.lv

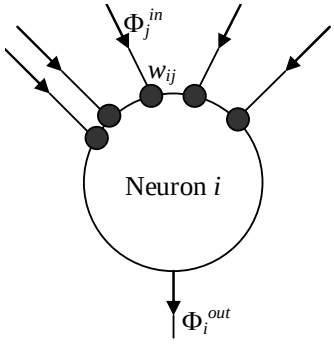

Figure 1. Single spiking neuron. Small filled circles (w_{ij}) denote synaptic weights, Φ_j^{in} denotes incoming spike train j; the neuron produces spike train Φ_i^{out} (adapted from [5])

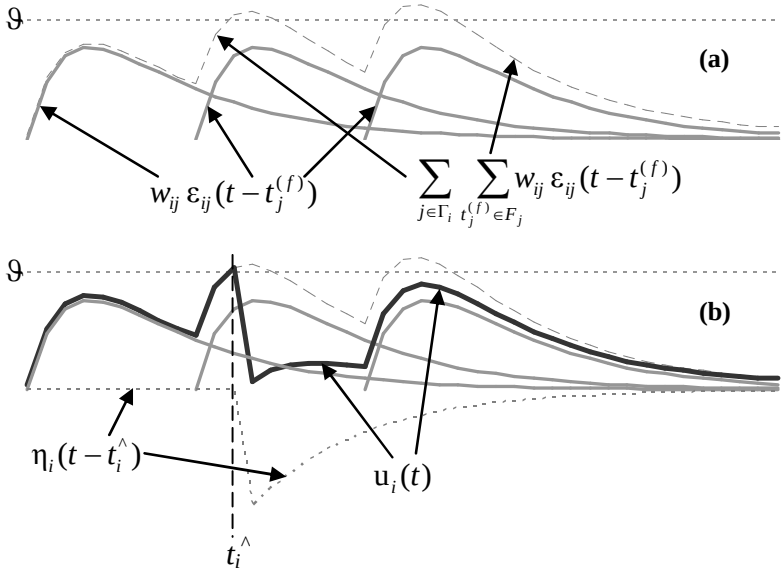

Figure 2. Graphic interpretation of activation state u_i (Eq. (3))

1.2. Operation of a Spiking Neuron

Neuron i is said to fire, if its activation state u_i reaches a threshold ϑ. A the moment of threshold crossing, a spike with firing time $t_i^{(f)}$ is generated (see Eq. 2)).
Thus Eq. (1) can be clarified with Eq. (2) [2]:

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

where activation state u_i is given by the linear superposition of all contributions (see also Figure 2 for graphic interpretation).

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

where Γ_i is the set of presynaptic neurons to neuron i; $t_i^{\wedge}$ is the last spike of neuron i; the function η_i takes care of refractoriness after a spike emitted (Eq. (7)); the kernels ε_{ij} model the neurons response to presynaptic spikes (Eq. (8)).

2. Learning in Spiking Neural Networks

Although there already exist several kinds of models of SNNs, the problem of learning with these networks is still a live issue. Both supervised and supervised learning methods are available for spiking neural networks. This section provides a brief insight into a couple of approaches to do learning in SNNs.

2.1. SpikeProp Algorithm

In [1], a supervised learning method for SNNs, based on the well known error-backpropagation is introduced. The general weight correction rule is similar with that of error-backpropagation, namely, weight correction is proportional to the corresponding input (for SNNs: response to presynaptic spikes ε_{ij}) and a special value, local gradient, δ_j:

$$\Delta w_{ij} \equiv \varepsilon_{ij}\, \delta_i \tag{4}$$

Local gradient δ_j is derived using activation state u_i:

$$\delta_i \equiv \frac{\partial E}{\partial t} \frac{\partial t}{\partial u_i}, \tag{5}$$

where error function E is defined as difference between desired and actual spike times respectively, also in a similar way to that of error-backpropagation.

Obtained in this way SNNs solve, in general, problems of the same classes, which multi-layer perceptrons (trained with error-backpropagation) are intended for.

2.2. Hebbian Learning

If for ‘non-spiking’ neural networks supervised learning methods are of a great predominance, then for SNNs the things are different.

Already in 1949, Donald Hebb proposed a precise rule that might govern the synaptic changes underlying learning: “When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency as one of the cells firing B, is increased” [4].

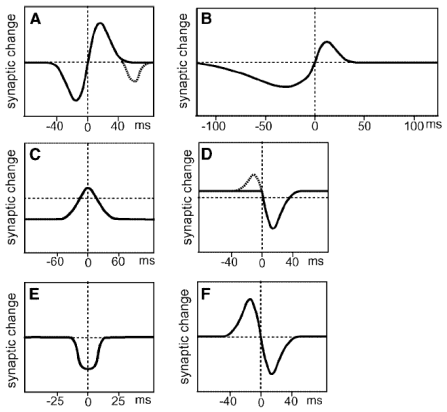

Figure 3. Spike-timing dependent learning rules (originated from several authors), where positive time indicates that the postsynaptic spike follows the presynaptic spike ([7]).

Hebb's formulation of his rule has several important features. Causality is one such feature. According to Hebb, the firing of neuron A must be causally related to the firing of neuron B, which means in practice that spikes in neuron A must precede spikes in neuron B. A simple correlation in which the order of spike times is unimportant would not fit with Hebb's hypothesis. A second feature is the assignment of a critical role to spikes. This is explicit for the postsynaptic cell and is implicit for the presynaptic cell since Hebb's discussion always considers neuronal interaction as mediated by spikes. [7].

Figure 3 shows several known strategies of synaptic weight changes, proposed by different authors.

3. The Learning Method with Adaptive Layer Activation

In this Section, a Hebbian rule based method of learning in SNNs is proposed. The key novelties of the method are the following:

1. Adjustment of average activation state of neurons in each layer in order to ensure a definite average rhythm of spikes within a layer.
2. Maintenance of fixed average volume of synaptic weight in each neuron in order to avoid weights to become 'hollow' (too great or too small).

In [3], several principles of unsupervised learning have been stated. The two above stated ideas ensure two of those principles to be considered:

1. **Cooperation.** Adjustment of activation state is common for all neurons in a layer; thereby cooperation among neurons in a layer is ensured.
2. **Competition.** Sum of weights in a neuron is fixed; thus there is competition among weights in a neuron.

Below in this section, these two ideas are described in detail.

3.1. Some details of construction and operation of neurons

In this subsection, a neural model is described, in which the proposed learning method was incorporated.

A feed-forward (MLP-like) network architecture is used.

Activation state u_i is computed similarly to Eq. (3) by adding a notion of **activation acceleration factor** a^{acc}:

$$\mathrm{u}_i(t) = \eta_i(t - t_i^{\wedge}) + \mathrm{a}^{acc}(t)\frac{\sum_{j\in\Gamma_i}\sum_{t_j^{(f)}\in F_j} w_{ij}\,\varepsilon_{ij}(t - t_j^{(f)})}{n_i}, \tag{6}$$

where n_i – weight count of the neuron i; activation acceleration factor a^{acc} is adjusted during learning; computation of kernels η and ε are specified in Eqs (7), (8) (adapted from [2], [6]):

$$\eta_i(s) = \begin{cases} -\vartheta\exp(-\frac{s}{\tau}); & s > 0 \\ 0; & s \le 0 \end{cases}, \tag{7}$$

where τ is a time constant.

$$\varepsilon_{ij}(s) = \begin{cases} \exp(-\frac{s-\Delta_j^{ax}}{\tau_m}) - \exp(-\frac{s-\Delta_j^{ax}}{\tau_s}); & s-\Delta_j^{ax} > 0 \\ 0; & s-\Delta_j^{ax} \le 0 \end{cases}, \tag{8}$$

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

The activation acceleration factor a^{acc} is introduced for each layer in order to govern the activation state u_i. Since operation of spiking neurons are fully characterized by firings and these occur only if u_i is big enough, the adjusting mechanism is required to ensure it, and the factor a^{acc} is to server for it. As a^{acc} is common for whole layer of neurons, the proposed acceleration makes firing of some neurons of the layer frequent while not guaranteeing the same effect for each neuron apart by itself.

Output of the neuron (in the output layer) is defined as follows:

$$y_i(t) = \begin{cases} 1; & (t-\Delta_i^{ax}) \in F_i \\ 0; & otherwise \end{cases}, \tag{9}$$

where F_i is the spike train, generated by neuron i; Δ_i^{ax} is axonal transmission delay.

3.2. The Learning Algorithm

The proposed algorithm consists of two parts:

- Modification of synaptic weights $\mathbf{w_i}$ (Eq. (10));
- Adjustment of the activation acceleration factor a^{acc} (Eq. (11)).

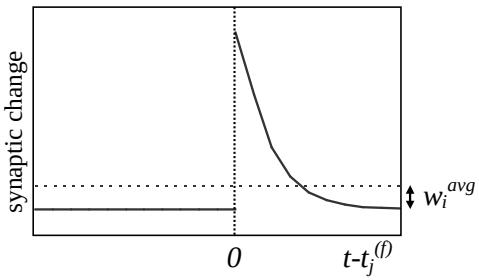

Figure 4. Spike-timing dependent learning rule, realized in Eq. (9)

3.2.1. Weight Modifications

In the proposed algorithm, described by Eq. (10), the synaptic weights of a neuron bear modifications only at moments when the neuron has just fired.

The main principle of weight modification is Hebbian like: the weight value is increased proportionally to how recently spikes were received from the corresponding presynaptic neuron ('recentliness' is represented by the response of incoming neuron, computed using kernel ε_{ij}).

This strategy is complemented by two additional mechanisms to normalize weight values:

- The maximum value of a weight is restricted by the constant c^{wmax};
- Hebb's rule describes the conditions under which synaptic efficacy increases but does not describe the conditions under which it decreases [7]. To determine this, the overall value of the weights of is kept at 0 (by subtracting the average weight value w_i^{avg}).

$$\Delta w_{ij} = \eta^{rate}(c^{wmax} - w_{ij}) \sum_{t_j^{(f)} \in F_j} \varepsilon_{ij}(t - t_j^{(f)}) - w_i^{avg}, \tag{10}$$

where η^{rate} is learning rate; c^{wmax} is a constant representing maximum allowed weight value; and w_i^{avg} – average weight value of the neuron.

Figure 4 proposes graphical interpretation of the proposed weight adjustment strategy (Eq. (10)) in a similar manner to that of Figure 3.

3.2.2. Adjusting the Activation Acceleration Factor

Eq. (11) describes another part of the algorithm, which takes care of adjustment of the activation acceleration factor a^{acc}. The factor a^{acc} is essential to ensure a definite rhythm of firing in a layer.

$$\mathrm{a}^{acc}(t) = \begin{cases} \mathrm{a}^{acc}(t-1) + c^{acc2}\left(\varepsilon^{cum}(t) - c^{\varepsilon}\right)\mathrm{a}^{acc}(t-1); & t > 0 \\ c^{acc1}; & t = 0 \end{cases}, \tag{11}$$

where c^{acc1} and c^{acc2} are constants representing modification style of a^{acc}; c^{ε} is a constant representing recommended value of the neurons' response, thus defining the di-

rection of modification of a^{acc}; ε^{cum} is cumulative response of all neurons in a layer in recent period.

The main idea of adjusting the factor a^{acc} is to make cumulative responses of layers to be close enough to some fixed value c^{ε} (**recommended response**).

$$\varepsilon^{cum}(t) = \begin{cases} c^{cum}\,\varepsilon^{avg}(t) + (1-c^{cum})\,\varepsilon^{cum}(t-1); & t > 0 \\ 0; & t = 0 \end{cases} \tag{12}$$

where ε^{avg} is average response of all neurons in a layer at time t.

$$\varepsilon^{avg}(t) = \frac{\sum_j \varepsilon_i(t)}{n} \tag{13}$$

where $\varepsilon_i(t)$ is response of neuron i in a layer at time t (neuron i itself, not some pre-synaptic neuron).

$$\varepsilon_i(t) = \sum_{t_i^{(f)} \in F_i} \exp(-\frac{t - t_i^{(f)}}{\tau_m}) - \exp(-\frac{t - t_i^{(f)}}{\tau_s}) \tag{14}$$

In terms of computation, Eq. (14) is simplification of Eq. (8).

4. Experimental Work

The proposed learning method was tested on detection of reiterative temporal patterns in an input stream. A trained neural network should be able to detect such patterns by giving output of 1 after pattern is encountered.

4.1. Configuration of Experimental Environment

The testing environment produces 3 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 at times patterns from a predefined store are interwoven into them. There are 6 possible temporal patterns in the store (see Figure 5), and at each test run, two of them are used.

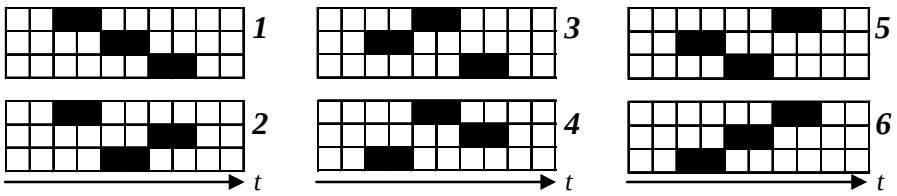

Figure 5. The 6 temporal patterns used in experiments: width=3 signals, length=10 times; black squares denote '1' or 'spike', but white squares – '0' or 'no spike'

Procedure *run_test*
Input
Υ – an initialized neural network
$n^{(inp)}$ – size of input of neural network
pattern_store – set of temporal patterns of width $n^{(inp)}$
$\pi^{(patt)}$, $\pi^{(1)}$ – constants, representing probability
$n^{(patt)}$ – pattern count to be exploited
t – time counter
it – count of iterations to be performed
Output
Sequence of signals generated and passed to neural network
Using
get_random_value – generates a random value from interval 0..1
Begin
Choose at random a subset of patterns $p \subset$ *pattern_store* with $|p| = n^{(patt)}$;
Do *it* times
If ***get_random_value*** $< \pi^{(patt)}$ then
Choose at random a pattern $p_i \in p$;
For $j \leftarrow 1$ to length(p_i) do
Pass p_{ij} to Υ;
run Υ with t
else
For $k \leftarrow 1$ to $n^{(inp)}$ do
If ***get_random_value*** $< \pi^{(1)}$ then *input*[k] = 1
else *input*[k] = 0
Pass *input* to Υ;
run Υ with t
$t \leftarrow t + 1$

Figure 6. Algorithm to generate a temporal sequence of signals to pass to a neural network

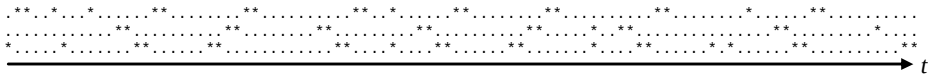

Figure 7. Example of generated signals using the procedure ***run_test*** (Figure 6). The environment was configured according Table 2 (in this example, patterns 5 and 6 of *pattern_store* have been used). Asterisks (*) represent input values of 1, while dots (.) represent 0. An excerpt of length of 100 is displayed

Figure 6 describes the algorithm of operation of the testing environment, while Figure 7 displays an excerpt of a generated sequence using this algorithm.

Neural networks used in the experiments operated as described above (Eqs. (2), (6), (9), (10), (11)). Table 1 describes architecture of neural networks, as well as various constants that affect operation.

Table 1. Parameters of configuration of neural network.

Parameter	**Used in Eqs.**	**Value or description**
Input count $n^{(inp)}$		3
Output count $n^{(outp)}$		1
Layer topology τ^{top}		feed-forward ; one of {3, 5, 1}, and {3, 5, 3, 1}
Constants τ_m, τ_s, τ	(7), (8), (14)	1.5, 0.5, 2.0
Threshold ϑ	(2), (7)	1.0

Parameter	Used in Eqs.	Value or description
Learning rate η^{rate}	(10)	0.05
Axonal delay Δ_i^{ax}	(8), (9)	chosen at random from axonal delay set Δ^{ax}
Axonal delay set Δ^{ax}		one of {0}, {0, 1}, {0, 1, 2}, and {0, 1, 2, 3}
Constants c^{acc1}, c^{acc2}, c^{cum}	(11), (12)	10.0, 0.02, 0.1
Recommended response c^{ε}	(11)	one of 0.01, 0.02, 0.03, 0.05
Initial weight values w_{ij}		chosen at random from interval -0.3..+0.3
Maximum weight value c^{wmax}	(10)	1.0

4.2. Performing the Experiment

32620 test runs were made during experimentation, according the algorithm shown in Figure 8, and configuration parameters depicted in Table 1 and Table 2.

Table 2. Parameters of experiments.

Parameter	Value or description
pattern_store	6 patterns, each pattern of size 10×3 (see Figure 5)
Probabilities $\pi^{(patt)}$, $\pi^{(1)}$	0.1, 0.2
Pattern count in one test run $n^{(patt)}$	2
count of iteration for the learning phase it^{learn}	5000
count of iteration for the recall phase it^{recall}	600

Procedure *do_experiment*
Input
ϒ – a neural network
Output
Result sets R^{good} and R^{bad}
Using
run_test – generates input, passes it to a neural network, and runs the network
Begin
Do many times
Initialize neural network ϒ according Table 1;
Initialize environment according Table 2;
/ r represents results of one test run */*
$r \leftarrow \varnothing$;
Call ***run_test*** with t = 0, it = it^{learn} in learning mode and record test parameters to r;
$a^{acc\_max} \leftarrow$ maximum activation acceleration factor a^{acc} among layers after learning
$a^{acc\_min} \leftarrow$ minimum activation acceleration factor a^{acc} among layers after learning
If $a^{acc\_max} / a^{acc\_min} > 5.0$ then
Add r to bad results R^{bad}
else
/ recall mode differs from learning mode with the fact that no learning is performed */*
Call ***run_test*** with t = it^{learn}, it = it^{recall} in recall mode, record input, output and test parameters to r;
Add r to good results R^{good}

Figure 8. Algorithm to obtain experimental results

5. Experimental Results and Conclusion

5.1. Collection and Assessment of the Results

Experimental activities were performed according the algorithm of Figure 8, and then the obtained result set R^{good} was further analyzed.

The main idea of analyzing the result data was to account the output spikes, which follow the patterns presented to input. Thus we recognize, whether a neural network has learned regularities in the input.

Figure 9 renders a brief insight into performance of neural network after the learning phase.

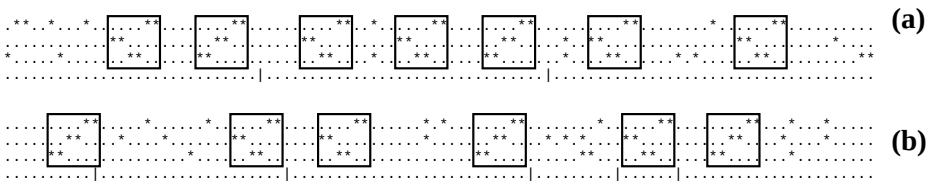

Figure 9. Example of obtained test results. Asterisks (*) represent input values of 1, dots (.) represent input and output values of 0, vertical bars (|) represent output spikes (i.e., values of 1). Actual patterns, presented to input, are enframed. Two excerpts of two different tests of length of 100 are displayed. Part (a) shows that the network has learned to recognize only 1 of 2 pattern types. Part (b) shows that the network has learned to recognize both types, but with mistakes: one spike is wrong, and two of patterns have not been recognized. To better keep track of the example, consult Figure 5

Experimental results represent the percentage of 'good' output spikes among all spikes.

For each pattern type in a test the 'good' output spikes were accounted with respect to 6 different positions and then taken the maximum of these 6. These positions were *i-3*, *i-2*, *i-1*, *i*, *i+1*, *i+2*, where *i* is the end of the pattern (the one after the last). For example, in Figure 9a both spikes are at the position -1 with respect to the actual pattern (consult Figure 5 to see that each pattern starts and ends with double 0).

For a fixed pattern *p* and position pos_i^p, a spike *s* has been accounted if the spike s was pos_i^p-1, pos_i^p, or pos_i^p+1. Thus, for example, in Figure 9a both spikes were accounted for positions -2, -1, and 0.

In that manner, a triple of numbers $<r_1, r_2, r_{total}>$ from each test was obtained: spike count for each of two patterns and the total spike count. The two cases of Figure 9 yields the following triples: <0, 2, 2> and <2, 2, 5>.

Quality of a neural network of a test was measured in two ways: simple $q^{(s)}$ and advanced $q^{(a)}$.

Simple way means the proportion of 'good' and all spikes:

$$q^{(s)} = \frac{r_1 + r_2}{r_{total}} \tag{15}$$

where r_1, r_2 – 'good' spike count for patterns 1 and 2 respectively, r_{total} – total spike count.

For the two cases of Figure 9, this is (0+2)÷2=100% and (2+2)÷5=80%. The quality of the first case is better, because both spikes are 'good'.

Advanced way means taking into account situation, whether **both** pattern types have been recognized correctly:

$$q^{(a)} = \frac{4r_1 r_2}{r_{total}^2} \tag{16}$$

For the two cases of Figure 9, this is $(4 \cdot 0 \cdot 2) \div 2^2 = 0\%$ and $(4 \cdot 2 \cdot 2) \div 5^2 = 64\%$. The quality of the first case is 0, because the network was able to recognize only one of two types of patterns.

5.2. Experimental Results and Analysis

21117 of 32620 (or ~65%) of results were 'good', i.e., were included in the result set R^{good}, which was further analyzed. A bad result means that the network has not converged during the learning.

Table 3 describes summarized test results according both quality measures $q^{(s)}$ and $q^{(a)}$. Results are grouped according selection of some parameters or parameter sets. Consult Table 1 for parameter types.

Table 3. Experimental results overview.

Parameter or parameter set	**q^s (%)**	**q^a (%)**
Layer topology $\tau^{top} = \{3, 5, 3, 1\}$	70.1	33.2
Layer topology $\tau^{top} = \{3, 5, 1\}$	60.3	24.4
Axonal delay set $\Delta^{ax} = \{0\}$	56.5	22.3
Axonal delay set $\Delta^{ax} = \{0, 1\}$	65.6	28.4
Axonal delay set $\Delta^{ax} = \{0, 1, 2\}$	67.7	21.2
Axonal delay set $\Delta^{ax} = \{0, 1, 2, 3\}$	67.4	30.1
Recommended response $c^{\varepsilon} = 0.01$	49.1	16.6
Recommended response $c^{\varepsilon} = 0.02$	55.7	16.5
Recommended response $c^{\varepsilon} = 0.03$	62.4	20.2
Recommended response $c^{\varepsilon} = 0.05$	65.9	34.6
Layer topology $\tau^{top} = \{3, 5, 3, 1\}$; Axonal delay set $\Delta^{ax} = \{0\}$; Recommended response $c^{\varepsilon} = 0.05$	73.8	42.5

- Experimental results show good quality on measure $q^{(s)}$. This means that obtained networks are good on detecting at least one of the hidden patterns types.
- Experimental results according the measure $q^{(a)}$ is worse. This means that obtained networks have mostly learned only one of two pattern types (like in Figure 1a).

– Significantly better results were obtained with recommended response $c^{\varepsilon} = 0.05$, especially according the quality measure $q^{(a)}$.
– Networks with two hidden layers were better than those of only one.
– Effect of different axonal transmission delays depends on the quality measure and is mostly uncertain.

5.3. Conclusion

The problem of learning in spiking neural networks is still a challenging issue. This paper introduces an unsupervised learning method that exploits cooperation of neurons using a mechanism of adaptive layer activation. This method is suitable to train neural networks for regularity detection in a stream of data.

Although the environment, built for experiments, was fairly simple, still the first results are to be considered very promising and inspiriting for further research.

Acknowledgements

This research has been supported by ESF (Project #LU ESS2004/3).

References

[1] S. M. Bohte, J. N. Kok, H. La Poutré. Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48, pp. 17-37 (2002)
[2] W. Gerstner. Spiking Neurons. In W. Maass and C. M. Bishop (Eds.), Pulsed neural networks. MIT Press, 1999.
[3] S. Haykin. Neural networks: a comprehensive foundation. 2nd ed. Prentice-Hall, Inc., 1999
[4] D. Hebb. The organization of behavior. John Wiley and Sons, New York (1949)
[5] R. Kempter, W. Gerstner, J., and L. van Hemmen. Hebbian learning and spiking neurons. Physical Review E, 59:4498-4514, 1999.
[6] W. Maas. Computing with Spiking Neurons. In W. Maass and C. M. Bishop (Eds.), Pulsed neural networks. MIT Press, 1999.
[7] P. D. Roberts, C. C. Bell. Spike timing dependent synaptic plasticity in biological systems. Biological Cybernetics, 87, pp. 392-403 (2002)