# Learning With Adaptive Layer Activation in Spiking Neural Networks

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

**Abstract.** A spiking neural network (SNN) is a neural model in which the communication between two neurons is performed by a "pulse code", i.e., the process of computation considers the timings of the signals (spikes) while ignoring their strength. It has been shown theoretically that a neural network which is composed of spiking neurons is able to learn a large class of target functions. In this paper, an unsupervised learning algorithm is introduced for a spiking neural network. The algorithm is based on the Hebbian rule, with the addition of the principle of adapting a layer activation level so as to guarantee frequent firings of neurons for each layer. The proposed algorithm has been illustrated with experimental results related to the 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*

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 [2]. 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 such models is usually arranged in steps – during each step exactly one pattern is classified or recognised, and no temporal information is processed directly by the neural network.

In contrast to classical models, spiking neural networks (SNNs) 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 the focus of much recent research, inspired by the opportunity to exploit another re-

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

semblance to the brain and by the sense of a challenge related to an understanding of the brain's functions.

Each neuron in an SNN produces a train of spikes (i.e., it fires), and the operation of the neuron $i$ is fully characterised by the set of firing times [3] (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 $i$.

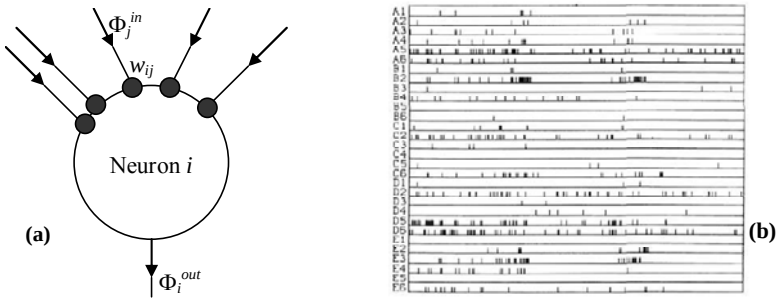


**Figure 1.** Spiking neurons and trains of spikes. (a) A single spiking neuron. The small filled circles $w_{ij}$ denote synaptic weights. $\Phi_j^{in}$ denotes an incoming spike train $j$. The neuron produces the spike train $\Phi_i^{out}$ (adapted from [6]). (b) A four-second recording of neural activity from 30 neurons from the visual cortex of a monkey. Each vertical bar indicates a spike [7]

### *1.2. Operation of a Spiking Neuron*

One of the most frequently applied spiking neuron models is the integrate-and-fire model. 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.

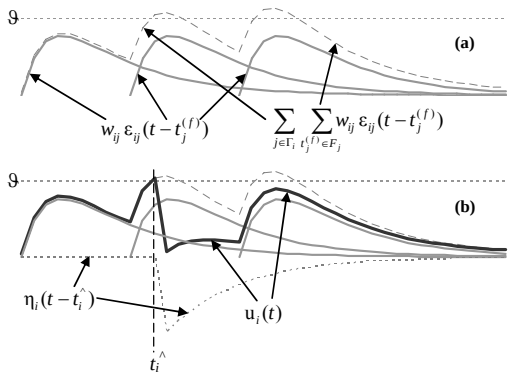


**Figure 2.** Graphic interpretation of activation state $u_i$ (Eq. (3))

Neuron $i$ is said to fire if its activation state $u_i$ reaches the threshold $\vartheta$. At the moment when the threshold is crossed, a spike with the firing time $t_i^{(f)}$ is generated (see Eq. 2)).

Thus Eq. (1) can be clarified with Eq. (2) [3]:

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

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

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

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

## 2. Learning in Spiking Neural Networks

Although there are several different kinds of models of SNNs, there is still the aspect of learning in relation to these. Both supervised and unsupervised learning methods are available for spiking neural networks. This section provides a brief insight into a couple of ways achieving learning in SNNs.

### *2.1. SpikeProp Algorithm*

In [1], a supervised learning method for SNNs is introduced. It is based on the well known concept of error-backpropagation. The general weight correction rule is similar to that which is applied to error-backpropagation, i.e., it is proportional to the corresponding input (for SNNs: response to presynaptic spikes $\varepsilon_{ij}$), and a special value – the local gradient $\delta_j$:

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

The local gradient $\delta_j$ 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 the difference between the desired and the actual spike times respectively, also in a similar way to that of error-backpropagation.

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

### *2.2. Hebbian Learning*

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

Back in 1949, Donald Hebb proposed a precise rule which might govern the synaptic changes which underlie 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" [5].

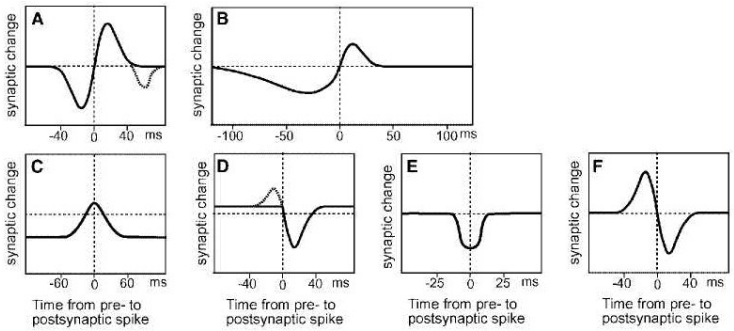


**Figure 3.** Spike-timing-dependent Hebbian-like learning rules, where positive time indicates that the postsynaptic spike follows the presynaptic spike (proposed by various authors). (A) Antisymmetric Hebbian learning rule; (B) Antisymmetric Hebbian learning rule; (C) Symmetric Hebbian learning rule; (D) Anti-Hebbian learning rule; (E) Symmetric anti-Hebbian learning rule; (F) Theoretical antisymmetric anti-Hebbian learning rule (adapted from [9])

Hebb's formulation of his rule has several important features. One is causality. According to Hebb, the firing of neuron *A* must be causally related to the firing of neuron *B*, which in practice means 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 satisfy Hebb's hypothesis. A second feature is the assignment of a critical role to spikes. This is explicit for the postsynaptic cell and implicit for the presynaptic cell because Hebb's discussion always considers neuronal interaction as mediated by spikes. [9]

Figure 3 shows several known strategies of synaptic weight changes, as proposed by various authors. The strategies manifest the direction and range of weight modifications with respect to differences between pre- and postsynaptic timings.

## 3. The Learning Method With Adaptive Layer Activation

In this section, a Hebbian rule-based method for learning in SNNs is proposed. The key innovations in this method are the following:

1) Adjustment of the average activation state of neurons in each layer so as to ensure a definite average rhythm of spikes within a layer;
2) Maintenance of the fixed average volume of synaptic weight in each neuron so as to avoid a situation in which weights become too "hollow" (too great in terms of absolute value).

Several principles of unsupervised learning have been stated in [4]. The two aforementioned ideas ensure that two of those principles are considered:

1) **Co-operation:** Adjustments of the activation state are common for all neurons in a layer which ensures co-operation among neurons in that layer.

2) **Competition:** The sum of weights in a neuron is fixed, which means that there is competition among weights in a neuron.

These two ideas are described in detail in this section.

### *3.1. Details in the construction and operation of neurons*

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

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

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

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

where $n_i$ is the weight count of the neuron $i$; the activation acceleration factor $a^{acc}$ is adjusted during learning; computation of the kernels η and ε is specified in Eqs (7), (8) (adapted from [3], [8]):

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

where $\tau$ is a time constant.

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

where $\tau_s$ and $\tau_m$ are time constants, and $\Delta_j^{ax}$ is the 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 $u_i$. Because the operation of spiking neurons is fully characterised by firings and these occur only if $u_i$ is big enough, an adjusting mechanism is required to ensure it, and that is what the factor $a^{acc}$ does. Because $a^{acc}$ is in common for the entire layer of neurons, the proposed acceleration means that the firing of some neurons of the layer will be frequent while not guaranteeing the same effect for each neuron individually.

The 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 $i$; $\Delta_i^{ax}$ 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)).

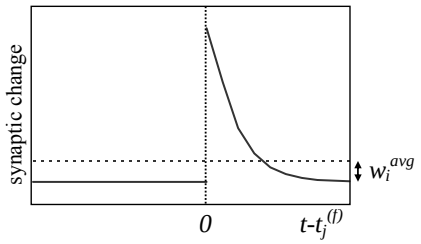


**Figure 4.** The spike-timing dependent learning rule, as presented in Eq. (9)

#### 3.2.1. Weight Modifications

In the proposed algorithm that is presented in Eq. (10), the synaptic weights of a neuron bear modifications only when the neuron has just fired.

The main principle of weight modification recalls the Hebbian approach – the weight value is increased proportional to how recently spikes were received from the corresponding presynaptic neuron ('recentliness' is represented by the response of the incoming neuron, and this is computed using kernel $\varepsilon_{ij}$).

This strategy is complemented by two additional mechanisms to normalise 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 [9]. 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 $\eta^{rate}$ is the learning rate; $c^{wmax}$ is a constant representing maximum allowed weight value; and $w_i^{avg}$ is the average weight value of the neuron.

Figure 4 proposes graphical interpretation of the proposed weight adjustment strategy – Eq. (10) It is similar in manner to that which is shown in Figure 3.

#### 3.2.2. Adjusting the Activation Acceleration Factor

Eq. (11) describes another part of the algorithm, which speaks to the 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 the modification style of $a^{acc}$; $c^{\varepsilon}$ is a constant which represents the recommended value of the neurons' response, thus defining the direction of modification of $a^{acc}$; $\varepsilon^{cum}$ is the cumulative response of all neurons in a layer in the recent period.

The main idea in adjusting the factor $a^{acc}$ is ensuring that the cumulative responses of layers are close enough to some fixed value $c^{\varepsilon}$ (**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 $\varepsilon^{avg}$ is the 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 the response of neuron $i$ in a layer at time $t$ (neuron $i$ itself, not some presynaptic neuron).

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

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

## 4. Experimental Work

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 an output of 1 after the pattern is encountered.

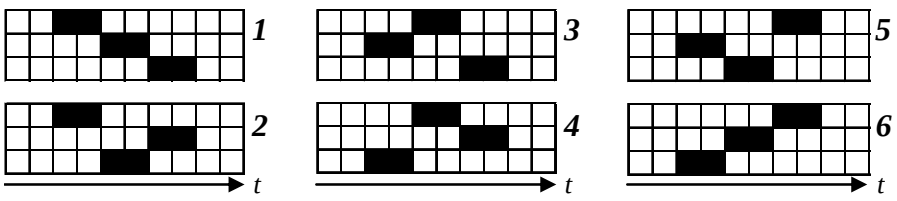


**Figure 5.** The 6 temporal patterns used in experiments: width=3 signals, length=10 times; black squares denote '1' or 'spike', but white squares – '0' or 'no spike'

### *4.1. Configuration of the Experimental Environment*

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 5), and two are used in each test run.

**Procedure *run_test***
**Input**
$\Upsilon$ – an initialised neural network
$n^{(inp)}$ – size of input of the 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 $\Upsilon$;
run $\Upsilon$ 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 $\Upsilon$;
run $\Upsilon$ with $t$
$t \leftarrow t + 1$

**Figure 6.** The algorithm to generate a temporal sequence of signals to pass to a neural network

Figure 6 describes the algorithm of the testing environments operation, while Figure 7 displays an excerpt from a generated sequence which uses this algorithm.

The neural networks, which were used in the experiments, operated as described above in Eqs. (2), (6), (9), (10) and (11).

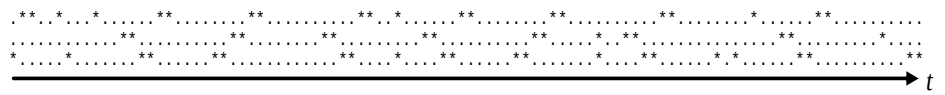


**Figure 7.** An example of generated signals using the procedure ***run_test*** (Figure 6). The environment was configured according to Table 2 (in this example, patterns 5 and 6 of *pattern_store* have been used). Asterisks (*) represent input values of 1, while dots (.) represent 0. An excerpt of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

Table 1 describes the architecture of neural networks, as well as the various constants which affect the operations so as to illuminate the experimental environment. The pa-

rameters of Table 1 and Table 2 were mostly obtained via optimization in a broader experimental project which goes beyond the scope of this paper.

**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 $\tau^{top}$ | | feed-forward; one of {3, 5, 1}, and {3, 5, 3, 1} |
| Constants $\tau_m$, $\tau_s$, $\tau$ | (7), (8), (14) | 1.5, 0.5, 2.0 |
| Threshold $\vartheta$ | (2), (7) | 1.0 |
| Learning rate $\eta^{rate}$ | (10) | 0.05 |
| Axonal delay $\Delta_i^{ax}$ | (8), (9) | chosen at random from axonal delay set $\Delta^{ax}$ |
| Axonal delay set $\Delta^{ax}$ | | 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 $c^{\varepsilon}$ | (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*

32,620 test runs were conducted during experiment in according the algorithm that is shown in Figure 8 and with the configuration parameters that are depicted in Tables 1 and 2.

**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;
$r \leftarrow \varnothing$; */* r represents results of one test run */*
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.** The algorithm to obtain experimental results

**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 iterations for the learning phase $it^{learn}$ | 5000 |
| count of iterations for the recall phase $it^{recall}$ | 600 |

## 5. Results of the Experiment and Conclusions

### *5.1. The Collection and Assessment of the Results*

Experiments were performed in accordance with the algorithm that is shown in Figure 8. Then the obtained result set $R^{good}$ was analyzed further.

The main idea in analysing the result data was to account for output spikes which follow the patterns that are presented to the input. Thus we see whether a neural network has learned the regularities of the input.

Figure 9 provides a look at the performance of a neural network after the learning phase.

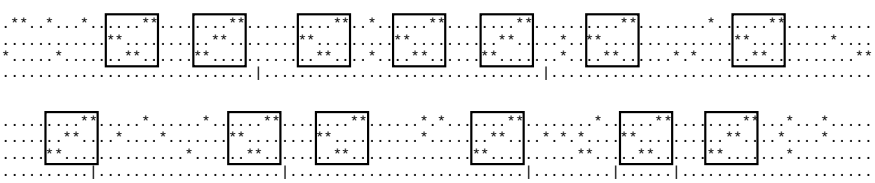


**Figure 9.** An example of obtained test results. Asterisks (*) represent input values of 1, dots (.) represent input and output values of 0, vertical bars (|) represent output spikes (i.e., values of 1). Actual patterns presented to input are in the frames. Two excerpts of two different tests of length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

The 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 for with respect to six different positions, and then the maximum of the six was taken. These positions were *i*-3, *i*-2, *i*-1, *i*, *i*+1, and *i*+2, where *i* is the end of the pattern (the one after the last). In Figure 9a, for instance, both spikes are at the position of -1 with respect to the actual pattern (consult Figure 5 to see that each pattern starts and ends with a double 0).

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

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

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

The 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, $r_{total}$ is the 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".

The advanced way means taking into account the situation, asking whether **both** pattern types have been recognised 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*

Of the 32,620 tests, 21,117 (approximately ~65%) had "good" results, i.e., they were included in the result set $R^{good}$, which was further analysed. A bad result means that the network has not converged during the learning.

**Table 3.** Experimental results overview.

| **Parameter** | **$q^s$ (%)** | **$q^a$ (%)** |
|---|---|---|
| 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 |

Table 3 describes summarised test results in accordance with the two quality measures – $q^{(s)}$ and $q^{(a)}$. The results are grouped in accordance with certain parameters (see Table 1 for the types of parameters). The results are to demonstrate the applicability of

the proposed learning algorithm (represented here by the parameter "recommended response"). Other parameters which are found in the results overview are the subject for future work. They are noted here simply to offer a more in-depth insight into the experimentation.

- The results of the experiments show good quality on measure $q^{(s)}$. This means that the obtained networks are good at detecting at least one of the hidden patterns types.
- The results in terms of the measure $q^{(a)}$ are worse. This means that the obtained networks mostly learned only one of the two pattern types (as in Figure 1a).
- Significantly better results were obtained with the recommended response $c^{\varepsilon} = 0.05$, especially in accordance with the quality measure $q^{(a)}$.
- Networks with two hidden layers were better than those with only one.
- The effect of different axonal transmission delays depends on the quality measure and it is mostly uncertain.

### *5.3. Conclusion*

Spiking neural networks are designed to make use of the temporal dynamics of neurons. SNNs are believed to be very powerful, and their principles of operation are more complicated than those of "ordinary" neural networks.

The problem of learning in spiking neural networks is still a challenging issue. This paper presents an unsupervised learning method which exploits the co-operation of neurons making use of a mechanism of adaptive layer activation. This method is suitable to train neural networks to detect regularity in a stream of data.

Although the environment of the experiments was fairly simple, the first results are to be considered as very promising and inspirational for further research.

## Acknowledgements

This research was supported by the ESF
(Contract No. 2004/0001/VPD1/ESF/PIAA/04/NP/3.2.3.1/0001/0001/0063).

## 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] L. Fausett. Fundamentals of Neural Networks. Prentice-Hall, Inc., 1993
[3] W. Gerstner. Spiking Neurons. In W. Maass and C. M. Bishop (Eds.), Pulsed neural networks. MIT Press, 1999.
[4] S. Haykin. Neural networks: a comprehensive foundation. 2nd ed. Prentice-Hall, Inc., 1999.
[5] D. Hebb. The organization of behavior. John Wiley and Sons, New York, 1949.
[6] R. Kempter, W. Gerstner, J., and L. van Hemmen. Hebbian learning and spiking neurons. Physical Review E, 59:4498-4514, 1999.
[7] J. Kruger, F. Aiple. Multimicroelectrode investigation of monkey striate cortex: spike train correlations in the infra-granular layers, Journal of Neurophysiology, vol. 60, pp. 798-828, 1988.
[8] W. Maas. Computing with Spiking Neurons. In W. Maass and C. M. Bishop (Eds.), Pulsed neural networks. MIT Press, 1999.
[9] P. D. Roberts, C. C. Bell. Spike timing dependent synaptic plasticity in biological systems. Biological Cybernetics, 87, pp. 392-403, 2002.