Available online at www.sciencedirect.com

SCIENCE DIRECT®

Neurocomputing 64 (2005) 335–357

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Unsupervised learning of synaptic delays based on learning automata in an RBF-like network of spiking neurons for data clustering

P. Adibi[a], M.R. Meybodi[b,*], R. Safabakhsh[a]

[a]*Computational Vision/Intelligence Laboratory, Computer Engineering Department, Amirkabir University of Technology, Hafez Avenue, Tehran, Iran*

[b]*Soft Computing Laboratory, Computer Engineering Department, Amirkabir University of Technology, Hafez Avenue, Tehran, Iran*

Received 28 January 2004
Communicated by T. Heskes
Available online 8 January 2005

Abstract

In this paper, a new delay shift approach for learning in an RBF-like neural network structure of spiking neurons is introduced. The synaptic connections between the input and the RBF neurons are single delayed connections and the delays are adapted during an unsupervised learning process. Each synaptic connection in this network is modeled by a learning automaton. The action of the automaton associated with each connection is considered as the delay of the corresponding synaptic connection. It is shown through simulations that the clustering precision of the proposed network is considerably higher than that of the existing similar neural networks.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Spiking neural networks; Delay learning; Learning automata; Data clustering

*Corresponding author. Tel.: +98 21 26 40226.
E-mail address: meybodi@ce.aut.ac.ir (M.R. Meybodi).

0925-2312/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2004.10.111

1. Introduction

In the theory of artificial neural networks (ANNs), various models have been proposed for the processing elements or "neurons", which are classified in three generations by Maass [10]. The "McCulloch and Pitts" model is considered as the first generation of artificial neurons. In this model, the output of a neuron can only have one of two possible values, which usually are 0 and 1. In sigmoidal neurons, which are considered to be the second generation of neurons, the output of a neuron is a continuous value, usually between 0 and 1, which is generated by a sigmoid function of a weighted sum of input variables. From the biological point of view, this continuous output indicates the firing rate of the neuron at each time.

Biological observations have shown that in some cases, the firing rate cannot be the only signal transmitting information from one neuron to another. It seems that the time delay of an input spike may contain information. For example, it has been shown that the time required for pattern analysis and classification in human beings is about 100 ms, and the image constructed on retina passes through at least 10 synaptic stages sequentially in the visual pathways [17,21]. Because of the low firing rate of cortical neurons (about 100 Hz), there is no sufficient time for sampling the inputs to find their frequency. Moreover, von der Malsburg has shown in [22] that considering temporal coding for neurons can easily overcome some problems of static coding (which is based on firing rate) such as the superposition catastrophe. Following these observations, the third generation of neuron models, i.e. spiking neurons, which are more realistic models of learning in biological neurons, have been proposed. In these models, action potentials are received by a neuron with different delays and weights and are integrated together to generate the membrane potential of the neuron. When this membrane potential exceeds a threshold, the neuron fires and produces a spike in its output. In these models, the time at which an output spike is produced, can indicate the output information of the neuron. Albeit, in some kinds of these models, both the time and the rate of spike generation may contain information [6].

Various models are reported for spiking neurons by Gerstner [6]. Two types of these models are the *Leaky Integrate and Fire* (LIF) model and the *Spike Response Model* (SRM). Recently, the learning problem in a network of spiking neurons has attracted a number of researchers. One reason for this interest is that learning in these networks can be considered as a more realistic model of learning in the biological neural networks. Moreover, in some applications, the spiking networks have much less computational complexity than the sigmoidal nets [10,11].

Some researchers have developed modified versions of more conventional learning algorithms of sigmoidal neural networks for learning in spiking neural networks (SNN's). For example, error backpropagation method for SNN [2], Kohonen self-organizing layer [19] and other self-organizing networks [3] with spiking neurons, spiking Hopfield networks [12], associative memories with spiking neurons [20], the RBF networks of spiking neurons [15,8,1], and spiking Boltzman machines [7] have been presented.

In the networks of spiking neurons, learning of synaptic delays can be considered as an addition to, or as a replacement of, the learning of synaptic weights. There are two approaches for computational implementation of delay learning [4]. The first one is called *delay selection* and the second one is called *delay shift*. In the delay selection method, the output of a neuron is assumed to be connected to the input of another neuron by multiple connections (which are called synaptic terminals), each with a different fixed delay, such that these delays cover a specific range of time. The strengths of these connections are updated by learning algorithms. Then, the weights of connections with more suitable delays will be increased and the weight of connections with unsuitable delays will be decreased. A learning algorithm based on this method is reported in [15]. In delay shift, it is supposed that only one delayed synaptic connection exists between two neurons. A model of learning based on delay shift is presented in [5].

One of the important applications of spiking networks is data clustering. An RBF-like network of spiking neurons used for this purpose, was first proposed by Hopfield [8]. The main idea in [8] is that a spiking neuron, as an RBF neuron, has a center which is encoded by the vector of the input synaptic delays. If an input vector, which is temporally encoded, is near to the center of this neuron, the neuron fires; and otherwise, it does not fire. Moreover, the nearer to a reference time the RBF neuron fires, the nearer is the input vector to the RBF center. Based on this idea, the vector of synaptic delays for each RBF neuron can be considered as the prototype of a cluster, and learning these delays is equivalent to clustering the input space. Many findings in neurobiology show the correctness of this idea [16]. Ruf et al. have proposed a learning method based on Hebb's rule for delay selection in synaptic terminals of such an RBF network [15,18]. Bohte et al. [1] have indicated two problems for this network. The first problem is unsuitable clustering for problems for which the number of clusters is greater than the input dimensionality. The second problem is the low clustering precision of this network. They have proposed a population coding to overcome the first problem [1]. Actually, by this coding, the input dimensionality is artificially increased and then clustering is performed in the resulting problem with high dimensional inputs, by the method of [15]. To overcome the second problem, a variable number of receptive fields in population coding is considered and another layer of neurons is added to the network [1]. These modifications lead to a network with more complex structure and also higher number of neurons.

In this paper, a new version of the network reported in [15] is proposed. The proposed network is different from the network reported in [15] in both structure and learning method, and it overcomes the two previously mentioned problems rather simply. The proposed network not only avoids increasing the number of neurons, but also makes the structure of the network simpler than the one reported in [15]. The structural differences of the proposed network consist of: (1) replacement of multiple connections (synaptic terminals) between input and RBF neurons in [15] by single connections in the new network, and (2) elimination of lateral and self-inhibitory connections of [15] in the proposed network (but there is a mechanism of finding winner neurons in our network, as will be seen, which can be considered as a replacement of lateral inhibitory connections). In the proposed network, rather than a Hebb rule, learning automaton is used as a model for learning [14]. Using learning automaton, we model synapses in a probabilistic manner by assigning probabilities to synaptic delays. Every time an input is presented to the network, the learning automata update these probabilities. This process continues until the probabilities of optimal delays approach 1 and the probabilities of other delays approach 0. We also have implemented the network proposed in [18] and compared it with the proposed method for high dimensional problems. It is shown that the clustering precision of the proposed network is considerably higher than that of the network reported in [18].

The rest of this paper is organized as follows. Section 2 describes the structure and the operation method of our network. A brief introduction to learning automata is given in Section 3. Section 4 describes the process of learning in the proposed network. In Section 5, the performance of the network is shown through experimental results. The last section gives the conclusions.

2. Structure and operation of the proposed network

The proposed network has a two-layer feed-forward structure, as shown in Fig. 1(a). The first layer (input layer) contains m neurons and the second layer (RBF layer) contains n neurons. There is one single synaptic connection between each input neuron and each RBF neuron. The weight of this connection is constant and equal to 1 and does not change during the learning process. Here, synaptic delays are the subject of learning. RBF neurons are Leaky Integrate and Fire (LIF)-type spiking neurons whose firing thresholds are considered to be constant. The input vector indicates the firing times of the input neurons and each element of the vector is in the range $[0, T_{\text{ref}}]$. When an input vector is presented to the network, the input neurons fire at the time determined by the vector and generate a spike at their outputs. For applications with non-temporal data, as in the experiments on real-world problems reported in Section 5.2, the input data is only interpreted as the firing time of the input neurons. The spikes, passing through synapses, become wide and delayed which are called Post Synaptic Potentials (PSPs) in this time (Fig. 1(b)).

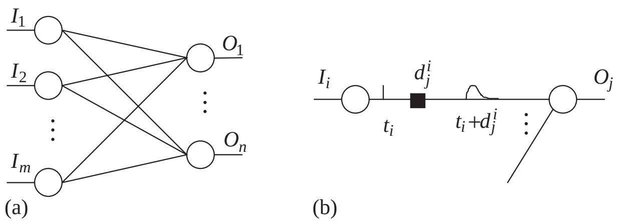

Fig. 1. (a) Structure of the network, (b) a synapse and its effect on the signal. I_i is the ith dimension input, O_j is the jth dimension output, t_i is the firing time of ith input neuron, and d_j^i is the delay of synaptic connection between input neuron i and output neuron j.

In the LIF model the equation of a PSP that starts from time zero is defined as

$$\varepsilon(t) = \frac{t}{\tau}\,\mathrm{e}^{(1-t/\tau)}, \tag{1}$$

where τ defines the width of PSP (Fig. 2). The membrane potential of an RBF neuron is obtained from summation of its input PSPs and the neuron fires when the membrane potential exceeds firing threshold value (Fig. 3). After firing, a neuron cannot fire any more for a period of time called *refractory period*.

The output of the jth RBF neuron in the LIF model, which is the neuron's firing time here, is obtained by solving the following equation for t, given the temporal coded input vector $(t_1, \ldots, t_n)^{\mathrm{T}}$:

$$\begin{aligned} P_{m_j}(t) &= \sum_{i=1}^{m} PSP(t, t_i, d_j^i) = \sum_{i=1}^{m} \varepsilon(t - t_i - d_j^i) \\ &= \sum_{i=1}^{m} \frac{(t - t_i - d_j^i)}{\tau}\,\mathrm{e}^{(1-(t-t_i-d_j^i)/\tau)} = Thr, \end{aligned}$$

where $(d_j^1, \ldots, d_j^m)$ is the synaptic delay vector of the RBF neuron number j.

The firing threshold is considered sufficiently large and therefore the neuron can fire only when its input PSPs are considerably near to each other in the temporal domain. The center of an RBF neuron is defined based on delays of its input synaptic connections (d_i's) as follows:

$$\mathbf{C} = (c_1, \ldots, c_m)^{\mathrm{T}} = (T_{\mathrm{ref}} - d_1, \ldots, T_{\mathrm{ref}} - d_m)^{\mathrm{T}}, \quad d_i \in [0, T_{\mathrm{ref}}]. \tag{2}$$

The network learns in an unsupervised manner. That is, when an input vector is presented to the network, the centers of RBF neurons are updated such that they finally reside near the center of the existing clusters in the input space and this way a clustering operation is performed.

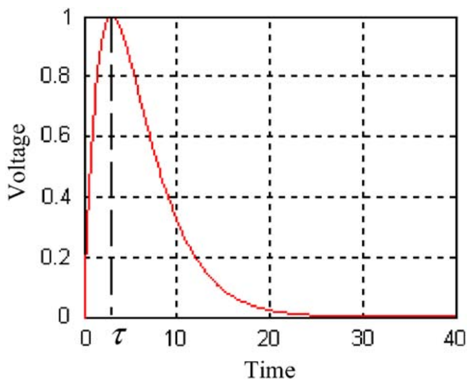

Fig. 2. Graph of a postsynaptic potential (PSP).

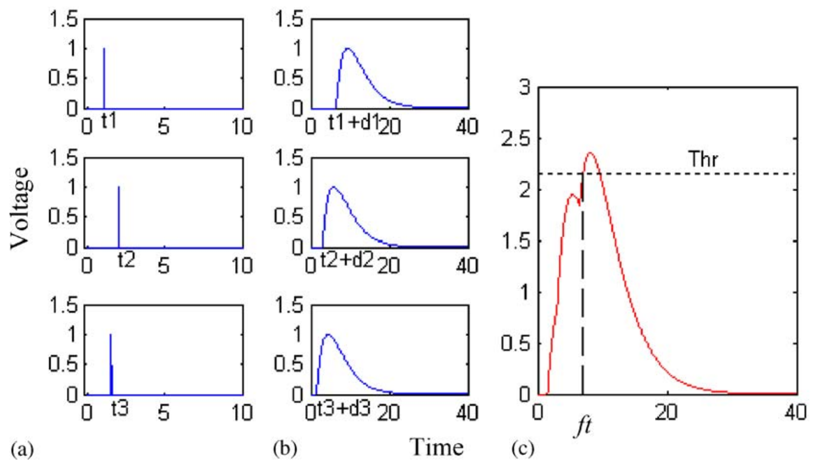

Fig. 3. Plots for (a) presynaptic potentials, (b) postsynaptic potentials, and (c) membrane potential of an RBF neuron with three inputs. [t1,t2,t3] is the input pattern vector which deals with firing times of input neurons, [d1,d2,d3] is the synaptic delay vector, *Thr* is the firing threshold, and *ft* is the firing time for the RBF neuron.

3. Learning automata

Learning in an automaton can be expressed as determining an optimal action out of a set of allowable actions. These actions are presented to an unknown stochastic environment. The environment produces an output signal in response to the input action which acts as a feedback signal in the automaton's learning. The relation between an automaton and its environment is shown in Fig. 4. An automaton that operates in a random environment and during the operation enhances its performance is called a *learning automaton* [14]. The environment is defined by a triple $\{\boldsymbol{\alpha}, C, \boldsymbol{\beta}\}$, where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_r\}$ is the set of inputs to the environment (or actions of the automaton), $\boldsymbol{\beta}$ is the set of possible values for the output of the environment (the output indicates to what degree the current action is suitable), and $C = \{c_1, c_2, \ldots, c_r\}$ is the set of punishment probabilities for each possible input that always are unknown.

Learning automata (LA) can be classified into two main families, fixed and variable structure learning automata [14]. Below we give a brief introduction to variable structure learning automata which will be used in this paper.

A variable-structure automaton is represented by sextuple $\langle \alpha, \beta, \Phi, P, G, T \rangle$, where β is a set of inputs, Φ a set of internal states, α a set of outputs, P denotes the state probability vector governing the choice of the state at each stage k, G is the output mapping, and T is a learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector. It is evident that the crucial factor affecting the performance of the variable structure learning automata, is the learning algorithm for updating the action probabilities.

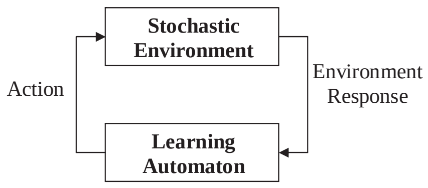

Fig. 4. Relation of learning automaton with its stochastic environment.

Learning automatons, based on the environment type, are divided into three groups [14]: (1) *P-models*: In these models the output of the environment can take only one of two values, 0 or 1. (2) *Q-models*: In these models the output of the environment can take a value in a set of discrete values in the range [0, 1]. (3) *S-models*: In these models the output of the environment is a continuous random variable with values in the range [0, 1]. The random environment can be *stationary* or *non-stationary*. In the first case, the punishment probabilities c_i's are constant while in the second case these probabilities can vary with time.

3.1. P-model variable structure learning automata

The general reinforcement scheme for these learning automata can be described as follows: assume that $p_k(n)$ is the probability of occurrence of action k at the current step n, $\boldsymbol{\alpha}(n)$ is the current action, and r is the number of actions of an automaton. The vector $p(n) = [p_1(n) \cdots p_r(n)]$ shows the action probabilities of the automaton. If $\boldsymbol{\alpha}(n) = \boldsymbol{\alpha}_i$, the probabilities of actions other than the current action i, are updated based on Eq. (3), and the probability of current action i is updated based on (4) [14].

$$p_j(n+1) = \begin{cases} p_j(n) - g_j(p(n)), & \boldsymbol{\beta}(n) = 0, \\ p_j(n) + h_j(p(n)), & \boldsymbol{\beta}(n) = 1, \end{cases} \tag{3}$$

$$p_i(n+1) = \begin{cases} p_i(n) + \sum\limits_{\substack{j=1 \\ j \neq i}}^{r} g_j(p(n)), & \boldsymbol{\beta}(n) = 0, \\ p_i(n) - \sum\limits_{\substack{j=1 \\ j \neq i}}^{r} h_j(p(n)), & \boldsymbol{\beta}(n) = 1. \end{cases} \tag{4}$$

It can be observed from the above equations that $\sum_{j \neq i} g_j(p(n))$ is used for rewarding the current action, $g_j(p(n))$ is used for punishment of action $j, j \neq i, \sum_{j \neq i} h_j(p(n))$ is used for punishment of the current action, and $h_j(p(n))$ is used for rewarding action $j, j \neq i$. Functions $h(\cdot)$ and $g(\cdot)$ usually have linear forms given by

$$\begin{aligned} g_j(p(n)) &= a p_j(n), \\ h_j(p(n)) &= \frac{b}{r-1} - b p_j(n). \end{aligned} \tag{5}$$

The automaton, defined by the above equation, is called $L_{\mathrm{R-P}}$ (Linear Reward–Penalty) for $a = b, L_{\mathrm{R-I}}$ (Linear Reward–Inaction) for $b = 0$, $L_{\mathrm{R-\varepsilon P}}$ (Linear Reward-epsilon-Penalty) for $a \gg b$, and $L_{\mathrm{I-P}}$ (Linear Inaction–Penalty) for $a = 0$ [14]. These names, also are used for other linear forms of functions $h()$ and $g()$. Generally speaking, for $L_{\mathrm{R-I}}$ we always have $h_j(p(n)) = 0$ and for $L_{\mathrm{I-P}}$ we always have $g_j(p(n)) = 0$.

3.2. S-model variable structure learning automata

This model differs from the P-model in that $\beta(n)$ is a continuous variable between 0 and 1. Therefore, the previous general reinforcement scheme (Eqs. (3) and (4)) can be generalized as follows [14]:

$$p_k(n+1) = \begin{cases} p_k(n) - (1-\beta(n))g_k(p(n)) + \beta(n)h_k(p(n)), & \alpha(n) \neq \alpha_k, \\ p_k(n) + (1-\beta(n))\sum_{j \neq k} g_j(p(n)) - \beta(n)\sum_{j \neq k} h_j(p(n)), & \alpha(n) = \alpha_k. \end{cases} \tag{6}$$

Again, functions $h(\cdot)$ and $g(\cdot)$ usually have linear forms. For $g_i(p(n)) = ap_i(n)$ and $h_i(p(n)) = 0$ $(0<a<1)$ the automaton is called $SL_{\mathrm{R-I}}$, and for $g_i(p(n)) = ap_i(n)$ and $h_i(p(n)) = a/(r-1) - ap_i(n)$ $(0<a<1)$ it is called $SL_{\mathrm{R-P}}$ [14]. Eqs. (7) and (8) show updating rules for action probabilities in these automata [14]:

$$SL_{\mathrm{R-I}}:\ p_i(n+1) = \begin{cases} p_i(n) - a(1-\beta(n))p_i(n), & \alpha(n) \neq \alpha_i, \\ p_i(n) + a(1-\beta(n))\sum_{j \neq i} p_j(n), & \alpha(n) = \alpha_i, \end{cases} \tag{7}$$

$$SL_{\mathrm{R-P}}:\ p_i(n+1) = \begin{cases} p_i(n) + \beta(n)\left(\dfrac{a}{r-1} - ap_i(n)\right) \\ \quad -a(1-\beta(n))p_i(n), & \alpha(n) \neq \alpha_i, \\ p_i(n) - a\beta(n)p_i(n) \\ \quad +a(1-\beta(n))\sum_{j \neq i} p_j(n), & \alpha(n) = \alpha_i. \end{cases} \tag{8}$$

For $g_j(p(n)) = 0$ and $h_j(p(n)) = bp_j(n)$ the automaton is called $SL_{\mathrm{I-P}}$, whose updating rule for action probabilities is as follows:

$$SL_{\mathrm{I-P}}:\ p_i(n+1) = \begin{cases} p_i(n) + b\beta(n)p_i(n), & \alpha(n) \neq \alpha_i, \\ p_i(n) - b\beta(n)\sum_{j \neq i} p_j(n), & \alpha(n) = \alpha_i. \end{cases} \tag{9}$$

4. The process of learning in the proposed network

After learning is completed, each RBF neuron represents a cluster, such that by presenting each data of a cluster to the learnt network, the RBF neuron, which represents this cluster, fires at a time which is the nearest firing time to a reference time in comparison with other RBF neurons. The desired time for the firing is the reference time which occurs when the input vector coincides with the center of the RBF neuron. The reference time is $T_{\text{ref}} + \text{Idealft}$ which will be explained later in Section 4.1.

Suppose that $(\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_m)^{\text{T}}$ temporally represents the center of a cluster. We want that the center of one of the RBF neurons finally coincides with this cluster. Based on Eq. (2) the goal will be as follows:

$$\mathbf{C} = (c_1, \ldots, c_m)^{\text{T}} = (T_{\text{ref}} - d_1, \ldots, T_{\text{ref}} - d_m)^{\text{T}} = (\hat{t}_1, \ldots, \hat{t}_m)^{\text{T}} \tag{10}$$

and consequently:

$$d_1 + \hat{t}_1 = d_2 + \hat{t}_2 = \cdots = d_m + \hat{t}_m = T_{\text{ref}}. \tag{11}$$

During learning, by presenting an input vector $(t_1, t_2, \ldots, t_m)^{\text{T}}$ from a cluster with center $(\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_m)^{\text{T}}$ to the network, the probability of delays on input synapses to an RBF neuron are updated based on the firing time of this RBF neuron, such that finally the Eq. (11) becomes true for one of the RBF neurons.

The model of learning automaton is used for learning in the network. One LA is assigned to each synapse (Fig. 5). Actions of each LA denote possible values for delays that a synapse may assume. For example, if LA has 50 actions which are delays in the range $[0, 10]$, then the actions of LA are $\alpha_1 = 0.0$, $\alpha_2 = 0.2, \ldots$, and $\alpha_{50} = 9.8$. The feedback signal to the LA from the environment is obtained from the firing time of the corresponding RBF neuron. Since the environment of an LA is an RBF neuron and the feedback signal to the LA is generated inside the network during its operation (i.e. without extra information from outside), the learning from the neural net point of view is an unsupervised learning.

At the beginning of each learning cycle, each LA selects one of its actions (synaptic delay) as a sample realization of its probability vector. Then one input vector is presented to the network. Based on the synaptic delays, the RBF neurons fire in different times (or do not fire at all). Then, for neurons that fire, if the firing time is close enough to the reference time then the environment provides a response indicating that the current actions or a set of actions of the learning automata

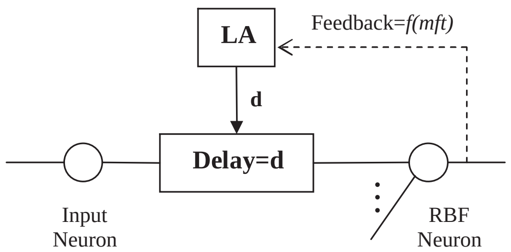

Fig. 5. The role of LA in the network. One learning automaton is associated to each synapse which emits the delay of the synapse in its output. The Feedback signal from the environment is a function of modified firing time of the corresponding RBF neuron.

associated to that RBF neuron are favorable. If the firing time is not close to the reference time, then the environment response indicating that the actions of the learning automata associated to that RBF neuron are unfavorable. For this network, various learning automata are tested in order to find a reliable and accurate learning method. In addition to the simple learning methods, two new LA-based strategies are proposed. These strategies are designed in such a way that the network works well for clustering tasks. We next discuss these strategies.

4.1. Simple strategies

At the beginning of each learning cycle (i.e. presenting each input to the network) each learning automaton is activated and generates an action as the corresponding synaptic delay. Firing time (*ft*) for each RBF neuron is determined by the synaptic delays and the current input. For neurons which do not fire, the firing time is considered to be infinity. The modified firing time (*mft*) for each neuron is calculated as follows:

$$mft = |ft - (T_{\mathrm{ref}} + Idealft)|, \tag{12}$$

where *Idealft* or ideal firing time is the time at which the RBF neuron fires if the PSPs of all the input synapses start to rise from the time zero. That is, if a zero input vector is fed to a network with m input neurons and one RBF neuron with a zero delay vector, then the firing time of the RBF neuron will be Idealft. It means that the ideal value for *mft* is zero and occurs when the input vector coincides with the center of the RBF neuron. Because, in this situation, the starting time of PSPs of all synapses of the RBF neuron ($t_i + d_i$'s) will be T_{ref} (Eq. (11)) and based on the definition of Idealft, the firing time will be $T_{\mathrm{ref}} + Idealft$, which results in $mft = 0$. The feedback signals from the environment for all LAs of the input synapses to the RBF neuron are computed as follows:

$$\boldsymbol{\beta}_1 = \min(1, mft/MODWID), \tag{13}$$

$$\boldsymbol{\beta} = \begin{cases} 1, & \boldsymbol{\beta}_1 > 0.5, \\ 0, & \boldsymbol{\beta}_1 \leqslant 0.5, \end{cases} \tag{14}$$

where the parameter MODWID is called modification width and is used as a measure of closeness of *ft* to the reference time, $T_{\mathrm{ref}} + Idealft$. Then the action probabilities for the learning automata are updated based on the signals $\boldsymbol{\beta}_1$ for the S-Model and $\boldsymbol{\beta}$ for the P-Model environment. From these equations we can see that lower *mft* causes more favorable responses and higher *mft* causes more unfavorable responses from the environment, and this leads to the minimization of *mft*. During the next learning cycles, one input from each cluster is presented to the network and the action probabilities are updated. This process continues until all inputs are used. Then, the whole process is repeated until the network converges or the number of iterations exceeds the maximum iteration parameter *Maxiteration*. Experimental results for $L_{\mathrm{R-I}}, L_{\mathrm{R-P}}, L_{\mathrm{R-\varepsilon P}}, SL_{\mathrm{R-I}}$, and $SL_{\mathrm{R-P}}$ methods are presented in Section 5.1.1.

4.2. Combined strategies without neighborhood modification

4.2.1. $SL_{\mathrm{R-I}} + SL_{\mathrm{I-P}}$ method

In this method, during each learning cycle the modified firing time is computed from Eq. (12). Then, if at least one neuron has been fired, the probabilities of actions of the LAs assigned to the input synapses of the neuron with minimum *mft* are updated according to the $SL_{\mathrm{R-I}}$ method. For other neurons which are fired and their *mft*'s are less than a threshold $MODWID\_H$ ($MODWID\_H > MODWID$), the probabilities of their synaptic delays are updated according to the $SL_{\mathrm{I-P}}$ method. In fact this combined strategy can be interpreted as a competitive learning process; the winner neuron (i.e. the neuron with minimum *mft*) whose center is close enough to the input data, is rewarded, the other neurons whose centers are more or less close to the input data are punished, and remaining neurons whose centers are far from the input are left without any change. The environment feedback signal for $SL_{\mathrm{R-I}}$ method is $\boldsymbol{\beta}_1$ and for $SL_{\mathrm{I-P}}$ method is $\boldsymbol{\beta}'_1 = 1 - \boldsymbol{\beta}_1$ ($\boldsymbol{\beta}_1$ is obtained from (13)). In the $SL_{\mathrm{I-P}}$ method, the closer *mft* is to time zero, the greater a penalty must be considered for the current action or a set of actions of LA. This is the reason why we have used $\boldsymbol{\beta}'_1 = 1 - \boldsymbol{\beta}_1$ for this punishment.

4.2.2. $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ method

This method is similar to the $SL_{\mathrm{R-I}} + SL_{\mathrm{I-P}}$ method except that the feedback signal is $\boldsymbol{\beta}$ for $L_{\mathrm{R-I}}$ and $\boldsymbol{\beta}' = 1 - \boldsymbol{\beta}$ for $L_{\mathrm{I-P}}$, where $\boldsymbol{\beta}$ is obtained from (14). Updating of action probabilities for $L_{\mathrm{R-I}}$ and $L_{\mathrm{I-P}}$ methods is performed according to (7) and (9), where the feedback signals in these equations are considered as above.

4.3. Combined strategies with neighborhood modification

These methods are generalizations of the $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ method. In these methods rather than rewarding or punishing only the current action of an LA, the neighboring actions of the current action or the neighboring actions of the optimal action are rewarded or punished, too. Note that actions of each learning automaton indicate synaptic delays.

4.3.1. Updating neighborhood of actions with optimal action as its center

In this case, if at least one neuron is fired, for the neuron with minimum *mft* the probabilities of actions in a neighborhood around the optimal action of the LAs corresponding to input synapses of that neuron are updated according to the $L_{\mathrm{R-I}}$ learning algorithm. For other neurons which are fired and their *mft*'s are less than $MODWID\_H$, the probabilities of actions in a neighborhood around the optimal actions of the learning automata corresponding to input synapses of those neurons are updated according to the $L_{\mathrm{I-P}}$ learning algorithm. Other neurons remain unchanged. The center of the neighborhood is the optimal action whose index

(Bai: Best action index) is calculated as follows:

$$Bai = Round\left(\frac{T_{\text{ref}} - t_i}{DBA}\right), \tag{15}$$

where t_i is the firing time of the neuron correspond to this synapse and Distance Between Actions (DBA) is equal to T_{ref}/r and indicates the time distance between actions, and r is the number of actions of the automaton. Optimal actions of the input synapses to an RBF neuron, indicate the delay vector which causes the center of the RBF neuron to coincide with the current input vector.

An interval around the optimal action with radius $RRS1$ (Radius of Reward Set 1) and another interval with radius $RRS2$ (Radius of Reward Set 2) such that $RRS1 < RRS2$ are defined. The actions in regions 1 and 2 as indicated in the Fig. 6 constitute the neighbors of the optimal (best) action.

The values of the radiuses of these intervals are calculated as follows:

$$RRS1 = Round\left(\frac{STDV}{DBA}\right), \quad RRS2 = Round\left(\frac{f^{*}STDV}{DBA}\right), \tag{16}$$

where $STDV$ is an approximation of the standard deviation of input clusters, f is a parameter with value greater than 1, and DBA is the distance between actions.

Using this neighborhood, two regions 1 and 2 are defined as shown in Fig. 6. These regions are named $RS1$ (Reward Set 1) and $RS2$ (Reward Set 2), respectively. Updating of action probabilities is performed in three steps as follows:

1. Applying the $L_{\text{R}-\text{I}}$ method with parameter a_1 for rewarding optimal action (Bai)

$$p_i(n+1) = \begin{cases} p_i(n) + a_1(1 - p_i(n)), & i = Bai, \\ p_i(n) - a_1 p_i(n), & i \neq Bai. \end{cases} \tag{17}$$

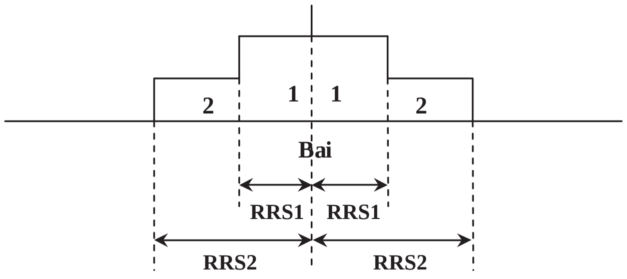

Fig. 6. Reward neighborhoods around the best action. The most reward is assigned to action Bai, and after it to actions in regions 1 and 2, respectively.

2. Applying the $L_{\mathrm{R-I}}$ method with parameter a_2 for rewarding the actions in $RS1$

$$p_i(n+1) = \begin{cases} p_i(n) - a_2 p_i(n), & \alpha_i \notin RS1,\ i \neq Bai, \\ p_i(n) + \dfrac{a_2}{NRS1} \displaystyle\sum_{\substack{j \notin RS1 \\ j \neq Bai}} p_j(n), & \alpha_i \in RS1, \end{cases} \tag{18}$$

where $NRS1$ denotes the number of actions in $RS1$.

3. Applying the $L_{\mathrm{R-I}}$ method with parameter a_3 for rewarding the actions in $RS2$

$$p_i(n+1) = \begin{cases} p_i(n) - a_3 p_i(n), & \alpha_i \notin RS2,\ \alpha_i \notin RS1,\ i \neq Bai, \\ p_i(n) + \dfrac{a_3}{NRS2} \displaystyle\sum_{\substack{j \notin RS1 \\ j \notin RS2 \\ j \neq Bai}} p_j(n), & \alpha_i \in RS2. \end{cases} \tag{19}$$

where $NRS2$ denotes the number of actions in $RS2$.

Depending on the size of the intervals, parameters a_1, a_2, and a_3 are determined in such a way that the optimal action earns maximum reward, actions in $RS1$ earn lower, and actions in $RS2$ earn the lowest reward.

If Bai is very close to action zero or action $r-1$, such that parts of the neighborhood lie outside of the range of actions, then the intervals as defined in Fig. 6 will be shortened. That is the values $NRS1$ and/or $NRS2$ are updated to indicate this change. Using the above equations for updating action probabilities is no problem as a result of shortening these intervals.

The above learning process was for the winner neuron. For other neurons with *mft*'s less than *MODWID_H*, the optimal action and the corresponding neighborhoods are obtained similarly and the $L_{\mathrm{I-P}}$ method is used to punish the actions in the total neighborhood (regions 1 and 2 in Fig. 6). The index of the best action (*Bai*) and the corresponding intervals are obtained as before but here the names $RRS1$, $RRS2$, $NRS1$, $NRS2$, $RS1$, and $RS2$ denoting reward (R) are replaced with names $RPS1$, $RPS2$, $NPS1$, $NPS2$, $PS1$, and $PS2$ denoting penalty (P). Also in the relations (17)–(19), reward parameters a_1, a_2, and a_3 are replaced with penalty parameters b_1, b_2, and b_3 as given in relations (20)–(22). Updating of action probabilities for these neurons are performed in three following steps:

1. Applying the $L_{\mathrm{I-P}}$ method with parameter b_1 to punish optimal action (*Bai*)

$$p_i(n+1) = \begin{cases} p_i(n) - b_1 p_i(n), & i = Bai, \\ \dfrac{b_1}{r-1} p_{Bai}(n) + p_i(n), & i \neq Bai. \end{cases} \tag{20}$$

2. Applying the $L_{\mathrm{I-P}}$ method with parameter b_2 to punish the actions in $PS1$

$$p_i(n+1) = \begin{cases} p_i(n) - b_2 p_i(n), & \alpha_i \in PS1, \\ p_i(n) + \dfrac{b_2 \sum_{j \in PS1} p_j(n)}{r-1-NPS1}, & \alpha_i \notin PS1,\ i \neq Bai. \end{cases} \tag{21}$$

3. Applying the $L_{\mathrm{I-P}}$ method with parameter b_3 to punish the actions in $PS2$

$$p_i(n+1) = \begin{cases} p_i(n) - b_3 p_i(n), & \alpha_i \in PS2, \\ p_i(n) + \dfrac{b_3 \sum_{j \in PS2} p_j(n)}{r - 1 - NPS1 - NPS2}, & \alpha_i \notin PS2,\ \alpha_i \notin PS1,\ i \neq Bai. \end{cases} \tag{22}$$

Again, parameters b_1, b_2, and b_3 are determined depending on the size of the intervals in such a way that the optimal action earns maximum penalty, actions in $PS1$ earn lower, and actions in $PS2$ earn the lowest penalties. In this method the environment feed-back signals are computed like the $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ signals in the $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ method. Note that relations (17)–(19) are used when $\boldsymbol{\beta} = 0$ and relations (20)–(22) are used when $\boldsymbol{\beta}' = 1$.

4.3.2. *Updating a neighborhood of actions with current action as its center*

In this method the center of the neighborhood is assumed to be the current action of LA. Therefore, the method is like the previous method except that the index Cai (Current action index) is used in place of Bai.

Decreasing learning parameters: A heuristic rule which has shown good results in some unsupervised learning methods, such as the SOM of Kohonen [9], is to use a decreasing learning rate during the learning process. We tested this heuristic rule and allow the learning parameters a_1, a_2, a_3 and b_1, b_2, b_3 in Eqs. (17)–(22) to decrease linearly. As shown in Table 4, this rule produces better clustering results in real world problems.

Definition of convergence: Before we present simulation results, we give our definitions of convergence.

- *Convergence of an LA*: When an action (or a set of at most $2^{*}CONVWID + 1$ adjacent actions) has a probability of more than $CONVTHR$, we say that the LA has converged.
- *Convergence of RBF neurons and network*: If all LAs of synapses of an RBF neuron have converged, we say that the neuron has converged. If all of the neurons in a network have converged, we say that the network has converged. We stop an experiment when the network converges in the above sense or when we have reached the predefined maximum number of iterations.
- *Convergence of an RBF neuron to a cluster*: If an RBF neuron for almost all samples of a cluster has the lowest *mft* among all RBF neurons, and almost all samples of other clusters do not have the lowest *mft*, we say that the neuron has converged to the cluster.

5. Evaluating the model

The evaluation of the proposed model is presented in this section. In Section 5.1 various LA-based methods are first tested on synthesized problems. Then, the method which has shown the best performance in these experiments is tested on several difficult clustering data sets, in Section 5.2.

5.1. Clustering of synthesized data sets

In this section, various learning methods presented in Section 4 are evaluated using a simple two-dimensional (2-D) clustering problem and the 40-D problem used in [18]. It is shown that the accuracy of the proposed net is better than that of the network discussed in [18]. For all experiments reported in this section, learning parameters are fixed throughout the experiments.

5.1.1. Two-dimensional problem clustering

A 2-D clustering problem with nine clusters is considered. Each element of the input vector is in $[0, T_{\mathrm{ref}}]$, where $T_{\mathrm{ref}} = 10\,\mathrm{ms}$. Each cluster contains ten samples and the distribution of clusters is Gaussian with a standard deviation of 0.2 ms (Fig. 7).

For this problem, a network with two input neurons ($m = 2$) and nine output neurons ($n = 9$) is used. The learning cycle duration is 50 ms and the refractory period is 30 ms. The values of the other parameters are selected as follows: $T_{\mathrm{ref}} = 10\,\mathrm{ms}$, $\tau = 3\,\mathrm{ms}$, $Thr = 1.93$, $Idealft = 2.265\,\mathrm{ms}$, $MODWID = 0.5\,\mathrm{ms}$, $MODWID\_H = 1.57\,\mathrm{ms}$, $STDV = 0.2$, $r = 50$ (number of actions of each LA), $f = 2$ (in Eq. (16)), and $Maxiteration = 200$. The initial values of the synaptic delays are selected randomly in the $[0, T_{\mathrm{ref}}]$ interval (Fig. 8). Two additional parameters $CONVWID = 4$ and $CONVTHR = 0.95$ are used for defining the convergence of LAs, as mentioned at the end of Section 4.3. As stated before, the total number of input patterns is 90 and these patterns are presented to the network at most Maxiteration times. Variable ρ counts the number of these presentations. For each method, we have repeated the experiment with 20 different random initializations. Tables 1–3 show the results. In these tables, the second column shows the average percentage of the number of correctly converged neurons (i.e. the neurons which are

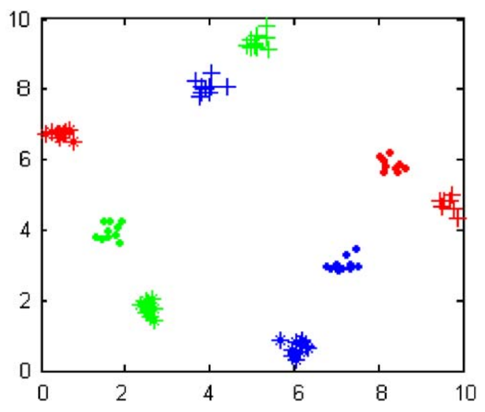

Fig. 7. 2-D clustering problem used for testing the network.

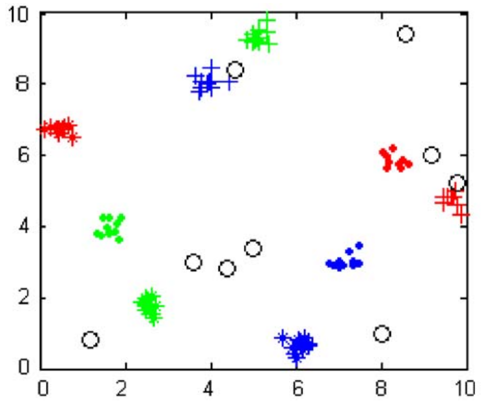

Fig. 8. Initial values for the RBF neurons centers (circles) which are selected randomly.

Table 1
Experimental results for simple strategies on 2-D problem

Method	Average number of correctly converged neurons (%)	Average network convergence rate (%)	Average ρ
$L_{\mathrm{R-I}}$	61.66	100	28.75
$L_{\mathrm{R-P}}$	0	0	—
$L_{\mathrm{R-\varepsilon P}}$	0	0	—
$SL_{\mathrm{R-I}}$	60.00	100	18.50
$SL_{\mathrm{R-P}}$	0	0	—

Table 2
Experimental results for combined strategies without neighborhood modification on 2-D problem

Method	Average number of correctly converged neurons (%)	Average network converged rate (%)	Average ρ
$SL_{\mathrm{R-I}} + SL_{\mathrm{I-P}}$	72.77	100	13.95
$L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$	70.13	100	10.63

Table 3
Experimental results for combined strategies with neighborhood modification on 2-D problem

Method	Average number of correctly converged neurons (%)	Average network converged rate (%)	Average ρ
Optimal action as the center	100	100	26.45
Current action as the center	74.44	20	81.75

converged to a separate cluster center) in 20 simulation runs. The third column shows the percentage of times that the network is converged. The fourth column shows the average number of iterations required for the network convergence. The definition of convergence is presented at the end of Section 4.3. The results of various learning methods are discussed below in more details.

Results for simple strategies: Table 1 shows the experimental results for simple strategies. The learning parameters used for various methods are as follows: $a = 0.1$ for $L_{\mathrm{R-I}}$, $a = b = 0.1$ for $L_{\mathrm{R-P}}$, $a = 0.1$ and $b = 0.001$ for $L_{\mathrm{R-\varepsilon P}}$, $a = 0.2$ for $SL_{\mathrm{R-I}}$, and $a = b = 0.1$ for $SL_{\mathrm{R-P}}$.

The results of the experiments show that among all the above methods, only for $L_{\mathrm{R-I}}$ and $SL_{\mathrm{R-I}}$, which are pure reward methods, the network has completely converged. Even for these two methods not all the clusters are detected (only about 60% of neurons have converged correctly to a separate cluster center). It seems that such simple methods are not capable of performing a suitable clustering task.

Results for combined strategies without neighborhood modification: Table 2 shows the experimental results for combined strategies without neighborhood modification. The learning parameters used for these two methods are $a_1 = 0.3$ and $a_2 = 0.03$. Through the simulation runs, the network has completely converged for both methods, as shown in the third column of Table 2, but still not all clusters are detected (only about 70% of neurons have converged correctly). The fourth column of Table 2 shows that the convergence occurs faster than the simple strategies.

In the combined strategies, as explained in Section 4.2, a competitive learning method is considered, where the winning neuron is updated using $SL_{\mathrm{R-I}}$ or $L_{\mathrm{R-I}}$ method, and other neurons whose centers are close to the winner's center are updated using $SL_{\mathrm{I-P}}$ or $L_{\mathrm{I-P}}$. The performance of this strategy is better than that of the simple strategies, because of its competitive nature and simultaneous consideration of pure reward and pure punishment for different neurons.

Results for combined strategies with neighborhood modification: Table 3 shows the experimental results for combined strategies with neighborhood modification. The two following methods are evaluated:

(a) Updating a neighborhood of actions with the optimal action as its center.
(b) Updating a neighborhood of actions with the current action as its center.

For both methods, the learning parameters are selected as follows: $a_1 = 0.3$, $a_2 = 0.3$, $a_3 = 0.1$, $b_1 = 0.2$, $b_2 = 0.2$, $b_3 = 0.05$. The values of neighborhood parameters with $f = 2$ in (16) are $RRS1 = RPS1 = 1$, $RRS2 = RPS2 = 3$, $NRS1 = NPS1 = 2$, and $NRS2 = NPS2 = 4$. Here, the value of $CONVWID$ is selected to be equal to $RRS2$.

Through all the simulation runs for method (a), the network has completely converged and all neurons have correctly converged to cluster centers, as shown in Table 3. But for method (b), only in 20% of simulation runs the network has converged.

The difference between the combined strategies with and without the neighborhood modification, as explained in Section 4.3, is the use of a new updating rule. In the new updating rule, instead of only one action, a set of neighboring actions are updated. Based on this modification which improves the reliability of the clustering task, in addition to the action corresponding to the current neuron center, the actions in its neighborhood are also updated. This means that instead of one point, an area in the input space is rewarded (or punished) in each updating step. Note that the actual center of a cluster is more probable to be inside an area than to coincide exactly with a point.

In case (a) above, the optimal action which is the action coinciding with the current input data, is considered as the center of the neighborhood; while in case (b), the current action which may have a small distance from the current input data is considered as the center of the neighborhood. The neighborhood of case (a) which is symmetric around the current data seems to be a better candidate for action probability updating than the neighborhood of case (b) which is not symmetric around the current data. The reason is that the later may result in rewarding (or punishing) some irrelevant actions. As shown in Table 3, the performance of case (a) is better than that of case (b).

Selecting the best LA-based method: To compare the performance of various LA-based methods, the reliability and speed of convergence can be considered as the evaluation factors. An acceptable clustering method is expected to perform a reliable correct clustering for the above 2-D problem which is a moderately simple clustering case. Therefore, the reliability is more critical than the speed in the above experiments. Tables 1–3 show that in 20 simulation runs the network has always converged only for five learning methods (the methods with 100% network convergence in the third columns of the Tables 1–3). Among these methods, only one method has always correctly converged to cluster centers (the method with 100% correctly converged neurons in second column of Table 3). Therefore, this method is the only reliable method among the various LA-based methods. The number of required iterations for this method is not minimal, but it is acceptable in comparison with other methods (the fourth column of the Tables 1–3). As mentioned before, the speed of convergence is not as important as the reliability in the above experiments.

From these experiments, we may conclude that the method $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ with updating neighborhoods centered on the optimal action has a better performance compared to the other methods. For this reason, this method is selected for conducting more experiments in the rest of the paper.

Output modification: In the previous experiments, the learned centers were identified based on the action of each LA which has the highest probability of selection and therefore each element of a center vector was a discrete quantity. This results in a quantization error in determining the centers. To avoid such an error, we consider the lth element of the center of a jth RBF neuron as follows:

$$c_j^l = \frac{\sum_{i \in S(i_0)} (T_{\mathrm{ref}} - d_{ji}^l) p_{ji}^l}{\sum_{i \in S(i_0)} p_{ji}^l}; \quad j = 1, \ldots, n; \quad l = 1, \ldots, m, \tag{23}$$

where d_{ji}^l is the synaptic delay of ith action of the automaton in lth dimension of the jth RBF neuron, i_0 is the index of the action with most probability in this

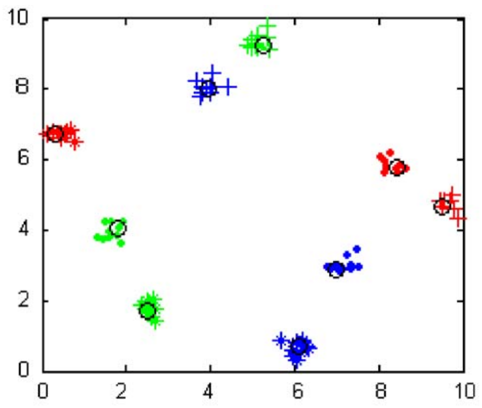

Fig. 9. The result of $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ method with updating neighborhood centered on the best action and modified output.

automaton, $S(i_0)$ is an interval of the actions of the automaton with center i_0 and radius CONVWID, p^l_{ji} is the probability of ith action of the automaton, c^l_j is the 1th element of the center of jth RBF neuron, m is input dimensionality, and n is the number of RBF neurons. Based on (23), not only the action with most probability but also other actions in an interval around this action contribute (proportional to their probabilities) to determining the centers of the clusters. Fig. 9 shows the centers obtained from $L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ Method with updating neighborhood centered on the optimal action, modified according to Eq. (23).

Experiments with various numbers of RBF neurons: Up to now, we considered the number of RBF neurons equal to the number of clusters. But in many applications for clustering the number of clusters is not known a priori. Thus, it is useful to evaluate the effect of the number of RBF neurons (n) on the performance of the proposed method. In the previous 2-D problem the number of clusters is $p = 9$. We showed results for the case in which the number of neurons is equal to the number of clusters ($n = p$). Now two cases are considered: first, the number of neurons is lower than the number of clusters ($n<p$), and second, the number of neurons is greater than the number of clusters ($n>p$).

In the first case ($n<p$), n neurons have always converged to n clusters and other clusters remain without prototype. In the second case ($n>p$), also p neurons have converged to p clusters and the remaining neurons have converged near to the cluster centers or remain unconverged. Therefore, the performance of the method in these cases is suitable and does not lead to incorrect clustering. It means that for an application, if the number of clusters is not known a priori, we can increase the number of RBF neurons one by one and repeat the learning process until the new neuron converges near the previous ones. In this step, the number of clusters is found. This number is the number of neurons whose centers are not very close to each other.

5.1.2. Forty-dimensional clustering problem

In this section, the performance of the proposed network is evaluated for high dimensional input data and compared with the network proposed in [18]. Experiments are performed on a 40-D clustering problem similar to one used in [18]. A 40-D clustering problem with 3 clusters is considered. Each element of the input vector is in $[0, T_{\mathrm{ref}}]$ interval ($T_{\mathrm{ref}} = 10\,\mathrm{ms}$). Each cluster contains 50 samples. The distribution of clusters is Gaussian with standard deviation of 0.1 ms.

The characteristics of the network, except for the following items, are similar to that of the network used for previous 2-D problem: $m = 40$ (number of input neurons), $n = 3$ (number of RBF neurons), $Thr = 25$, $Idealft = 0.9447$, $STDV = 0.1$, and $CONVTHR = 0.9$.

The results of the previous method ($L_{\mathrm{R-I}} + L_{\mathrm{I-P}}$ method with updating neighborhood centered on the optimal action, without decreasing learning parameters) are acceptable. In all of the ten experiments conducted with this method with different random initializations, the RBF neurons converged to the cluster centers. For the sake of comparison we implemented the network and learning method proposed in [18] with the same parameters reported there. In the proposed method the average number of required iterations and the average mean square error are $\rho_{\mathrm{avg}} = 8.9$ and $MSE_{\mathrm{avg}} = 0.00232$, respectively, while these values for the method proposed in [18] are $\rho_{\mathrm{avg}} = 7.6$ and $MSE_{\mathrm{avg}} = 0.04268$, respectively. It can be seen that with comparable number of iterations, the accuracy of clustering in the proposed method is considerably higher. One reason for this higher accuracy is the selection of parameter r equal to 50 which leads to more number of delays in each connection than that of the network proposed in [18], where number of delays in each synaptic connection, is equal to the number of synaptic terminals (16 in [18]) which is fixed and is part of definition of the network structure. In the experiments the mean square error is computed according to the following equation:

$$MSE = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(c_j^i - u_j^i)^2}{n^*m}, \tag{24}$$

where n is the number of RBF neurons (number of clusters) (here $n = 3$), m is the input dimensionality (here $m = 40$), c_j^i is the ith element of the center of jth RBF neuron, and u_j^i is the ith element of the true center of jth cluster.

5.2. Clustering real-world data sets

The final set of experiments is conducted on complex multidimensional data of real-world problems to compare the clustering ability of the proposed net and the *k-means* clustering method. Several standard benchmark databases have been selected from the UCI Repository of Machine Learning Databases [13] for this purpose. In all experiments, we have split the data set into two disjoint subsets. The training subset has been presented to the network in the training phase, while the test subset has been used to measure the classification performance of the trained network. Each of the two subsets has 50% of the samples.

Table 4
Correct classification percentage on real world problems

Problem	Proposed method without decay	Proposed method with decay	*k-means*
BalanceScale	55.290	60.924	63.871
Glass	35.922	35.922	35.922
Hayes-roth	43.846	47.293	40.769
Ionosphere	72.162	74.007	44.828
Iris	71.667	86.111	74.444
PimaIndiansDiabetes	65.183	65.183	65.445

Table 4 shows the correct classification rates resulting for the proposed and *k-means* methods. The second and third columns in this table indicate the cases where the proposed method learning parameters (of Eqs. (17)–(22)) are fixed and are decreasing, respectively. In these experiments, all samples of the training subset are presented to the network. After completion of the training, the winning neuron for all the samples of the training subset is determined. For every neuron, its receptive field, i.e. the set of training samples for which this neuron is the winner, is then computed. Each neuron is then assigned to the class with the most training samples in its receptive field. The test samples are then presented to the network one by one. If the winning neuron corresponds to the class of the test sample, we count it as a successful classification. Otherwise, it is a classification failure. We have repeated each experiment for five different random initializations. The average of the classification rates obtained for these runs is reported in Table 4.

The parameter values used in the above experiments are as follows: $Thr = m/2$, $r = 200$, $DBA = T_{\text{ref}}/r$, $Idealft = 0.696$, $MODWID = 1.9$, $MODWID\_H = 2.7$, $STDV = 0.15$, $f = 1.6$, $a_1 = 0.3$, $a_2 = 0.3$, $a_3 = 0.1$, $b_1 = 0.2$, $b_2 = 0.2$, and $b_3 = 0.05$. Note that the above values of the learning parameters ($a_1 - a_3$ and $b_1 - b_3$) for the method with decay are their initial values. Parameter m shows the number of input neurons which is fixed to be equal to the dimensionality of the problem. Parameter n, the number of RBF neurons, is equal to the number of classes in each problem, and T_{ref} is set to 10 ms for all data sets. All data sets are initially normalized in the interval $[0, 10]$. For the *k-means* method, parameter k is equal to the number of classes for each problem.

Results reported in Table 4 show that decreasing the learning parameters during the learning process improves the performance of the proposed method. Table 4 also shows that the performance of the proposed method with decreasing learning parameters is generally better than the *k-means* method. This is because that *k-means* method always finds the hyper-spherical clusters, while the proposed method considers limited elongations by tuning its parameters.

6. Conclusion

In this paper, a new version of the RBF network reported by Ruf et al. [15,18] was proposed. In the proposed network, rather than using a Hebb rule the model of learning automaton has been used for realization of delay shift in synaptic terminals of the network. The proposed network provides simpler solutions to the problems in Ruf's RBF network as reported in [1], i.e. the clustering of low dimensional inputs is performed correctly, as compared to the net of [18] which was unable to do this, and the clustering of high dimensional inputs is performed more precisely than by the net of [18]. The proposed method was tested on a number of real world problems. Experiments showed the superiority of the proposed method over the *k-means* method.

Acknowledgments

The authors would like to thank the reviewers for their careful and constructive criticism of the original manuscript.

References

[1] S.M. Bohte, A.P. Han, N.K. Kok, Unsupervised clustering with spiking neurons by sparse temporal coding multi-layer RBF networks, in: Proceeding of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000, pp. 279–284.

[2] S.M. Bohte, J.N. Kok, H. La Poutre, Error-backpropagation in temporally encoded networks of spiking neurons, Neurocomputing 48 (2002) 17–37.

[3] Y. Choe, R. Miikkulainen, Self-organization and segmentation in a laterally connected orientation map of spiking neurons, Neurocomputing 21 (1998) 139–157.

[4] C.W. Eurich, K. Pawelzik, U. Ernst, A. Thiel, J.D. Cowan, J.G. Milton, Delay adaptation in the nervous system, Neurocomputing 32 (2000) 741–748.

[5] C.W. Eurich, K. Pawelzik, U. Ernst, A. Thiel, J.D. Cowan, J.G. Milton, Dynamics of self-organized delay adaptation, Phys. Rev. Lett. 82 (2000) 1594–1597.

[6] W. Gerstner, Spiking neurons, in: W. Mass, C.M. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, MA, 1998, pp. 3–53 (Chapter 1).

[7] G.E. Hinton, A. Brown, Spiking Boltzmann machines, in: Advances in Neural Information Processing Systems, vol. 12, MIT Press, Cambridge, MA, 2000.

[8] J.J. Hopfield, Pattern recognition computation using action potential timing for stimulus representation, Nature 376 (1995) 33–36.

[9] T. Kohonen, Self-Organizing Maps, third ed., Springer, Berlin, 2001.

[10] W. Maass, Network of spiking neurons: the third generation of neural network models, Neural Networks 10 (9) (1997) 1659–1671.

[11] W. Maass, Computing with spiking neurons, in: W. Mass, C.M. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, MA, 1999, pp. 55–85 (Chapter 2).

[12] W. Maass, T. Natschläger, Networks of spiking neurons can emulate arbitrary hopfield nets in temporal coding, Network: Comput. Neural System 8 (9) (1997) 355–372.

[13] P.M. Murphy, UCI Repository of Machine Learning Databases and Domain Theories [online], http://www.ics.uci.edu/~mlearn/MLRepository.html, June 2004.

[14] K.S. Narendra, M.A.L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, New Jersey, 1989.

[15] T. Natschläger, B. Ruf, Spatial and temporal pattern analysis via spiking neurons, Network: Comput. Neural Systems 9 (3) (1998) 319–332.

[16] J. O'Keefe, M.L. Recce, Phase relationship between hippocampal place units and the hippocampal theta rhythm, Hippocampus 3 (1993) 317–330.

[17] D.I. Perrett, E.T. Rolls, W.C. Caan, Visual neurons responsive to faces in the monkey temporal cortex, Exp. Brain Res. 47 (1982) 329–342.
[18] B. Ruf, Computing and learning with spiking neurons—theory and simulations, Ph.D. Dissertation, Institute for Theoretical Computer Science Technical University Graz, Austria, 1998.
[19] B. Ruf, M. Schmitt, Self organization of spiking neurons using action potential timings, IEEE Trans. Neural Networks 9 (3) (1998) 575–578.
[20] F.T. Sommer, T. Wennekers, Associative memory in networks of spiking neurons, Neural Networks 14 (2001) 825–834.
[21] S.T. Thorpe, M. Imbert, Biological constraints on connectionist modeling, in: R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, L. Steels (Eds.), Connectionism in Perspective, Elsevier, North-Holland, Amsterdam, 1989, pp. 63–92.
[22] C. Von der Malsburg, The correlation theory of brain function, Internal Report 81-2, Department of Neurobiology, Max-Planck-Ins. for Biophysical Chemistry, Gottingen, Germany, 1981.

Peyman Adibi was born in Isfahan, Iran, in 1975. He received the B.S. degree in computer engineering from Isfahan University of Technology, and M.S. degree in computer engineering from Amirkabir University of Technology, in 1998 and 2001, respectively. He is currently pursuing the Ph.D. degree in computer engineering at Amirkabir University of Technology. His research interests include neural networks, computer vision, image processing, and computer networks.

Mohammad Reza Meybodi received the B.S. and M.S. degrees in economics from Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.D. degrees from Oklahoma University, USA, in 1980 and 1983, respectively in computer science. Currently he is a full Professor in computer engineering department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983–1985 as an assistant professor at Western Michigan University, and from 1985–1991 as an associate professor at Ohio University, USA. His research interests include, channel management in cellular networks, learning systems, parallel algorithms, soft computing and software development.

Reza Safabakhsh was born in Isfahan, Iran, in 1953. He received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1976, and the M.S. and Ph.D. degrees in electrical engineering from the University of Tennessee, Knoxville, in 1979 and 1986, respectively.

He Worked at the University of Tennessee, Knoxville, TN, USA, from 1977 to 1983, at Walters State College, Morristown, TN, USA, from 1983 to 1986, and at the Center of Excellence in Information Systems, Nashville, TN, USA, from 1986 to 1988. Since 1988, he has been with the Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, where he is currently a Professor and the head of the Artificial Intelligence Group and the director of the Computational Vision/Intelligence Laboratory. His current research interests include spiking neural nets, self-organizing maps, digital watermarking, content-based image retrieval, and cooperation and learning in multiagent systems.

Dr. Safabakhsh is a member of the IEEE and several honor societies, including Phi Kappa Phi and Eta Kapa Nu. He is the Founder and a member of the Board of Executives of the Computer Society of Iran, and was the President of this society for the first 4 years.