International Journal of Neural Systems, Vol. 17, No. 3 (2007) 171–181

World Scientific
www.worldscientific.com

AN SOM-BASED ALGORITHM FOR OPTIMIZATION WITH DYNAMIC WEIGHT UPDATING

YI-YUAN CHEN and KUU-YOUNG YOUNG*
Department of Electrical and Control Engineering
National Chiao-Tung University, Hsinchu, Taiwan
National Chiao-Tung University Vision Research Center
**kyoung@mail.nctu.edu.tw.*

Received 4 May 2006
Revised 15 March 2007
Accepted 4 May 2007

The self-organizing map (SOM), as a kind of unsupervised neural network, has been used for both static data management and dynamic data analysis. To further exploit its search abilities, in this paper we propose an SOM-based algorithm (SOMS) for optimization problems involving both static and dynamic functions. Furthermore, a new SOM weight updating rule is proposed to enhance the learning efficiency; this may dynamically adjust the neighborhood function for the SOM in learning system parameters. As a demonstration, the proposed SOMS is applied to function optimization and also dynamic trajectory prediction, and its performance compared with that of the genetic algorithm (GA) due to the similar ways both methods conduct searches.

Keywords: Self-Organizing Map; optimization; dynamic function; genetic algorithm.

1. Introduction

The self-organizing map (SOM), as a kind of unsupervised neural network, is performed in a self-organized manner in that no external teacher or critic is required to guide synaptic changes in the network.[4,13] By contrast, for the other two basic learning paradigms in neural networks, supervised learning is performed under the supervision of an external teacher[8] and reinforcement learning involves the use of a critic that evolves through a trial-and-error process[3]; these other two also demand the input-output pairs as the training data. The appealing features of learning without needing the input-output pairs makes the SOM very attractive when dealing with varying and uncertain data. In its many applications, the SOM has been used for both static data management and dynamic data analysis, such as data mining, knowledge discovery, clustering, visualization, document browsing, text archiving, image retrieval, speaker recognition, mobile communication, robot control, identification and control of dynamic systems, local dynamic modeling, nonlinear control, and tracking moving objects.[1,2,8,11,13,14,17–20,22] There have also been many approaches proposed to improve or modify the original SOM algorithm for different purposes.[2,7,12,19,23–25] However, from our survey, its search abilities have not been adequately exploited yet.[5,6,9,15,16,21] This need thus motivates us to propose an SOM-based algorithm (SOMS) for optimization problems involving both static and dynamic functions.

There have been several applications of the SOM for optimization problems.[9,15,21] Michele *et al.* proposed a learning algorithm for optimization based on the Kohonen SOM evolution strategy (KSOM-ES).[15] In this KSOM-ES algorithm, the adaptive grids are

*Corresponding author.

used to identify and exploit search space regions that maximize the probability of generating points closer to the optima. Su *et al.* proposed an SOM-based optimization algorithm (SOMO).[21] Through the self-organizing process in SOMO, solutions to a continuous optimization problem can be simultaneously explored and exploited. Our proposed SOMS will extend the application further to optimization problems involving dynamic functions. When searching for a dynamic function, the goal may be to look for a set of optimal parameters that lead to the desired performance of the dynamic system from limited measured data. For instance, in a missile interception application, the task may be to predict the most probable launching position and velocity of an incoming missile from the measured radar data. Thus, the proposed SOMS should be able to execute both system performance evaluation and the subsequent search in a real-time manner.

In developing the SOMS, we will first examine the SOM regarding its learning strategy and search ability. As the dynamic system is tackled, the SOM learning may involve system parameters that fall in quite different ranges, e.g., position and velocity. To achieve high learning efficiency under such widely varying parameters, we propose a new weight updating rule which may dynamically adjust the shape and location of the neighborhood function for the SOM, in an individual basis, in learning the system parameters. Meanwhile, at the current stage, the proposed SOMS is effective for optimization problems with one optimum. The rest of this paper is organized as follows. Evaluation of the SOM search ability and the proposed SOMS are discussed in Sec. 2. The proposed weight updating rule is described in Sec. 3. To evaluate its effectiveness, in Sec. 4, the SOMS is applied to both function optimization and dynamic trajectory prediction. The performance is especially compared with that of the genetic algorithm (GA), since these two learning algorithms exhibit similarities when searching. Finally, conclusions are given in Sec. 5.

2. Proposed SOM-Based Search Algorithm (SOMS)

Before proposing the SOMS, we first evaluate the SOM by its search ability. The SOM, first introduced by Kohonen, transforms input vectors into a discrete map (e.g., a 2-D grid of neurons) in a topological ordered fashion adaptively.[13,19] During each iteration of learning, each neuron competes with each other to gain the opportunity to update its weight, and the one that generates the output most close to the desired value (vector) is chosen as a winner. Because the SOM allows local interaction between neighboring neurons, the weights of the winner and also those of its neighbors are all updated. Through repeated weight modification, a cluster (or clusters) may form and become more and more compact until a final configuration develops. The SOM thus has a structure very suitable for parallel processing. We further exploit this parallelism and design an organized search. In other words, we take advantage of the SOM in its distribution of the neurons in a grid pattern and the presence of local interaction in between the grid. Take the missile interception application as an example again. We may distribute the possible launching positions and velocities of the missile (as weight vectors) into the network in an organized fashion. Under this arrangement, the searches among the neurons are closely related through the grid, leading to a more rapid convergence.

Figure 1(a) shows the proposed SOMS, which consists of mainly the evaluation and search mechanisms and the dynamic model that stands for the target system. Initially, the function for performance evaluation is installed in the evaluation mechanism, and possible solutions (e.g., vectors of dynamic parameters), selected from the estimated range, will be distributed among the neurons of the SOM. During each time interval of the learning process, each of all the possible solutions in the neurons is sent to the dynamic model one by one. In other words, the dynamic model will be equipped with a possible set of dynamic parameters repeatedly, when used to derive the output data corresponding to the target system. The evaluation mechanism will then compute the difference between the derived data and the incoming measured data. From the results, the search mechanism chooses the solution leading to the most accurate derived data as the winner, and updates the weights of this winner and its neighboring neurons. Note that this SOMS can also be applied to continuous optimization problems, with the dynamic model replaced by the objective function for a given optimization problem and the input by the reference data.

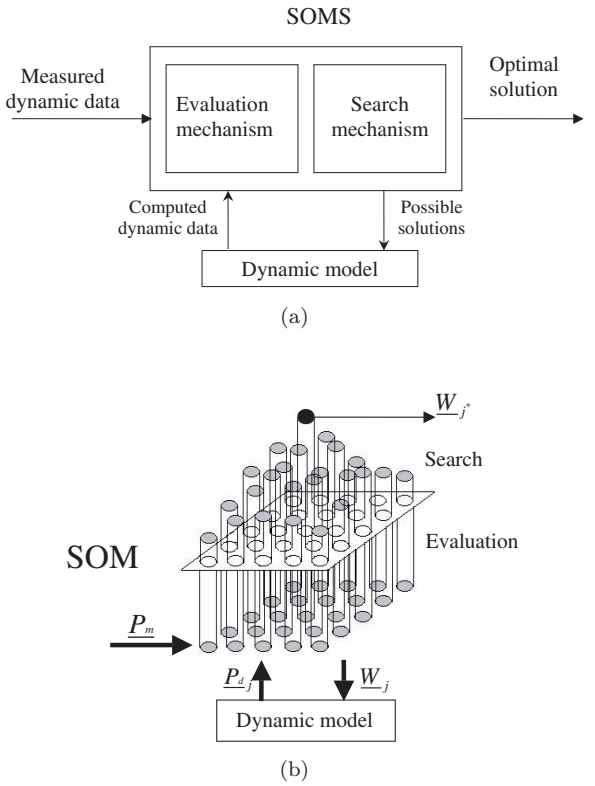


Fig. 1. (a) Proposed SOM-based algorithm for optimization. (b) The structure and operation of the SOM in the SOMS.

The SOM, shown in Fig. 1(b), executes the functions of search and evaluation in the SOMS. In Fig. 1(b), each neuron j in the SOM contains a vector of a possible solution set $\underline{W}_j$ (the weight vector). Each time new measured data $\underline{P}_m$ are sent into the scheme, the SOM is triggered to operate. All of the possible solution sets in the neurons will then be sent to the dynamic model to derive their corresponding data $\underline{P}_{d_j}$. The SOM evaluates the difference between $\underline{P}_m$ and each $\underline{P}_{d_j}$. Of all the neurons, it chooses the neuron j^*, which corresponds to the smallest difference, as the winner. The learning process then continues, and the network will eventually converge to the optimal solution. And even when the optimal solution is not within the estimated range for some cases, the search mechanism is still expected to move the possible candidates out of their initial locations and guide them to converge to the optimal solution.

3. Proposed Weight Updating Rule

For effective weight updating in the SOM, the topological neighborhood function and learning rate need to be properly determined. Their determination may depend on the properties of the system parameters to learn. As mentioned above, system parameters may operate in quite different working ranges. To achieve high learning efficiency, the weight updating should be executed on an individual basis, instead of using the same neighborhood function for all the parameters. We thus propose a new SOM weight updating rule which can dynamically adjust the center and width of their respective neighborhood function for the SOM in learning each of the system parameters.

The proposed weight updating rule is designed to first let the weight vectors approach the vicinity of the optimal solution set when it falls outside the coverage of the SOM. The weight vector cluster is then moved to the center of the SOM. The process will continue until the solution set falls within the SOM. Later, the rule will make the weight vectors converge to a more and more compact cluster centering at the optimal solution. We first define a Gaussian distribution function D_j that covers the entire neuron space and centers at its middle:

$$D_j = \exp\left(-\frac{d_{j,j^*}^2}{2\sigma_d^2}\right) \tag{1}$$

where d_{j,j^*} stands for the lateral connection distance between neuron j and j^* and σ_d the standard deviation for D_j. We then define another Gaussian distribution function $F(\underline{W}_j(k))$ as the neighborhood function for the q dimensional $\underline{W}_j$:

$$F(\underline{W}_j(k)) = \exp\left(-\frac{1}{q}\sum_{i=1}^{q}\frac{(w_{j,i}(k) - w_{j^*,i}(k))^2}{2\sigma_{w_i}^2}\right) \tag{2}$$

where $w_{j^*,i}(k)$ stands for the ith element in $\underline{W}_{j^*}(k)$ and σ_{w_i} the standard deviation of the distribution for $w_{j,i}(k)$. Note that $F(\underline{W}_j(k))$ is defined by considering the effects from all the q elements in $\underline{W}_j(k)$. Here, D_j, in the neuron space, is used as a reference distribution for $F(\underline{W}_j(k))$, in the parameter space, to approximate. In other words, we intend to map the magnitude difference of the parameter into the neuron space.[10] To make $F(\underline{W}_j(k))$ approach D_j, an error function $E_j(k)$ is then defined as

$$E_j(k) = \frac{1}{2}\left(D_j - F(\underline{W}_j(k))\right)^2. \tag{3}$$

During the learning, we can find that when $w_{j^*,i}(k)$ is much different from $\tilde{w}_i(k)$, the average of

all $w_{j,i}(k)$, the optimal solution is possibly located far outside the estimated range; contrarily, when $w_{j^*,i}(k)$ is close to $\tilde{w}_i(k)$, the optimal solution is possibly within the estimated range. Based on this observation, we propose a method to speed up the learning. For illustration, we define a Gaussian distribution function $G(w_{j,i}(k))$ as the neighborhood function for each element $w_{j,i}(k)$, the ith element in $\underline{W}_j(k)$ in the kth stage of learning:

$$G(w_{j,i}(k)) = \exp\left(-\frac{(w_{j,i}(k) - \tilde{w}_i(k))^2}{2\sigma_{w_i}^2}\right). \quad (4)$$

The strategy is to vary the mean and variance of $G(w_{j,i}(k))$ by moving its center to where $w_{j^*,i}(k)$ is located and enlarging (reducing) the variance $\sigma_{w_i}^2$ to be ${\sigma_{w_i}^n}^2 = |w_{j^*,i}(k) - \tilde{w}_i(k)|^2$, where $|\cdot|$ stands for the absolute value, as illustrated in Fig. 2. The new neighborhood function $G^n(w_{j,i}^n(k))$ is then

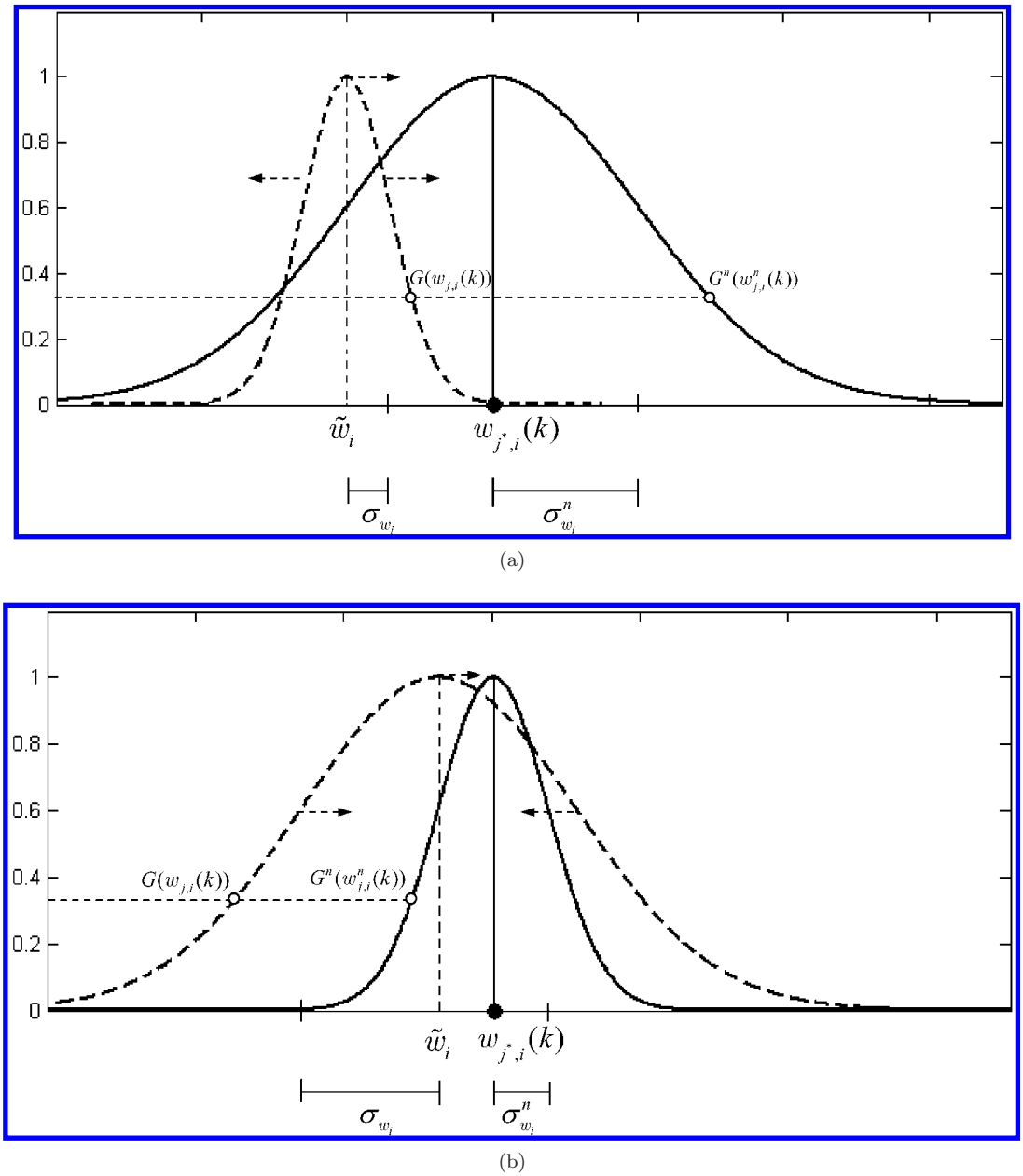


Fig. 2. Center and width adjustment for the neighborhood function $G(w_{j,i}(k))$, when (a) $(\tilde{w}_i - w_{j^*,i}(k))^2 \geq \sigma_{w_i}^2$ and (b) $(\tilde{w}_i - w_{j^*,i}(k))^2 < \sigma_{w_i}^2$.

formulated as

$$G^n(w^n_{j,i}(k)) = \exp\left(-\frac{(w^n_{j,i}(k) - w_{j^*,i}(k))^2}{2\sigma^{n\ 2}_{w_i}}\right)$$
$$= \exp\left(-\frac{(w_{j,i}(k) - \tilde{w}_i(k))^2}{2\sigma^2_{w_i}}\right)$$
$$= G(w_{j,i}(k)) \quad (5)$$

where $w^n_{j,i}(k)$ stands for the new $w_{j,i}(k)$ after the adjustment. As indicated in Fig. 2, $G^n(w^n_{j,i}(k))$ is equal to $G(w_{j,i}(k))$ when $w_{j,i}(k)$ varies to $w^n_{j,i}(k)$. From Eq. (5), during each iteration of learning, $G(w_{j,i}(k))$ is dynamically centered at the location of the winning neuron j^*, with a larger (smaller) width when $\tilde{w}_i(k)$ is much (less) different from $w_{j^*,i}(k)$. It thus covers a more fitting neighborhood region, and leads to a higher learning efficiency. With $G^n(w^n_{j,i}(k))$, the new weight $w^n_{j,i}(k)$ is derived as

$$w^n_{j,i}(k) = \frac{|w_{j^*,i}(k) - \tilde{w}_i(k)|}{\sigma_{w_i}} \cdot (w_{j,i}(k) - \tilde{w}_i(k)) + w_{j^*,i}(k). \quad (6)$$

And, with a desired new weight $w^n_{j,i}(k)$, the learning should also make $\underline{W}_j(k)$ approach $\underline{W}^n_j(k)$, in addition to minimizing the error function $E_j(k)$ in Eq. (3). A new error function $E^n_j(k)$ is thus defined as

$$E^n_j(k) = \frac{1}{2}[(D_j - F(\underline{W}_j(k)))^2 + (\underline{W}_j(k) - \underline{W}^n_j(k))^2]. \quad (7)$$

Based on Eq. (7) and the gradient-descent approach, the weight-updating rule is derived as

$$w_{j,i}(k+1)$$
$$= w_{j,i}(k) - \eta(k)\frac{\partial E^n_j(k)}{\partial w_{j,i}(k)}$$
$$= w_{j,i}(k) - \eta(k)\left[\frac{\partial E_j(k)}{\partial F(\underline{W}_j(k))} \cdot \frac{\partial F(\underline{W}_j(k))}{\partial w_{j,i}(k)} + (w_{j,i}(k) - w^n_{j,i}(k))\right]$$
$$= w_{j,i}(k) - \eta(k)\left[\frac{(w_{j,i}(k) - w_{j^*,i}(k))}{q \cdot \sigma^2_{w_i}} \cdot F(\underline{W}_j(k)) \cdot (D_j - F(\underline{W}_j(k))) + (w_{j,i}(k) - w^n_{j,i}(k))\right] \quad (8)$$

where $\eta(k)$ stands for the learning rate in the kth stage of learning. In the initial stage of the learning, $w_{j,i}(k)$ and $w_{j^*,i}(k)$ may be much different from each other, and the learning process can be speeded up with a larger $\eta(k)$. Later on, when they almost coincide, the learning rate may be decreased gradually. A function for $\eta(k)$ that satisfies the demand is formulated as

$$\eta(k) = \eta_1 \cdot e^{-k/\tau} + \eta_0 \quad (9)$$

where η_0 and η_1 are constants smaller than 1, and τ time constant. Of course, other types of functions can also be used. Together, the weight updating rule described in Eq. (8) and the learning rate in Eq. (9) will force the minimization of the difference between the weight vector of the winning neuron and those corresponding to every neuron in each learning cycle. The learning will eventually converge.

4. Applications

To demonstrate its capability, the SOMS is applied to both function optimization and dynamic trajectory prediction. Based on the SOMS, we first develop learning schemes corresponding to each of the applications. Simulations are then executed for performance evaluation. The results are especially compared with those of the genetic algorithm (GA) because of their similar searching abilities. The GA is basically a search algorithm based on the mechanics of natural selection and natural genetics.[6] It employs multiple concurrent search points called chromosomes and evaluates the fitness of each chromosome. The search procedure uses random choice as a tool to guide a highly explorative search through a coding of a parameter space. Both the SOM and GA have the merit of parallel processing. Furthermore, both of their searches are through the guidance of the evaluation function, while the SOM in our design adopts a somewhat organized search and the GA has a somewhat random approach. This implies that the SOM may be more suitable for applications with certain knowledge, especially when the distribution of the possible solutions is not utterly random. On the contrary, for applications with no *a priori* knowledge available, the GA may yield better performance.

4.1. *Function optimization*

For a function optimization problem, the goal may be to maximize (minimize) an objective function $O(\cdot)$. Let $O(\underline{W}_j(k))$ be the function value for the weight

vector $\underline{W}_j(k)$, which represents a possible solution. During the learning process, an initial reference value $P_r(k)$ is chosen first and later set to be the current maximal (minimal) value of the objective function for maximization (minimization). With the repeated approximation of the objective function to $P_r(k)$, the SOMS will approach the optimal maximal (minimal) value gradually. The learning algorithm for function optimization is organized as follows.

Algorithm for function optimization based on the SOMS: Maximize (minimize) an objective function using the SOMS.

Step 1: Set the stage of learning $k = 0$. Choose an initial reference value $P_r(0)$. Estimate the ranges of the possible parameter space and randomly store the possible parameters $\underline{W}_j(0)$ into the neurons, where $j = 1, \ldots, N \times N$, $N \times N$ the total number of neurons in the 2D ($N \times N$) space.

Step 2: Compute $O(\underline{W}_j(k))$ for all $\underline{W}_j(k)$.

Step 3: Among the neurons, find the one with the largest (smallest) value as the winning neuron j^* for the maximization (minimization) problem.

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors according to the weight updating rule described in Sec. 3.

Step 5: Check whether the difference between $O(\underline{W}_{j^*}(k))$ and $P_r(k)$ is smaller than a preset threshold value. If it is not, let $k = k + 1$, set $P_r(k)$ to be the current maximal (minimal) value of the objective function for the maximization (minimization) problem, and go to Step 2; otherwise, the learning process is completed and output the optimal value.

Two standard test functions are used to demonstrate the proposed algorithm, a 2-D Griewant function

$$f(x_1, x_2) = 1 + \frac{1}{4000}\left[(x_1 - 100)^2 + (x_2 - 100)^2\right] - \cos(x_1 - 100) \cdot \cos\left(\frac{x_2 - 100}{\sqrt{2}}\right) \quad (10)$$

and a 30-D Rosenbrock function

$$f(\underline{x}) = \sum_{i=1}^{30}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]. \quad (11)$$

These two test functions have also been used in Ref. 21. The optimization here is to minimize these two functions. Their global minimal values are known in advance: for the Griewant function, it is 0 when $(x_1, x_2) = (100, 100)$; for the Rosenbrock function, it is also 0 when all x_i are equal to 1. The SOM is chosen to be of 5×5 neurons and the learning rate as

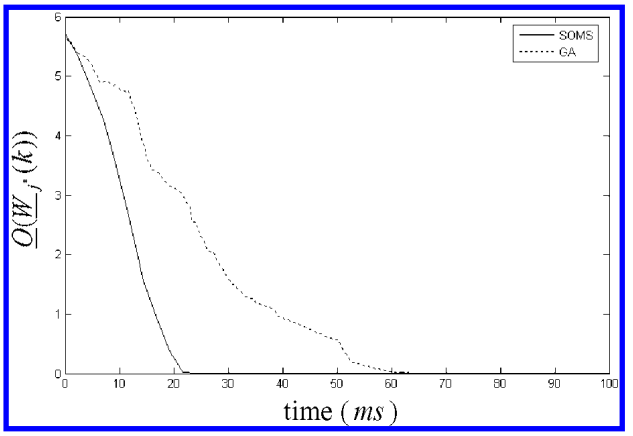
$$\eta(k) = 0.7 \cdot e^{-k/50} + 0.2. \quad (12)$$

For comparison, we also use the GA for function minimization, which is with a population size of 25 to match that of the SOM, and the crossover and mutation probability of 0.6 and 0.0333, respectively.

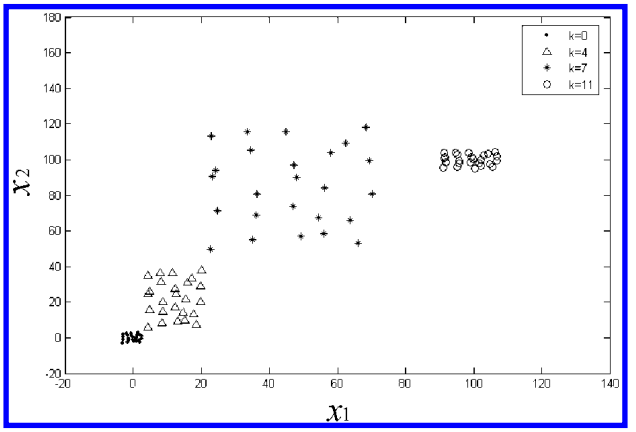
We start with the learning for the 2-D Griewant function. The initial $\underline{W}_j(0)$ for the SOMS was randomly chosen within the ranges of $(-3, 3) \times (-3, 3)$, i.e., the optimal solution was outside of the estimated region. Figure 3 shows the simulation results. In Fig. 3(a), both SOMS and GA found the optimal minimal value successfully, but the SOMS converged faster. Figures 3(b) and (c) show the weight vector movement ($k = 0 \sim 11$) for the SOMS and GA, respectively. From the figures, we observed that the search in the SOMS was basically in grouping and more directional; by contrast, that of the GA was in a random manner. This indicates that the SOMS was more effective for this 2-D Griewant function minimization because the distribution of the possible solutions might not be utterly random. In the minimization of the 30-D Rosenbrock function, we simulated the case where the optimal solution was within the estimated region. For its complexity, the size of the SOM was enlarged to be of 25×25 neurons, while that of the GA was also enlarged accordingly. The learning rate for the SOMS and the crossover and mutation probabilities for the GA were set to be the same. Each $w_{j,i}(0)$ of the initial $\underline{W}_j(0)$ was randomly chosen within the range of $(-5, 5)$. In addition to the SOMS and GA, the SOMO proposed in Ref. 21 was also used for the minimization, with its parameters adjusted via a trial-and-error process to yield salient performance. Figure 4 shows the simulation results. In Fig. 4, all SOMS, GA, and SOMO found the optimal minimal value successfully, while the SOMS converged faster. It indicates that the SOMS was also more effective for the 30-D Rosenbrock function minimization.

4.2. *Dynamic trajectory prediction*

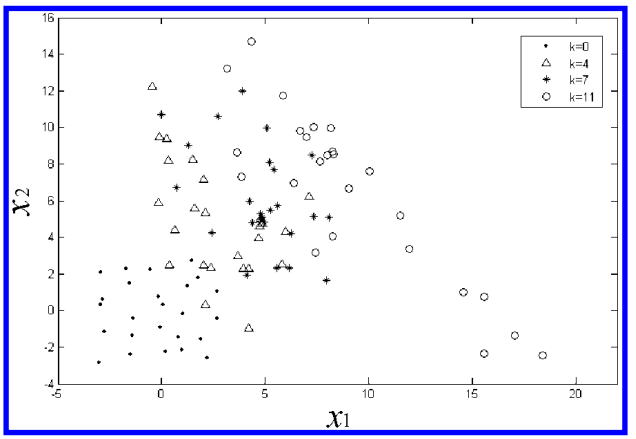
For a dynamic trajectory prediction problem, the goal may be to estimate the launching position and velocity of a moving object using the measured



(a)



(b)



(c)

Fig. 3. Minimization of the 2-D Griewant function using the SOMS and GA with the optimal solution outside of the estimated region: (a) minimal function values $O(\underline{W}_{j^*}(k))$ during the learning process, (b) weight vector movement in the SOM, and (c) weight vector movement in the GA.

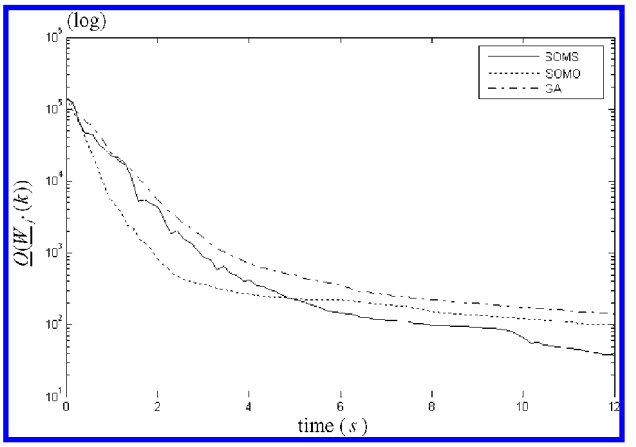


Fig. 4. Minimal function values $O(\underline{W}_{j^*}(k))$ during the learning process for the minimization of the 30-D Rosenbrock function using the SOMS, GA, and SOMO.

data. Through a learning process, the SOMS may determine the most probable initial state through repeatedly comparing the measured data with the predicted trajectories derived from the possible initial states stored in the neurons of the SOM. We consider the SOMS very suitable for this application because the relationship between the initial state and its resultant trajectory is not entirely random. We can thus distribute the initial states into the SOM in an organized fashion, and make it as a guided search.

In this application, the nonlinear dynamic equation describing the trajectory of the moving object and the measurement equation are first formulated as

$$\underline{x}(k+1) = f_k(\underline{x}(k)) + \underline{\xi}_k \tag{13}$$

$$\underline{p}(k) = g_k(\underline{x}(k)) + \underline{\zeta}_k \tag{14}$$

where f_k and g_k are the vector-value function defined in R^q and R^l (q and l the dimension), respectively, and their first-order partial derivatives with respect to all the elements of $\underline{x}(k)$ continuous. $\underline{\xi}_k$ and $\underline{\zeta}_k$ are the zero-mean Gaussian white noise sequence in R^q and R^l, respectively, with

$$E[\underline{\xi}_k] = 0 \tag{15}$$

$$E[\underline{\xi}_j \underline{\xi}_k^T] = Q\delta_{jk} \tag{16}$$

$$E[\underline{\zeta}_k] = 0 \tag{17}$$

$$E[\underline{\zeta}_j \underline{\zeta}_k^T] = R\delta_{jk} \tag{18}$$

$$E[\underline{\xi}_j \underline{\zeta}_k^T] = 0 \tag{19}$$

where $E[\cdot]$ stands for the expectation function, Q and R the covariance matrix of the input noise and

output noise, respectively, and δ_{jk} the Dirac delta function. Q and R are expected to be uncertain and varying in noisy, unknown environments, and their estimated values possibly imprecise, even incorrect. Being unaware of the statistical properties of the dynamic model, the SOMS is utilized to find the optimal initial state via learning. The learning algorithm for dynamic trajectory prediction is organized as follows.

Algorithm for dynamic trajectory prediction based on the SOMS: Predict an optimal initial state for the trajectory of a moving object using the measured position data.

Step 1: Set the stage of learning $k = 0$. Estimate the ranges of the possible launching position and velocity of the moving object, and randomly store the possible initial states $\underline{W}_j(0)$ into the neurons, where $j = 1, \ldots, N \times N$, $N \times N$ the total number of neurons in the 2D ($N \times N$) space.

Step 2: Send $\underline{W}_j(k)$ into the dynamic model, described in Eqs. (13)–(14), to compute $\underline{P_d}_j(k)$, the predicted position.

Step 3: For each neuron j, compute its output $O_j(k)$ as the Euclidean distance between the measured position data $\underline{P_m}(k)$ and $\underline{P_d}_j(k)$:

$$O_j(k) = \sum_{i=0}^{k} \|\underline{P_m}(i) - \underline{P_d}_j(i)\|. \tag{20}$$

Find the winning neuron j^* with the minimum $O_{j^*}(k)$:

$$\begin{aligned} O_{j^*}(k) &= \sum_{i=0}^{k} \|\underline{P_m}(i) - \underline{P_d}_{j^*}(i)\| \\ &= \min_j \sum_{i=0}^{k} \|\underline{P_m}(i) - \underline{P_d}_j(i)\|. \end{aligned} \tag{21}$$

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors.

Step 5: Check whether $O_{j^*}(k)$ is smaller than a prespecified value ϵ:

$$O_{j^*}(k) < \epsilon. \tag{22}$$

If Eq. (22) does not hold, let $k = k + 1$ and go to Step 2; otherwise, the prediction process is completed and output the predicted optimal initial state to the dynamic model to derive the object trajectory.

Note that the value of ϵ is empirical according to the demanded resolution in learning, and we chose it very close to zero. In addition, during each stage of learning, we perform a number of learning to increase the SOM learning speed. This number of learning is set to be a large number in the initial stage of the learning process, such that the SOMS may converge faster at the price of more oscillations, and decreased gradually to achieve smooth learning in later stages of learning.

To demonstrate the effectiveness of the proposed SOMS and weight updating rule, we performed a series of simulations for dynamic trajectory prediction based on using the SOMS, the SOMS without the proposed center and width adjustment on the neighborhood function (named as SOMSO), and GA. The trajectory to predict in the simulations was designed to emulate that of a missile. Its governing equations of motion in the 3D Cartesian coordinate system are described as

$$\ddot{x} = \frac{-g_m x}{(x^2 + y^2 + z^2)^{3/2}} + 2\omega\dot{y} + \omega^2 x + \xi_x \tag{23}$$

$$\ddot{y} = \frac{-g_m y}{(x^2 + y^2 + z^2)^{3/2}} + 2\omega\dot{x} + \omega^2 y + \xi_y \tag{24}$$

$$\ddot{z} = \frac{-g_m z}{(x^2 + y^2 + z^2)^{3/2}} + \xi_z \tag{25}$$

where g_m and ω stand for the gravitational constant and the rotative velocity of the earth, respectively, and set to be $g_m = 3.986 \times 10^5 km^3/s^2$ and $\omega = 7.2722 \times 10^{-5} rad/s$. (ξ_x, ξ_y, ξ_z) are assumed to be continuous-time uncorrelated zero-mean Gaussian white noise processes. Referring to Eq. (13) and letting $\underline{X} = (x, y, z, \dot{x}, \dot{y}, \dot{z})^T = (x_1, x_2, x_3, x_4, x_5, x_6)^T$, we can obtain the discretized dynamic equation as

$$\underline{X}(k+1) = f(\underline{X}(k)) + \underline{\xi}_k \tag{26}$$

where

$$f(\underline{X}(k)) = \begin{bmatrix} x_1(k) + tx_4(k) \\ x_2(k) + tx_5(k) \\ x_3(k) + tx_6(k) \\ x_4(k) - tg_m x_1(k)/(x_1(k)^2 + x_2(k)^2 \\ \quad +x_3(k)^2)^{3/2} + 2t\omega x_5(k) + t\omega^2 x_1(k) \\ x_5(k) - tg_m x_2(k)/(x_1(k)^2 + x_2(k)^2 \\ \quad +x_3(k)^2)^{3/2} + 2t\omega x_4(k) + t\omega^2 x_2(k) \\ x_6(k) - tg_m x_3(k)/(x_1(k)^2 \\ \quad +x_2(k)^2 + x_3(k)^2)^{3/2} \end{bmatrix} \tag{27}$$

and

$$\underline{\xi}_k = [0\ 0\ 0\ \xi_{x_4}\ \xi_{x_5}\ \xi_{x_6}]^T \tag{28}$$

with t the sampling time. $(\xi_{x_4}, \xi_{x_5}, \xi_{x_6})$ are assumed to be uncorrelated zero-mean Gaussian white noise sequences with a constant variance $\sigma_f^2 = (0.1m/s^2)^2$. And, referring to Eq. (14), the measurement equation is formulated as

$$\underline{P}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \underline{X}(k) + \underline{\zeta}_k \tag{29}$$

where $\underline{\zeta}_k = (\zeta_{x_1}, \zeta_{x_2}, \zeta_{x_3})$ are the measurement noise sequences with a zero mean and constant variance $\sigma_m^2 = (15m)^2$. The ranges of the possible initial states $\underline{W}_j(0)$ were estimated to be

$$\begin{aligned} &68.6 \times 10^5 m \le x_1(0) \le 68.8 \times 10^5 m \\ &2.7 \times 10^5 m \le x_2(0) \le 2.8 \times 10^5 m \\ &4.8 \times 10^5 m \le x_3(0) \le 4.9 \times 10^5 m \\ &110 m/s \le x_4(0) \le 150 m/s \\ &810 m/s \le x_5(0) \le 850 m/s \\ &1360 m/s \le x_6(0) \le 1380 m/s. \end{aligned} \tag{30}$$

Within the ranges described in Eq. (30), the possible launching positions and velocities of the missile were selected and stored into the 729 (27×27) neurons of the 2D SOM. In addition, the learning rate for the SOMS was chosen to be

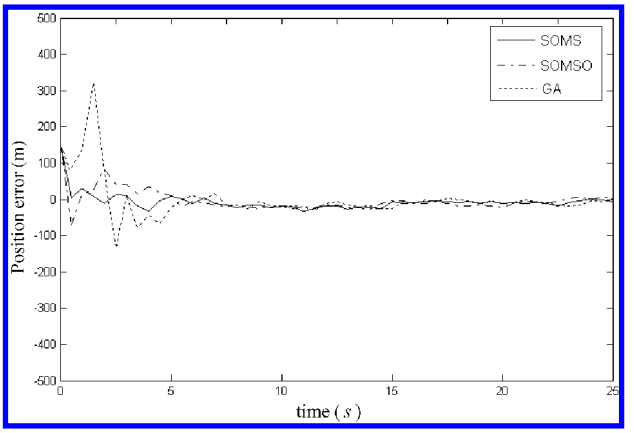
$$\eta(k) = 0.8 \cdot e^{-k/50} + 0.2 \tag{31}$$

The sampling time t was $0.5s$. For the GA, the population size was selected to be 729 to match with the SOM, and the crossover and mutation probability 0.6 and 0.0333, respectively.

We first applied the SOMS, SOMSO, and GA for trajectory prediction with a good estimate of the initial state. The ideal initial state of the missile was assumed to be $(68.7 \times 10^5 m, 2.7 \times 10^5 m, 4.8 \times 10^5 m, 130 m/s, 820 m/s, 1370 m/s)$, which was within the estimated range. The variance of the measurement noise was set to be $(15m)^2$. Figure 5 shows the simulation results. All SOMS, SOMSO, and GA predicted the initial state quite well and thus resulted in very small estimated errors, except for the initial stage of the prediction, as shown in Fig. 5(a) (only the position error in the X-direction (x_1) is shown for illustration). Figure 5(b) shows how the neighborhood function $F(\underline{W}_j(k))$, described in Eq. (3), varied during the SOM learning process. In Fig. 5(b), from a random distribution in the beginning of the learning, $F(\underline{W}_j(k))$ gradually approximated the Gaussian distribution along with the stage of learning.

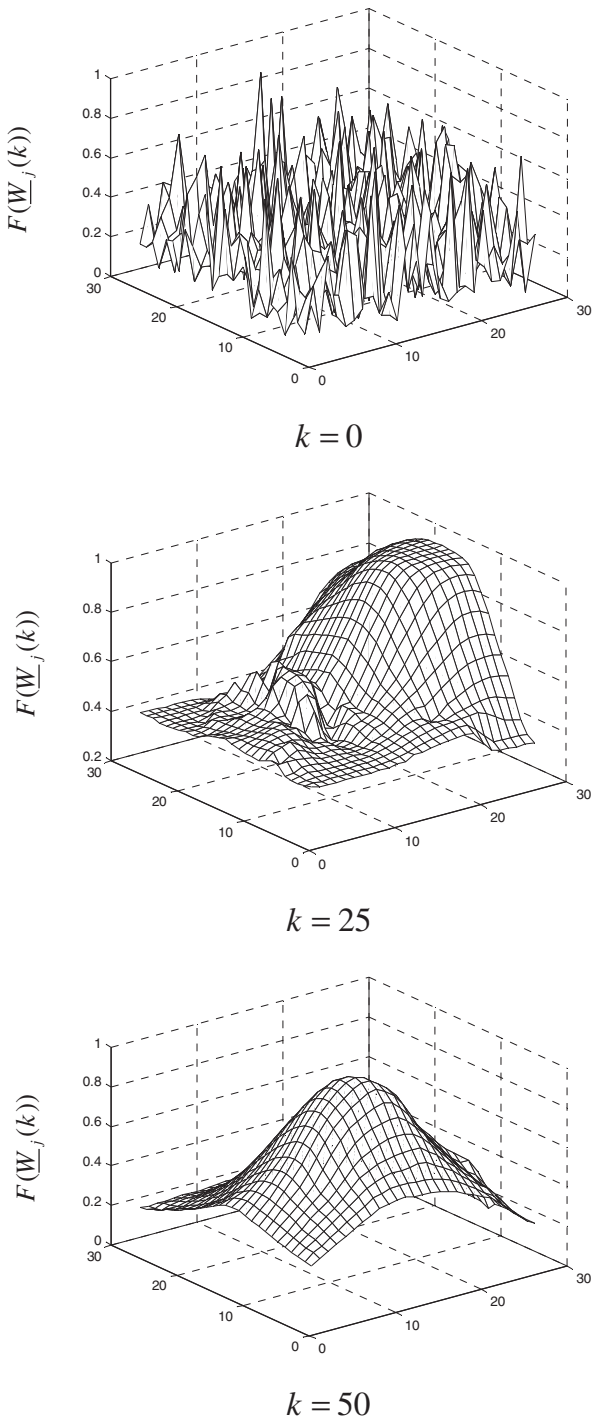
In the second set of simulations, we investigated their performances for the condition of a bad estimate of the initial state. In this simulation, the ideal initial state was assumed to be $(64 \times 10^5 m, 4.8 \times 10^5 m, 2.4 \times 10^5 m, 215 m/s, 2130 m/s, 1030 m/s)$, which was outside the estimated range. The variance of the measurement noise was enlarged to be $(30m)^2$. From the simulation results shown in Fig. 6, the influence of the bad estimate on the SOMS and SOMSO was mostly at the initial stage of the prediction. After the transient, the SOMS and SOMSO still managed to find the optimal initial state. Meanwhile, we also observed that the SOMS converged faster than the SOMSO. As for the GA, it converged very slowly as the optimal initial state did not fall within the estimated range. We thus conclude that the SOMS performed better than the GA for this dynamic trajectory prediction application, and the proposed dynamic weight updating rule was effective.

For further investigation, in the third set of simulations, we compared the performance of the SOMS with that of the Kalman filtering, a famous approach widely used in predicting the movements of the satellites, airplanes, ships, etc. As the dynamic model of



(a)

Fig. 5. Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a good estimate of the initial state: (a) the estimated position error in the X-direction and (b) the variation of the neighborhood function $F(\underline{W}_j(k))$ during the SOMS learning process.



(b)

Fig. 5. (*Continued*)

the moving object was available, the Kalman filter yielded satisfactory performance when the statistics of the environmental noises and initial conditions were well approximated. However, with bad

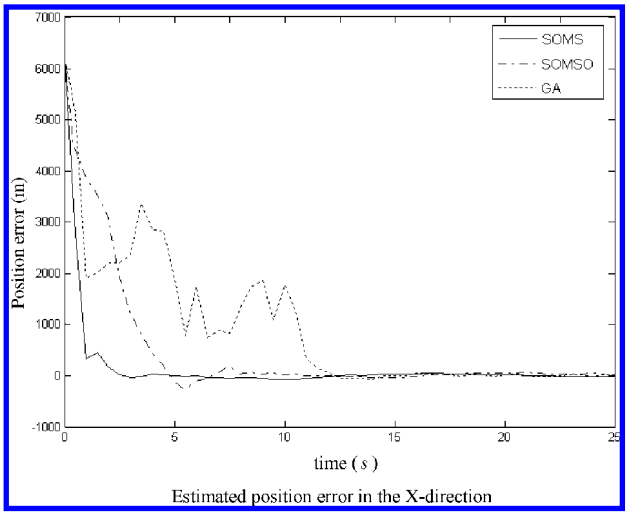


Fig. 6. Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a bad estimate of the initial state.

estimates of the noise distribution and initial conditions, its performance was much degraded. By contrast, their influence on the SOMS was mostly as the transient in the early stage of the prediction. The results show that the SOMS is more robust to uncertainty, while the Kalman filter may not be that suitable for unknown, noisy environments.

5. Conclusion

In this paper we have proposed an SOM-based algorithm for optimization problems, which can be used for both static and dynamic functions in real time. To achieve high learning efficiency for system parameters in different working ranges, we have also proposed a new SOM weight updating rule. The applications of the proposed SOMS on both function optimization problems and dynamic trajectory predictions have clearly proven its effectiveness. To further exploit its search ability, in future work, we will apply the SOMS for system identification and control problems. Another worthwhile future work will be to extend the proposed SOMS for systems involving multiple targets.

Acknowledgment

This work was supported in part by the National Science Council under grant NSC 94-2218-E-009-006, and also Department of Industrial Technology under grant 94-EC-17-A-02-S1-032.

References

1. A. P. Azcarraga, T. N. Yap Jr., J. Tan and T. S. Chua, Evaluating keyword selection methods for WEBSOM text archives, *IEEE Trans. on Knowledge and Data Engineering* **16**(3) (2004) 380–383.
2. G. A. Barreto and A. F. R. Araujo, Identification and control of dynamical systems using the Self-Organizing Map, *IEEE Trans. on Neural Networks* **15**(5) (2004) 1244–1259.
3. A. G. Barto, Reinforcement learning and adaptive critic methods, *Handbook of Intelligent Control*, White and Sofge (eds.), Van Nostrand-Reinhold, New York (1992) 469–491.
4. G. A. Carpenter and S. Grossberg, The ART of adaptive pattern recognition by a self-organizing neural network, *IEEE Computer* **21**(3) (1988) 77–88.
5. Y. Y. Chen and K. Y. Young, An intelligent radar predictor for hight-speed moving-target tracking, *International J. Fuzzy Systems* **6**(2) (2004) 90–99.
6. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison Wesley, New York, 1989).
7. M. Hagenbuchner and A. C. Tsoi, A supervised Self-Organizing Map for structures, *IEEE Conference on Neural Networks* (2004) 1923–1928.
8. S. Haykin, *Neural Networks: A Comprehensive Foundation* (Macmillan, New York, 1994).
9. H.-D. Jin, K.-S. Leung, M.-L. Wong and Z.-B. Xu, An efficient Self-Organization Map designed by genetic algorithms for the traveling salesman problem, *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics* **33**(6) (2003) 877–888.
10. J. A. Kangas, T. K. Kohonen and J. T. Laaksonen, Variants of Self-Organizing Maps, *IEEE Trans. on Neural Networks* **1**(1) (1990) 93–99.
11. S. Kaski, T. Honkela, K. Lagus and T. K. Kohonen, WEBSOM — Self-Organizing Maps of document collections, *Neurocomputing* **21** (1998) 101–117.
12. K. J. Kim and S. B. Cho, Fusion of structure adaptive Self-Organizing Maps using fuzzy integral, *IEEE Conference on Neural Networks* (2003) 28–33.
13. T. Kohonen, *Self-Organizing Map* (Springer-Verlag, Berlin Heidelberg, 1995).
14. J. Laaksonen, M. Koskela and E. Oja, PicSOM — Self-organizing image retrieval with MPEG-7 content descriptors, *IEEE Trans. on Neural Networks* **13**(4) (2002) 841–853.
15. M. Milano, P. Koumoutsakos and J. Schmidhuber, Self-organizing nets for optimization, *IEEE Trans. on Neural Networks* **15**(3) (2004) 758–765.
16. K. Obermayer and T. J. Sejnowski (eds.), *Self-Organizing Map formation: Foundation of neural computation* (MIT Press, Cambridge, 2001).
17. J. C. Principe, L. Wang and M. A. Motter, Local dynamic modeling with Self-Organizing Maps and applications to nonlinear system identification and control, *Proceedings of the IEEE* **86**(11) (1998) 2240–2258.
18. D. Sbarbaro and D. Bassi, A nonlinear controller based on Self-Organizing Maps, *IEEE Int. Conference on Systems, Man and Cybernetics* (1995) 1774–1777.
19. H. Shah-Hosseini and R. Safabakhsh, TASOM: A new adaptive Self-Organization Map, *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics* **33**(2) (2003) 271–282.
20. M. C. Su and H. T. Chang, Fast self-organizing feature map algorithm, *IEEE Trans. on Neural Networks* **11**(3) (2000) 721–733.
21. M. C. Su, Y. X. Zhao and J. Lee, SOM-based optimization, *IEEE Int. Conference on Neural Networks* (2004) 781–786.
22. J. A. Walter and K. I. Schulten, Implementation of self-organizing neural networks for visuo-motor control of an industrial robot, *IEEE Trans. on Neural Networks* **4**(1) (1993) 86–96.
23. S. Wu and T. W. S. Chow, PRSOM: A new visualization method by hybridizing multidimensional scaling and Self-Organizing Map, *IEEE Trans. on Neural Networks* **16**(6) (2005) 1362–1380.
24. P. Xu, C. H. Chang and A. Paplinski, Self-organizing topological tree for online vector quantization and data clustering, *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics* **35**(3) (2005) 515–526.
25. H. Yin, ViSOM — A novel method for multivariate data projection and structure visualization, *IEEE Trans. on Neural Networks* **13**(1) (2002) 237–243.

This article has been cited by:

1. GEMA BELLO-ORGAZ, HÉCTOR D. MENÉNDEZ, DAVID CAMACHO. 2012. ADAPTIVE K-MEANS ALGORITHM FOR OVERLAPPED GRAPH CLUSTERING. *International Journal of Neural Systems* **22**:05. . [Abstract] [References] [PDF] [PDF Plus]
2. M. LOURDES BORRAJO, BRUNO BARUQUE, EMILIO CORCHADO, JAVIER BAJO, JUAN M. CORCHADO. 2011. HYBRID NEURAL INTELLIGENT SYSTEM TO PREDICT BUSINESS FAILURE IN SMALL-TO-MEDIUM-SIZE ENTERPRISES. *International Journal of Neural Systems* **21**:04, 277-296. [Abstract] [References] [PDF] [PDF Plus]
3. EZEQUIEL LÓPEZ-RUBIO, RAFAEL MARCOS LUQUE-BAENA, ENRIQUE DOMÍNGUEZ. 2011. FOREGROUND DETECTION IN VIDEO SEQUENCES WITH PROBABILISTIC SELF-ORGANIZING MAPS. *International Journal of Neural Systems* **21**:03, 225-246. [Abstract] [References] [PDF] [PDF Plus]
4. ALEXANDER N. GORBAN, ANDREI ZINOVYEV. 2010. PRINCIPAL MANIFOLDS AND GRAPHS IN PRACTICE: FROM MOLECULAR BIOLOGY TO DYNAMICAL SYSTEMS. *International Journal of Neural Systems* **20**:03, 219-232. [Abstract] [References] [PDF] [PDF Plus]
5. JUNPING ZHANG, UWE KRUGER, XIAODAN WANG, DEWANG CHEN. 2010. A RIEMANNIAN DISTANCE APPROACH FOR CONSTRUCTING PRINCIPAL CURVES. *International Journal of Neural Systems* **20**:03, 209-218. [Abstract] [References] [PDF] [PDF Plus]
6. Hojjat Adeli, Ashif Panakkat. 2009. A probabilistic neural network for earthquake magnitude prediction. *Neural Networks* **22**:7, 1018-1024. [CrossRef]
7. GIULIANO GROSSI. 2009. ADAPTIVENESS IN MONOTONE PSEUDO-BOOLEAN OPTIMIZATION AND STOCHASTIC NEURAL COMPUTATION. *International Journal of Neural Systems* **19**:04, 241-252. [Abstract] [References] [PDF] [PDF Plus]