# An SOM-Based Search Algorithm for Dynamic Systems

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

## Abstract

**—The self-organizing map (SOM), as a kind of unsupervised neural network, has been applied for both static data management and dynamic data analysis. To further exploit its ability in search, in this paper, we propose an SOM-based search algorithm (SOMS) for dynamic systems, in which the SOM is employed as a searching mechanism. And, a new SOM weight updating rule is proposed to enhance the learning efficiency, which may dynamically adjust the neighborhood function for the SOM in learning system parameters. For demonstration, the proposed learning scheme is applied for continuous optimization problem and also dynamic trajectory prediction, and its effectiveness evaluated via the simulations based on using the SOM and GA, due to their resemblance in learning and searching.**

**Key words:** Self-Organizing Map, Search Algorithm, Dynamic System, 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 [2], [7]. In its many applications, the SOM has been used for both static data management and dynamic data analysis, such as data mining, knowledge discovering, clustering, visualization, speaker recognition, mobile communication, robot control, identification and control of dynamic systems, local dynamic modeling, nonlinear control, and moving object tracking [1], [5], [7], [10], [11], [12], [14]. However, from our survey, its ability in search has not been well exploited yet [3], [4], [6], [8], [9], [13]. It thus motivates us to propose an SOM-based search algorithm (SOMS) for dynamic systems.

In its use as a search mechanism, the SOM has been applied mainly for optimization problem [6], [8], [13]. Michele et al. proposed a learnig algorithm for optimization based on the Kohonen SOM evolution strategy (KSOM-ES) [8]. In this KSOM-ES algorithm, the adaptive grids are used to identify and exploit search space regions that maximize the probability of generating points closer to the optima. Su et al. propose an SOM-based optimization algorithm (SOMO). 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 systems. In search of dynamic system, the goal may be to look for a set of optimal parameters that lead to the desired performance of the dynamic system from limited incoming dynamic 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.

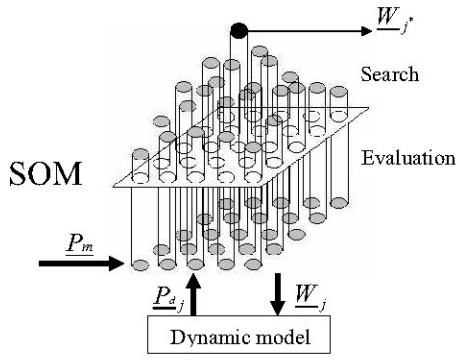



Fig. 1. The structure and operation of the SOM in the proposed SOMS.

In developing the SOMS, we will first examine the SOM in its learning strategy and search ability. Meanwhile, as the dynamic system is tackled, the SOM learning may involve both the system parameters and their derivatives, which operate in quite different ranges. To achieve high learning efficiency under such a wide parameter variation, 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.

## 2. PROPOSED SOM-BASED SEARCH ALGORITHM

Figure 1 shows the structure and operation of the SOM in the SOMS. The SOM performs two main operations: evaluation and search. In Figure 1, each neuron $j$ in the SOM contains a vector of a possible solution set $\underline{W}_j$ (the weight vector). Each time new measured dynamic 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 dynamic 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. When the weight vector $\underline{W}_{j^*}$ of this winning neuron $j^*$ differs from the average of all weights $\underline{\tilde{W}}(k)$, the weight vectors of $j^*$ and its neighbors will be updated to exploit a new range. When

$\underline{W}_{j^*}$ is the same as $\tilde{W}$, the weight vectors will be updated so as to make them form more and more compact clusters centering at neuron $j^*$. Under successful learning, the SOM will finally converge to an optimal solution set. Note that, this SOMS can also be applied for continuous optimization problems, with the dynamic model replaced by the objective function for the given optimization problem and the input by the reference data.

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 in an individual basis, instead of using a 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 make the distance between neuron $j$ and $j^*$ correspond to that of their weight vectors, $\underline{W}_j$ and $\underline{W}_{j^*}$. We first define a Gaussian distribution function $G(w_{j,i}(k))$ as the neighborhood function for each element $w_{j,i}(k)$, the $i$th element in $\underline{W}_j(k)$:

$$G(w_{j,i}(k))=\exp(-\frac{(w_{j,i}(k)-\tilde{w}_i(k))^2}{2\sigma^2_{w_i}}) \quad (1)$$

where $\tilde{w}_i(k)$ stands for the average of all $w_{j,i}(k)$ and $\sigma_{w_i}$ the standard deviation of the neighborhood function $G(w_{j,i}(k))$. We then define another two Gaussian neighborhood functions, $D_j(k)$ in the neuron space and $F(\underline{W}_j(k))$ in the weight vector space, in the $k$th stage of learning as

$$D_j(k) = \exp(-\frac{d^2_{j,j^*}(k)}{2\sigma^2_d}) \quad (2)$$

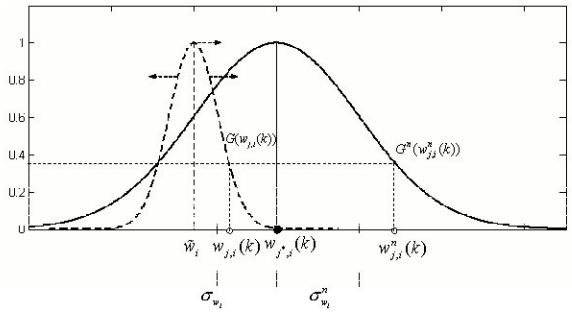
$$F(\underline{W}_j(k)) = \exp\left(\frac{-1}{q}\sum_{i=1}^{q}\frac{(w_{j,i}(k)-w_{j^*,i}(k))^2}{2\sigma^2_{w_i}}\right) \quad (3)$$

where $d_{j,j^*}(k)$ stands for the lateral connection distance between neuron $j$ and $j^*$, $w_{j^*,i}(k)$ the $i$th element in $\underline{W}_{j^*}(k)$, $q$ the dimension of $\underline{W}_j$, and $\sigma_d$ the standard deviation of the neighborhood function $D_j(k)$. An error function $E_j(k)$ is then defined as

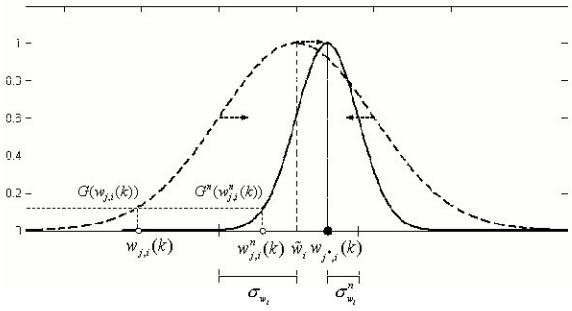
$$E_j(k) = \frac{1}{2}(D_j(k) - F(\underline{W}_j(k)))^2 \quad (4)$$

During the learning, we can find that when $w_{j^*,i}(k)$ is much different from $\tilde{w}_i(k)$, the optimal solution is possibly located much outside of 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. To speed up the learning, we thus propose varying 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 $\sigma^2_{w_i}$ to be ${\sigma^n_{w_i}}^2 = |w_{j^*,i}(k)-\tilde{w}_i(k)|^2$, where $|\cdot|$ stands for the absolute value, as illustrated in Figure 2. The new $G^n(w_{j,i}(k))$ is then formulated as

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




(a) $\left(\tilde{w}_i - w_{j^*,i}(k)\right)^2 \geq \sigma^2_{w_i}$




(b) $\left(\tilde{w}_i - w_{j^*,i}(k)\right)^2 < \sigma^2_{w_i}$

Fig. 2. Variations of the weights in the learning process: (a) when $\left(\tilde{w}_i - w_{j^*,i}(k)\right)^2 \geq \sigma^2_{w_i}$, (b) when $\left(\tilde{w}_i - w_{j^*,i}(k)\right)^2 < \sigma^2_{w_i}$

where $w^n_{j,i}(k)$ stands for the new $w_{j,i}(k)$ after the adjustment. Consequently, 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 this new $G^n(w^n_{j,i}(k))$, 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.(4). A new error function $E^n_j(k)$ is thus defined as

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

Based on the gradient-descent approach, the weight-updating rule is derived as

$$\begin{aligned} w_{j,i}(k+1) &= w_{j,i}(k)-\eta(k)\frac{\partial E^n_j(k)}{\partial w_{j,i}(k)} \\ &= w_{j,i}(k)-\eta(k)[\frac{\partial E_j(k)}{\partial F(\underline{W}_j(k))}\cdot\frac{\partial F(\underline{W}_j(k))}{\partial w_{j,i}(k)}+ \\ &\quad (w_{j,i}(k)-w^n_{j,i}(k))] \\ &= w_{j,i}(k)-\eta(k)[\frac{(w_{j,i}(k)-w_{j^*,i}(k))}{q\cdot\sigma_{w_i}}\cdot F(\underline{W}_j(k))\cdot(D_j(k) \\ &\quad -F(\underline{W}_j(k)))+(w_{j,i}(k)-w^n_{j,i}(k))] \end{aligned} \quad (8)$$

where $\eta(k)$ stands for the learning rate in the $k$th 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

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

where $\eta_0$ and $\eta_1$ are constants smaller than 1, and $\tau$ 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 converge eventually.

# 3.TRAJECTORY PREDICTION APPLICATION

For a dynamic trajectory prediction problem, the goal may be to estimate the launching position and velocity of a moving object using the measured data. Through a learning process, the SOMS may determine a 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 utterly 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

$$\underline{x}(k+1) = f_k(\underline{x}(k))+\underline{\xi}_k \tag{10}$$

$$\underline{p}(k) = g_k(\underline{x}(k))+\underline{\zeta}_k \tag{11}$$

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 $R^l$.

**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 $j=1,\ldots,m\times n$, $m\times n$ the total number of neurons in the 2D ($m\times n$) space.

**Step 2:** Send $\underline{W}_j(k)$ into the dynamic model, described in Eqs.(10)-(11), to compute $\underline{P_d}_j(k)$.

**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 $\underline{P_d}_j(k)$:

$$O_j(k)=\sum_{i=0}^{k}\|\underline{P_m}(i)-\underline{P_d}_j(i)\|. \tag{12}$$

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

$$O_{j^*}(k)=\sum_{i=0}^{k}\|\underline{P_m}(i)-\underline{P_d}_{j^*}(i)\|=\min_j \sum_{i=0}^{k}\|\underline{P_m}(i)-\underline{P_d}_j(i)\|. \tag{13}$$

**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 pre-specified value $\epsilon$:

$$O_{j^*}(k)<\epsilon. \tag{14}$$

If Eq.(14) 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 $\epsilon$ 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.

# 4. SIMULATION

In the simulations, the trajectory to predict was for a moving object that emulated a missile. Its governing equations of motion in the 3D Cartesian coordinate system are described as

$$\ddot{x} = \frac{-g_m x}{(x^2+y^2+z^2)^{3/2}}+2\omega\dot{y}+\omega^2 x+\xi_x \tag{15}$$

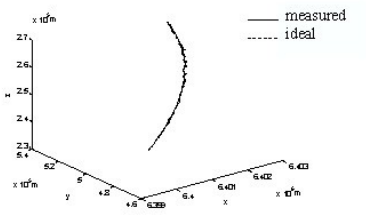
$$\ddot{y} = \frac{-g_m y}{(x^2+y^2+z^2)^{3/2}}+2\omega\dot{x}+\omega^2 y+\xi_y \tag{16}$$

$$\ddot{z} = \frac{-g_m z}{(x^2+y^2+z^2)^{3/2}}+\xi_z \tag{17}$$

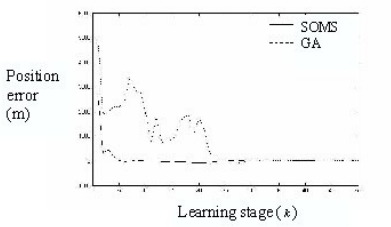
where $g_m = 3.986\times 10^5 km^3/s^2$ and $\omega = 7.2722\times 10^{-5} rad/s$ stand for the gravitational constant and the rotative velocity of the earth, respectively. $(\xi_x,\xi_y,\xi_z)$ are assumed to be continuous-time uncorrelated zero-mean Gaussian white noise processes with a constant variance $\sigma_f^2 = (0.1m/s^2)^2$. The measurement data is a 3-D position (x,y,z) with a zero mean and constant variance $\sigma_m^2 = (30m)^2$ noise. The ranges of the possible initial states $\underline{W}_j(0)$ were estimated to be

$$\begin{aligned}
&68.6\times10^5 m\le x_1(0)\le 68.8\times10^5 m\\
&2.7\times10^5 m\le x_2(0)\le 2.8\times10^5 m\\
&4.8\times10^5 m\le x_3(0)\le 4.9\times10^5 m\\
&110m/s\le x_4(0)\le 150m/s\\
&810m/s\le x_5(0)\le 850m/s\\
&1360m/s\le x_6(0)\le 1380m/s.
\end{aligned} \tag{18}$$

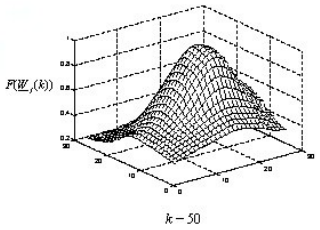
Within the ranges described in Eq.(18), the possible initial states of the missile were selected and stored into the 729 (27×27) neurons of the 2D SOM. The learning rates for each parameter were determined according to Eq.(9), with the time constant $\tau$ set to be 50 and $\eta_0$ and $\eta_1$ between 0.1 and 0.8. For the GA, the population size was selected




(a) Ideal and measured trajectories




(b) Estimated position error in the X-direction




(c) Neighborhood function $F(\underline{W}_j(k))$

Fig. 3. Simulation results for trajectory prediction using the SOMS and GA with bad estimates of the initial state and larger noise distribution: (a) the ideal and measured trajectories, (b) the estimated position error in the X-direction, and (c) the neighborhood function $F(\underline{W}_j(k))$ at k=50.

to be 729 to match with the SOMS, and the crossover and mutation probability 0.6 and 0.0333, respectively.

In the simulation, we investigated the performances of the SOMS and GA under the bad estimates of the initial state distribution. 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. Figure 3(a) shows the ideal and measured trajectories, Figure 3(b) the estimated position error (only the position error in the X-direction ($x_1$) is shown for illustration), and Figure 3(c) the variation the neighborhood function $F(\underline{W}_j(k))$ in the 2D neuron space. In Figure 3(c), from a random distribution in the beginning of the learning, $F(\underline{W}_j(k))$ gradually approximated the expected Gaussian distribution along with the stage of learning. The influence of bad estimate on the SOMS was mostly at the initial stage of the prediction. After the transient, the SOMS still managed to find the optimal state.

To further demonstrate the capability of the SOMS, we also applied the SOMS for the continuous optimization problem. Two standard functions (Griewank and Rosenbrock functions) [13] were used to test the performances of the SOMS and GA. The results show that the proposed SOMS performed better than GA. GA converged very slowly when the optimal initial state did not fall within the estimated range. And, it was not that straightforward to determine a proper population size and crossover and mutation probabilities to speed up its convergence rate. We thus conclude that the proposed SOMS performed better than the GA in robustness and effectiveness in the simulation examples.

# 5. CONCLUSION

In this paper, we have proposed an SOM-based search algorithm (SOMS), in which the SOM is used as a search mechanism. The proposed SOMS can be applied for optimal parameter search for the dynamic system in a real-time manner. The performance of the proposed scheme has been compared with those of the GA via the simulations on trajectory prediction and also continuous optimization problem. The results show that the SOMS algorithm performs better than GA. It may be because the SOM in our design adopts a somewhat organized search and the GA in some sense a random approach. To further exploit its search ability, in future work, we will apply the SOM for system identification and control.

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] G. A. Barreto and A. F. R. Araujo, "Identification and Control of Dynamical Systems Using the Self-Organizing Map," *IEEE Trans. on Neural Networks*, Vol. 15(5), pp. 1244-1259, 2004.

[2] G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," IEEE Computer, Vol. 21(3), pp. 77-88, 1988.

[3] Y.Y.Chen and K. Y. Young, "An Intelligent Radar Predictor for Hight-speed Moving-target Tracting," *International journal of Fuzzy Systems*, Vol. 6(2), pp. 90-99, 2004.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, New York, 1989.

[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.

[6] 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*, Vol. 33(6), pp. 877-888, 2003.

[7] T. Kohonen, *Self-Organizing Map*, Springer, Berlin, Germany, 1997.

[8] M. Milano, P. Koumoutsakos, and J. Schmidhuber, "Self-Organizing Nets for Optimization," *IEEE Trans. on Neural Networks*, Vol. 15(3), pp. 758-765, 2004.

[9] K. Obermayer and T. J. Sejnowski, ed., *Self-Organizing Map Formation: Foundation of Neural Computation*, MIT Press, Cambridge, 2001.

[10] 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*, Vol. 86(11), pp. 2240-2258, 1998.

[11] H. Shah-Hosseini and R. safabakhsh, "TASOM: a New Adaptive Self-Organization Map," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 33(2), pp. 271-282, 2003.

[12] M. C. Su and H. T. Chang, "Fast Self-Organizing Feature Map Algorithm," *IEEE Trans. on Nueral Networks*, Vol. 11(3), pp. 721-733, 2000.

[13] M. C. Su, Y. X. Zhao, and J. Lee, "Som-based Optimization," *IEEE Int. Conference on Neural Networks*, pp. 781-786, 2004.

[14] 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*, Vol. 4(1), pp. 86-96, 1993.