Generalization of the Self-Organizing Map: From Artificial Neural Networks to Artificial Cortexes

Tetsuo Furukawa and Kazuhiro Tokunaga

Kyushu Institute of Technology, Kitakyushu 808-0196, Japan,
furukawa@brain.kyutech.ac.jp, tokunaga@brain.kyutech.ac.jp
WWW home page: http://www.brain.kyutech.ac.jp/~furukawa

Abstract. This paper presents a generalized framework of a self-organizing map (SOM) applicable to more extended data classes rather than vector data. A modular structure is adopted to realize such generalization; thus, it is called a modular network SOM (mnSOM), in which each reference vector unit of a conventional SOM is replaced by a functional module. Since users can choose the functional module from any trainable architecture such as neural networks, the mnSOM has a lot of flexibility as well as high data processing ability. In this paper, the essential idea is first introduced and then its theory is described.

1 Introduction

In this paper, a generalized framework of Kohonen's self-organizing map (SOM) is presented. The generalization is realized by adopting a modular structure, thus it is called *modular network SOM* (mnSOM), which was first proposed by Tokunaga *et al*. [1, 2] Our aim is to develop a generalized SOM algorithm that allows users to generate a map of given objects that are not only vector data but also functions, dynamical systems, controllers, associative memories and so on. We also aim to give the capability of information processing to every nodal unit of a SOM. Thus, unlike the conventional SOM, the map generated by the mnSOM is no longer static and can be an assembly of information processors that can dynamically process data.

The idea of the mnSOM is simple: every reference vector unit of the conventional SOM is replaced by a trainable functional module such as a neural network (Figure 1). The functional modules can be designed to suit each application while keeping the backbone algorithm of the SOM untouched. This generalization strategy provides high degrees of both design flexibility and reliability to SOM users, because the mnSOM allows one to choose the functional modules from the great number of already proposed trainable architectures, and at the same time the consistent extension method assures the theoretical consistency, e.g., statistical properties, of the result.

As an example, let us consider a case in which an mnSOM user wants to make a map of controllers for a set of n-controlled objects. For this, all the user has to do is (i) determine the architecture of the controllers (it should be trainable, e.g., neural network controllers) as functional modules of the mnSOM, and (ii) define an appropriate distance measure that determines the distance between two controllers. The task of the mnSOM is to train those functional modules to be desired controllers for the n-objects, while at the same time generating a feature map that indicates the similarities

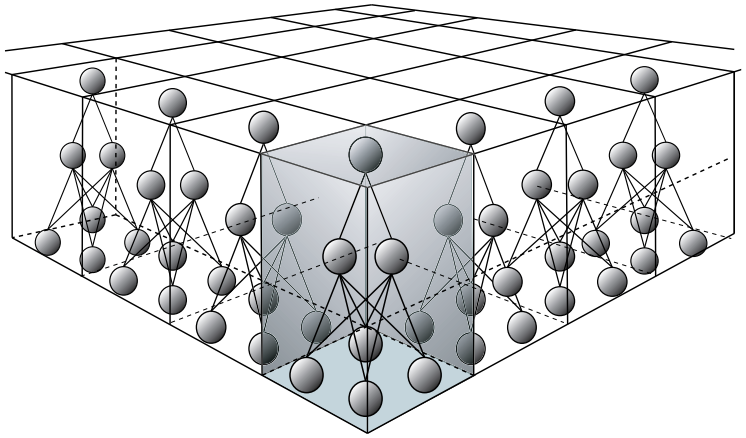

Fig. 1. The architecture of mnSOM

or differences between those controllers. If the controlled-objects A and B have similar dynamics, then the corresponding controllers should be located near each other in the map space of the mnSOM, whereas if controlled-objects C and D have quite different dynamics, then those controllers should be arranged further apart. Additionally, the intermediate modules are expected to become controllers for objects that have intermediate dynamics of the given ones. After the training has finished, the user can then use it as an assembly of controller modules that can adapt the dynamic changes of the target object. This is a new aspect that is not found in the conventional SOM. Therefore, the mnSOM is expected to greatly enlarge the number of fields for applications of SOMs .

2 Architecture of an mnSOM

The architecture of an mnSOM is shown in Figure 1. The architecture is such that each vector unit of a conventional SOM is replaced by a trainable functional module. These modules are usually arrayed on a lattice that represents the coordinates of the feature map. (Though any modifications such like a growing-map, a hierarchical map and a neural gas network are all available; here, we consider the simplest map structure).

Figure 1 illustrates the case of multilayer perceptron (MLP) modules as a typical case, but many other module types are available. Table 1 shows a catalogue of module types that have been tried. In the case of MLP modules, i.e., MLP-mnSOM, each MLP-module represents a nonlinear function, and as the result the entire mnSOM generates a map of functions. Therefore, MLP-mnSOM is an SOM in function space rather than vector space [1–4]. Users can also employ radial basis function network modules (RBF-mnSOM) instead of the conventional MLPs. In such cases, the distance measure is defined in the function space. Siblings of MLP are all possible to be mnSOM modules. For example, recurrent neural networks (e.g., Jordan type and Elman type) and autoassociative neural networks (i.e., 3- or 5-layer autoencoder MLPs with a sand clock structure) can be employed as well [5–8].

Table 1. Examples of module types and their applications

Module type (Name)	Object type	Applications
Layer type		
– Multilayer perceptron (MLP-mnSOM)	nonlinear functions	Weather dynamics [1–3] Bifurcation map of logistic mapping [5, 6]
– Autoassociative network (ANN-mnSOM)	manifolds	2D images of 3D objects [7] Texture map [8] Periodical waveforms
– Reccurent network (RNN-mnSOM)	dynamical systems	Dumped oscillatory systems [7] Bifurcation map of BVP model [5, 6] Autonomous mobile robots [15, 16]
– Predictor & cotroller pair (SOAC)	adaptive controllers	Inverted pendulums [17] Autonomous underwater vehicle [19–21]
– RBF network (RBF-mnSOM)	nonlinear functions	
– Single-layer perceptron (Operator map)	linear operators	[12]
SOM type		
– SOM (SOM^2)	manifolds	2D images of 3D objects [9, 10] Face image recognition [9] Shape classification [11]
– Neural gas network (NG^2, NG-SOM)	density functions	Handwritten character recognition [11]
– Local linear map	nonlinear functions	
Stochastic type		
– Hopfield network – Boltzmann machine	associative memories	
Component analysis type		
– PCA (ASSOM)	linear subspaces	[13]
– Nonlinear PCA	nonlinear subspaces	*See autoassociative network module*
Single neuron type		
– Hebbian neuron (Basic SOM)	static vectors	
Other module type		
– Image filter	visual image filters	Adaptive visual filter [22]

Another big group of mnSOMs is made up of the SOM module types proposed by Furukawa [9, 10]. SOM-module-mnSOM, called SOM^2, is a "self-organizing homotopy" rather than a "self-organizing map" [11]. One of the prominent properties of this group is that this type mnSOM can have a nested structure like a Russian doll. For example, SOM^2 can be a module of a meta-mnSOM. Thus, SOM^2-module mnSOM, i.e., SOM^3, is also possible. It is easy to extend the n-th order, i.e., SOM^n as SOM^{n-1}-module mnSOM. SOM^2 and its family demonstrate their potential when tasks involve nonlinear manifolds or nonlinear subspaces.

Stochastic type networks such as a Boltzmann machine and a Hopfield network make up another group. They are expected to generate a "map of memories".

The mnSOM includes some variations of SOM that have been proposed previously. If one employs a linear operator module, then it is an Operator Map as proposed by Kohonen [12]. When a principle component analysis (PCA) module is used, then the mnSOM becomes an ASSOM [13]. If one employs Hibbian neurons as the functional modules, then the mnSOM becomes a conventional SOM [14]. Therefore, the mnSOM is a generalization of an SOM rather than an extension, because it includes the conventional cases.

Though there are many architectures that have not been tried before, they would be also available as modules of the mnSOM. Users can derive the algorithm theoretically described in the next section, without needing to try any heuristic ways.

3 Theory of mnSOM

Now let us describe the generalized theory of an mnSOM algorithm. Suppose that an mnSOM user is trying to map a set of I objects, $\mathcal{O} = \{O_1, \ldots, O_I\}$. In a conventional SOM, each data vector is A mapping object, whereas in the case of an MLP-mnSOM, each object corresponds to each of the nonlinear functions i.e., the input–output relations.

There is one big difference between a conventional SOM and the generalized SOM case. In the case of the conventional SOM, all the mapping objects, i.e., the data vectors, are known and there is no need to estimate the objects. But in the generalized case, it often happens that the entities of the objects are unknown. For example, let us consider a case in which a user is trying to map a set of dynamical systems. In such a case, the observed input and output signals of the systems are usually given, however, their dynamics are unknown in most cases. Therefore, the user should identify those dynamics in parallel with generating their self-organizing map. Thus the mnSOM should solve the simultaneous estimation problem.

Let us assume that $D_i = \{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,J}\}$ is the dataset observed from the i-th object O_i. If the mapping objects are systems or functions, then $\mathbf{r}_{i,j}$ is defined as a set of input-output vectors $\mathbf{r}_{i,j} = (\mathbf{x}_{i,j}, \mathbf{y}_{i,j})$ observed from the i-th system. Suppose that the mnSOM has K functional modules $\{M^1, \ldots, M^K\}$, which are designed to have the ability of regenerating, or mimicking the objects. In other words, a module is capable of approximating an object O_i after training by D_i. Suppose further that the property of each function module M^k is determined by a parameter set θ^k. In the case of MLP-mnSOM, θ^k is the weight vector of the k-th MLP module. Each functional module M^k

is given a fixed position ξ^k in the map space. Therefore, ξ^k assigns the coordinates of M^k in the map space, while θ^k determines the position in the data space.

Under such a situation, the tasks of the mnSOM are (i) to identify the entities of $\{O_i\}$ from the observed datasets $\{D_i\}$ by training the function modules $\{M^k\}$, and (ii) to generate a map that shows the degrees of similarity and difference between the objects. These two tasks should be processed in parallel. Note that the map generated by the mnSOM is expected to show the relationships between the entities of the objects, direct comparisons between the datasets are meaningless.

The mnSOM user needs to define an appropriate distance measure $L^2(O_i, M^k)$ that signifies the difference between an object O_i and a module M^k. Since the distance measure depends on how the user wants to define similarities and differences between two objects, the measure should be defined depending on the user's purpose.

By using the distance measure, an important derivative definition, namely, the definition of mass center can be determined as follows.

$$\bar{O}(\mathbf{m}, \mathcal{O}) \triangleq \arg\min_{O} \sum_{i=1}^{I} m_i L^2(O_i, O) \tag{1}$$

Here $\bar{O}$ is the center of mass of the given objects $\mathcal{O} = \{O_1, \ldots, O_I\}$ with the weights $\mathbf{m} = (m_1, \ldots, m_I)$. If $\mathcal{O}$ belongs to a vector space, then $\bar{O}$ is given by

$$\bar{O} = \frac{m_1 O_1 + \cdots + m_I O_I}{m_1 + \cdots + m_I}. \tag{2}$$

Since the entities of the objects are assumed to be unknown, we can measure only the distance between an estimated object and a module, i.e., $L^2(\hat{O}(D_i), M^k)$. Here $\hat{O}(D_i)$ is the object entity estimated from D_i.

Each module is updated so as to be the center of mass, the weights of which are given by the neighborhood function. Thus, the update algorithm of mnSOM is described as

$$\theta^k(t+1) = \arg\min_{\theta} \sum_{i=1}^{I} \phi_i^k(t) L^2(\hat{O}(D_i), M(\theta)). \tag{3}$$

$\phi_i^k(t)$ is the mass of the i-th object for the k-th module given by the neighborhood function at time t. Usually a gauss function is used as the neighborhood function;

$$\phi_i^k(t) = \exp\left[-\frac{\left\|\xi_i^* - \xi_i^k\right\|^2}{\sigma(t)}\right] \tag{4}$$

Here ξ_i^* denotes the coordinate of the winner module of the i-th object.

In many cases (but not always) the estimated distance $L^2(\hat{O}(D_i), M^k)$ can be approximated by the mean square error between the module M^k and the dataset D_i such as

$$L^2(O_i, M^k) \simeq E_i^k \triangleq \frac{1}{J} \sum_{j=1}^{J} \left(e_{i,j}^k\right)^2. \tag{5}$$

Here $e_{i,j}^k$ is the error between the data vector $\mathbf{r}_{i,j}$ and the corresponding output of the k-th module. In such cases, the update algorithm (3) becomes much easier, as follows.

$$\theta^k(t+1) = \arg\min_{\theta} \sum_{i=1}^{I} \sum_{j=1}^{J} \phi_i^k(t) \left(e_{i,j}^k\right)^2. \tag{6}$$

The algorithm of the generalized SOM consists of three processes; like in the case of the conventional SOM. First, in the *competitive process*, the least average error module becomes the "winner" or the "best matching module" (BMM) for a given dataset. The BMM is determined for every dataset. Second, in the *cooperative process*, the learning weight is determined using the neighborhood function. This process is identical with the conventional one. Finally, in the *adaptive process*, all modules are updated so as to be the mass center of the objects with the weights $\{\phi_i^k\}$. These three processes are iterated reducing the neighborhood size until the network gets to a steady state.

The algorithm described above is general case; now we look at the case of an MLP-mnSOM as an example. Since MLPs represent nonlinear functions, the distance measure is defined in function space.

$$L^2(O_i, M^k) = \int \left\| f_i(\mathbf{x}) - g^k(\mathbf{x}) \right\|^2 p(\mathbf{x}) d\mathbf{x} \tag{7}$$

$f_i(\mathbf{x})$ is the i-th object, i.e., the i-th nonlinear function and $g^k(\mathbf{x})$ is the function represented by the k-th MLP module. Here $f_i(\mathbf{x})$ is assumed to be unknown, and the observed input–output data are supposed to be given. Thus, $D_i = \{\mathbf{r}_{i,j}\} = \{(\mathbf{x}_{i,j}, \mathbf{y}_{i,j})\}$ is available to use as the training dataset. In this case, the mean square error is determined by the error between the output of the k-th module $g^k(\mathbf{x}_{i,j})$ and the actual (desired) output $\mathbf{y}_{i,j}$, i.e.,

$$e_{i,j}^k = \left\| g^k(\mathbf{x}_{i,j}) - \mathbf{y}_{i,j} \right\|. \tag{8}$$

The least mean square error module for D_i is determined as the winner (BMM) of the i-th object. In this case, the weight vectors of MLP-modules are updated by the backpropagation algorithm.

$$\Delta\theta^k = -\eta \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{\phi_i^k(t)}{\phi_1^k + \cdots + \phi_I^k} \frac{\partial e_{i,j}^k}{\partial \theta^k} \tag{9}$$

Note that (9) updates the MLP toward the center of mass of the given functions with the weights $\{\phi_i^k\}$. This backpropagation learning is iterated several times for all data vectors (not only once) with fixing $\phi_i^k(t)$, so that θ^k is updated enough. The detailed algorithms of individual module types have been described in previous works [2, 3, 6, 7, 9, 10].

4 Conclusion: Can our mnSOM be an artificial cortex?

The mnSOM has several advantages comparing to other neural network architectures. First, the mnSOM can process larger tasks than single neural networks, and it has less

interference of memories because of its modular structure. Second, the entire output of an mnSOM is trained in a supervised manner with given datasets, while maps of functions are organized in an unsupervised manner. Therefore the mnSOM seems to transcend the dualism of supervised and unsupervised learning. Finally, the mnSOM is a meta-learning framework which rules an assembly of functional modules. The flexibility of the type of module is also an advantage inherent in the mnSOM.

Interestingly, the architecture of the mnSOM looks similar to the column structure of our cortex. Each module looks like a functional column of the cortex, and the map in the mnSOM corresponds to the map of a brain. Of course the mnSOM was not invented by mimicking the cortex, and it is just a straightforward generalization of Kohonen's SOM. But considering the above advantages, our mnSOM is expected to be a good platform to initiate an artificial cortex; though much remains to be done to realize that far off goal.

Acknowledgement

This work was partially supported by a Center of Excellence Program (Center #J19) granted by MEXT of Japan. This work was also partially supported by a Grant-in-Aid for Scientific Research (C) granted by MEXT of Japan.

References

1. Tokunaga, K., Furukawa, T., Yasui, S.: Modular network SOM: Extension of SOM to the realm of function space. Proc. of WSOM2003 (2003) 173–178
2. Tokunaga, K., Furukawa, T., Yasui, S.: Modular network SOM: Self-organizing maps in function space. Neural Information Processing – Letters and Reviews **9**(1) (2005) 15-22
3. Furukawa, T., Tokunaga, K., Morishita, K., Yasui, S.: Modular network SOM (mnSOM): From vector space to function space. Proc. of IJCNN2005 (2005) 1581-1586
4. Tokunaga, K., Furukawa, T.: Modular network SOM: Theory, algorithm and applications. Proc. of ICONIP2006 (2006)
5. Furukawa T., Tokunaga K., Kaneko S., Kimotsuki K., Yasui, S.: Generalized self-organizing maps (mnSOM) for dealing with dynamical systems. Proc. of NOLTA2004 (2004) 231–234
6. Kaneko, S., Tokunaga, K., Furukawa, T.: Modular network SOM: The architecture, the algorithm and applications to nonlinear dynamical systems. Proc. of WSOM2005 (2005) 537-544
7. Tokunaga, K., Furukawa, T.: Nonlinear ASSOM constituted of autoassociative neural modules. Proc. of WSOM2005 (2005) 637-644
8. Tokunaga, K., Furukawa, T.: Realizing the nonlinear adaptive subspace SOM (NL-ASSOM) Proc. of BrainIT 2005 (2005) 76
9. Furukawa, T.: SOM^2 as "SOM of SOMs". Proc. of WSOM2005 (2005) 545-552
10. Furukawa, T.: SOM of SOMs: Self-organizing map which maps a group of self-organizing maps. Lecture Notes in computer Science, **3696** (2005) 391-396
11. Furukawa, T.: SOM of SOMs: An extension of SOM from 'map' to 'homotopy'. Proc. of ICONIP2006 (2006)
12. Kohonen, T.: Generalization of the Self-organizing map. Proc. of IJCNN93 (1993) 457–462
13. Kohonen, T., Kaski, S., Lappalainen, H.: Self-organized formation of various invariant-feature filters in the adaptive-subspace SOM. Neural Computation **9** (1997) 1321–1344
14. Kohonen, T.: Self-Organizing Maps, 3.ed., Springer (2001)

15. Aziz Muslim, M., Ishikawa, M., Furukawa, T.: A new approach to task segmentation in mobile robots by mnSOM. Proc. of IJCNN2006 (2006)
16. Aziz Muslim, M., Ishikawa, M., Furukawa, T.: Task segmentation in a mobille robot by mnSOM: A new approach to training expert modules. Proc. of ICONIP2006 (2006)
17. Minatohara, T., Furukawa, T.: Self-organizing adaptive controllers: Application to the inverted pendulum. Proc. of WSOM2005 (2005) 41-48
18. Minatohara, T., Furukawa, T.: A proposal of self-organizing adaptive controller (SOAC). Proc. of BrainIT2005 (2005) 56
19. Nishida, S., Ishii, K.: An adaptive controller system using mnSOM. Proc. of BrainIT 2005 (2005) 85
20. Nishida, S., Ishii, K.: An adaptive neural network control system using mnSOM. Proc. of OCEANS2006 (*in press*)
21. Nishida, S., Ishii, K.: An Online Adaptation Control System using mnSOM. Proc. of ICONIP2006 (2006)
22. Horio, K., Suetake, N.: Inverse halfoning based on pattern information and filters constructed by mnSOM. Proc. of BrainIT 2005 (2005) 102