Modular Network SOM

Kazuhiro Tokunaga

Tetsuo Furukawa

Department of Brain Science and Engineering,

Graduate School of Life Science and Systems Engineering,

Kyushu Institute of Technology, Kitakyushu, Japan

Acknowledgements

We are very grateful to Prof. Syozo Yasui for insightful discussions. This work was partially supported by a Center of Excellence Program (Center #J19) granted in 2003 to the Department of Brain Science and Engineering, Kyushu Institute of Technology, by the Japan Ministry of Education, Culture, Sports, Science and Technology. This work was also partially supported by the Japan Ministry of Education, Science, Sports and Culture Grant-in-Aid for Scientific Research (C), 17500193, 2005.

Corresponding Author

Department of Brain Science and Engineering, Kyushu Institute of Technology,

2-4 Hibikino, Wakamatsu-ku, Kitakyushu 808-0196, Japan

Tel: +81–93–695–6124

Fax: +81–93–695–6134

E-mail: furukawa@brain.kyutech.ac.jp (T. Furukawa)

Modular Network SOM

Abstract

This study aims to develop a generalized framework of an SOM called a *modular network SOM (mnSOM)*. The mnSOM has an array structure consisting of functional modules that are trainable neural networks, e.g., multi-layer perceptrons (MLPs), instead of the vector units of the conventional SOM. In the case of MLP-modules, an mnSOM learns a group of systems or functions in terms of the input-output relationships in parallel with generating a feature map of them. Thus an mnSOM with MLP modules is an SOM in function space rather than in vector space. In this paper, first, as an example, we focus on a class of mnSOM that consists of MLP modules and introduce the architecture and algorithm. Then, a more generalized framework is described. Finally, some simulation results of an MLP-module-mnSOM are presented.

Keywords: SOM, modular network, function space

1 Introduction

Kohonen's Self-Organizing Map (SOM) (Kohonen, 2001) is an unsupervised learning algorithm for generating topology-preserving transformation from a high-dimensional data vector space to a low-dimensional map space, and it is a powerful tool used in many areas such as data mining, analysis, classification, and visualization. Applications of SOM have spread into numerous areas such as web-based search (Kohonen, Kaski, Lagus, Salojärvi, Honkela, Paatero, & Saarela, 2000), bioinformatics (Abe, Kanaya, Kinouchi, Ichiba, Kozuki, & Ikemura, 2003), and finance (Deboeck, & Kohonen, 1998), and its importance continues to increase. Despite its increasing importance, however, the conventional SOM and most of its extensions can only deal with vectorized data. If one wishes to deal with a non-vector dataset, then one needs to make the data vectorized in advance or modify the SOM itself to adapt to the data type. Therefore, generalizing the SOM family is an unavoidable problem in which the SOM algorithm is described independently of the data type.

Our study aims to develop a generalized framework of the conventional SOM by adopting the idea of a modular network in what we call a *modular network SOM (mnSOM)*. The idea of the mnSOM is simple: every vector unit of the conventional SOM is replaced by a trainable functional module such as a neural network. Thus the architecture of the mnSOM is an assembly of the functional modules arrayed on a lattice (Fig. 1). The functional modules can be designed to suit each application while keeping the backbone algorithm of the SOM untouched. This generalization strategy provides both a high degree of design flexibility and reliability to SOM users because the mnSOM allows one to choose the functional modules from the great number of neural architectures already proposed, and at the same time the consistent extension method assures the theoretical consistency, e.g., statistical properties, of the results. Furthermore, this strategy allows for the functional capability of data processing to all nodal units of the SOM. Thus, our mnSOM can be used as an assembly of data processors after training. This is another advantage of the mnSOM.

As an example, let us consider a case in which an mnSOM user is trying to make a map of the weather dynamics of the world's cities. In this case, the task is to generate a feature map that visualizes the degrees of similarity or difference between the weather dynamics of each city; if the weather patterns of New York and Tokyo are described by similar dynamics, then the two cities will be neighbors in the weather map, whereas if London and Singapore have different dynamics, then they will be mapped far apart. Of course one could make a map by using a conventional SOM after vectorizing all the weather data. However, such a map would provide no information about the relationships between the dynamics. To achieve this task, one should identify the weather dynamics of every city and then generate a feature map by comparing the dynamics. This is what one can do with an mnSOM. By using an mnSOM, all one has to do is: (i) design a neural network that can learn the weather dynamics of a single city, (ii) employ the neural network as the functional module of the mnSOM, and (iii) train the mnSOM by entering world wide weather data. In addition, the mnSOM can be used as an assembly of weather forecast systems after training.

A similar concept has been proposed as Operator Map (Kohonen, 1993), suggesting that SOM can be extended from vector space to function space. From the viewpoint, operator map was the first proposal of this generalization, though Kohonen only showed the algorithm for linear operator case. Extensions of SOM from vectors to subspaces are also related to this work, such as ASSOM (Kohonen, Kaski, & Lappalainen, 1997), and parameterized SOM (Walter & Ritter, 1996). But our purpose is to establish a more general and flexible framework that takes over these cases, thus our aim is to challenge to build a unified theory of SOM variations with functional modules. One may also find a resemblance to the Local Linear Map (Martinetz, Ritter, & Schulten, 1990), though our mnSOM is essentially different. The purpose of a Local Linear Map is to represent a single, nonlinear function as the combination of the local linear operators, whereas the purpose of an mnSOM is to generate a feature map of an assembly of nonlinear functions which are represented by nonlinear function modules.

In this paper, the ideas, architecture, and algorithms of the proposed mnSOM are first outlined, with special attention given to the case of MLP-modules. Then, we try to describe a more generalized framework. Finally, the simulation results of some application examples are presented.

2 Theoretical framework

2.1 mnSOM with MLP modules

The architecture of an mnSOM is shown in Fig. 1. Basically, the architecture is such that each vector unit of the conventional SOM is replaced by a functional module. These modules are arrayed on a lattice that represents the coordinates of the feature map. Many trainable algorithms can be employed as functional modules, though Fig. 1 illustrates the case of an MLP-module. Since each module represents a certain '*functional feature*' determined by the module architecture, the entire mnSOM represents a map of those features.

In order to outline the key concept of mnSOM, let us start from an example case in which an mnSOM user tries to make a map of a group of functions $\{f_1(\cdot), \ldots, f_I(\cdot)\}$, which are unknown initially. Instead, the sample datasets $\{D_1, \ldots, D_I\}$ are assumed to be given. Here, $D_i = \{(\boldsymbol{x}_{i,1}, \boldsymbol{y}_{i,1}), \ldots, (\boldsymbol{x}_{i,J}, \boldsymbol{y}_{i,J})\}$ is a class of input-output vectors sampled from the ith function. Thus, the vector pairs satisfy $\boldsymbol{y}_{i,j} = f_i(\boldsymbol{x}_{i,j})$. The tasks of the mnSOM are (i) to identify the unknown functions, and (ii) to generate a feature map of those functions. In other words, the task of the mnSOM is to undertake topological mapping from a function space to a map space. Therefore, the mnSOM should be an SOM in the function space rather than the vector space.

What the user needs to do first is select the module architecture. In this case, the module is required to have the ability to learn a function from a set of data vectors. Therefore, MLPs, radial basis function (RBF) networks, and their siblings are the candidates of the module type. Here, let us consider the case of MLP-modules, i.e., MLP-module-mnSOM (MLP-mnSOM). Hereafter, the MLP-mnSOM is assumed to have K MLP modules which represent K functions $g^1(\cdot), \ldots, g^K(\cdot)$. These functions are determined by the weight vectors $\boldsymbol{w}^1, \ldots, \boldsymbol{w}^K$, which are updated by the backpropagation algorithm. (In this paper, let the subscripts represent the indexes of mapping objects or datasets, whereas the superscripts refer to the indexes of mnSOM modules.)

Fig.1 about here.

The algorithm of mnSOM consists of four processes: *evaluative process*, *competitive process*, *cooperative process*, and *adaptive process*. In the *evaluative process*, the outputs of all MLP modules are evaluated for each input-output data vector pair. If, for example, an input data vector $\boldsymbol{x}_{i,j}$ is picked up, then the output of the kth module $\tilde{\boldsymbol{y}}^k_{i,j} = g^k(\boldsymbol{x}_{i,j})$ is calculated for that input. This calculation process is repeated for $k = 1, \ldots, K$, using the same input $\boldsymbol{x}_{i,j}$. After evaluating all outputs for all inputs, then the errors of all modules for each data class are evaluated. Now let E^k_i

be the error of the kth module for the dataset from the ith function, i.e.,

$$E_i^k = \frac{1}{J}\sum_{j=1}^{J}\left\|\tilde{\boldsymbol{y}}_{i,j}^k - \boldsymbol{y}_{i,j}\right\|^2 \tag{1}$$

$$= \frac{1}{J}\sum_{j=1}^{J}\left\|g^k(\boldsymbol{x}_{i,j}) - f_i(\boldsymbol{x}_{i,j})\right\|^2. \tag{2}$$

If J (the number of samples) is large enough, then the square distance between the kth module and the ith system in the function space is approximated by the error E_i^k as follows.

$$L^2(g^k, f_i) = \int \left\|g^k(\boldsymbol{x}) - f_i(\boldsymbol{x})\right\|^2 p_i(\boldsymbol{x})d\boldsymbol{x} \simeq E_i^k \tag{3}$$

Here, $p_i(\boldsymbol{x})$ denotes the probability density function of the input data vectors $\boldsymbol{x}_{i,j}$ of the ith system. In this paper it is assumed that $\{p_i(\boldsymbol{x})\}$ for $i = 1, \ldots, I$ are approximately the same as $p(\boldsymbol{x})$, due to the normalization of the data distribution for each class. [1]

In the *competitive process*, the module which minimizes the error E_i^k is determined as the *best matching module (BMM)*, i.e., the winner module for the ith class. Thus, let k_i^* be the module number of the BMM for the ith class. Then k_i^* is defined as

$$k_i^* = \arg\min_k E_i^k. \tag{4}$$

In the *cooperative process*, the learning weights are calculated by using the neighborhood function. Let $\psi_i^k(t)$ denote the learning weight of the kth module for the ith class at calculation time t. Then $\psi_i^k(t)$ is given by the following equations.

$$\psi_i^k(t) = \frac{h\left(\|\boldsymbol{\xi}^k - \boldsymbol{\xi}^{k_i^*}\|; t\right)}{\sum_{i'=1}^{I} h\left(\|\boldsymbol{\xi}^k - \boldsymbol{\xi}^{k_{i'}^*}\|; t\right)} \tag{5}$$

$$h(l;t) = \exp\left[-\frac{l^2}{2\sigma^2(t)}\right] \tag{6}$$

Here $\boldsymbol{\xi}^k$ expresses the position of the kth module in the map space, and $h(l;t)$ is a neighborhood function which shrinks with the calculation time t. By using the learning weights ψ_i^k, the energy function of the kth module E^k is given as follows.

$$E^k(t) = \sum_{i=1}^{I} \psi_i^k(t)\, E_i^k(t) \tag{7}$$

[1]If $\{p_i(\boldsymbol{x})\}$ are different from each other significantly, then it is impossible to define the distance measure between the given functions. Thus, comparisons between the functions have no meaning in such cases. There is an issue of how to deal with such cases, but it is not the focus of this paper, since it depends on how the user wants to define similarities and differences of the functions.

It means that the mnSOM algorithm gives the energy function of each module by using the SOM algorithm. Note that E^k has the global minimum point at which $g^k(\cdot)$ satisfies

$$g^k(\boldsymbol{x}) = \sum_{i=1}^{I} \psi_i^k(t)\, f_i(\boldsymbol{x}). \tag{8}$$

This is the essence of the generalization.

In the *adaptive process*, all MLP modules are updated by the backpropagation learning algorithm, so that every module get closed to the global minimum point determined by the energy funtion as follows.

$$\Delta \boldsymbol{w}^k = -\eta \frac{\partial E^k}{\partial \boldsymbol{w}^k} = -\eta \sum_{i=1}^{I} \psi_i^k \frac{\partial E_i^k}{\partial \boldsymbol{w}^k} \tag{9}$$

Therefore the kth MLP-module is updated so as to converge to the mass center of $\{f_i(\boldsymbol{x})\}$ given by (8). The backpropagation learning is repeated until all modules are sufficently updated. For example, (9) is repeated around thousand times for every iteration time t. During the training of MLPs, each input vector $\boldsymbol{x}_{i,j}$ is presented one by one as the input, and the corresponding output $\boldsymbol{y}_{i,j}$ is presented as the desired signal. These four processes are iterated, with shrinking the neighborhood function area until the network gets to a steady state.

This algorithm of the MLP-mnSOM is a straight-forward extension of the batch version SOM algorithm. In fact, if the MLP modules have only bias units, i.e., every module always outputs a constant vector, then the above algorithm becomes the same as the conventional one. Therefore, MLP-mnSOM includes the conventional case. If the modules represents linear operators, then the mnSOM can be called as the operator map.

For the help of understanding the algorithm, it is worth noting some important points. First, since the energy function E^k is determined for each module, each global minimum point is different from all the others. Therefore MLP-modules are updated toward to different goals individually. Second, it is important to stress that the energy function $E^k(t)$ does not define the energy function of the entire mnSOM. Of course the sum of the energy functions $E(t) = \sum_k E^k(t)$ is minimized in the adaptive process, however, this total energy $E(t)$ is only valid during the adaptive process at calculation time t, in which the learning weights $\{\psi_i^k(t)\}$ are fixed. In the next iteration, i.e., at time $t + 1$, $\psi_i^k(t + 1)$ is updated, consequently the energy function $E^k(t + 1)$ is also updated.

2.2 Generalized framework of mnSOM

Now let us try to describe more generalized cases, i.e., an mnSOM algorithm for more generalized classes and for more generalized module types. Suppose that an mnSOM user is trying to map a set of I objects, $O = \{O_1, \ldots, O_I\}$. In the conventional SOM, each data vector is the mapping object, whereas in the previous example of MLP-mnSOM, each object corresponds to each of the functions. It is not necessary that the entities of objects are identified in advance, i.e., they are generally assumed to be unknown. Instead, a sample dataset observed from each object is assumed to be available. Thus, let $D_i = \{\boldsymbol{r}_{i,1}, \ldots, \boldsymbol{r}_{i,J}\}$ be the dataset observed from the ith object O_i. In this paper, it is assumed that $\{\boldsymbol{r}_{i,j}\}$ are vector data.

Suppose that the mnSOM have 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 that the property of each functional module M^k is determined by a parameter set $\boldsymbol{\theta}^k$. Thus in the case of MLP-mnSOM, $\boldsymbol{\theta}^k$ is the weight vector of the kth MLP module. Suppose further that each functional module M^k is given a fixed position $\boldsymbol{\xi}^k$ in the map space. Therefore $\boldsymbol{\xi}^k$ assigns the coordinates of M^k in the map space, while $\boldsymbol{\theta}^k$ determines the position in the data space.

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

The mnSOM user needs only to give an appropriate definition of a distance measure that signifies the difference between two objects or between an object and a module, $L(O_i, M^k)$. By using this distance measure, two important derivative definitions can also be determined. One is the definition of *error* between a data vector and a module. Suppose that $e^k_{i,j}$ represents the square error between a data vector $\boldsymbol{r}_{i,j}$ and a module M^k. Then the average error of D_i and M^k can be expected to be an approximation of the square distance between O_i and M^k. Thus, the average error E^k_i is regarded as the estimated square distance $L^2(O_i, M^k)$, as follows.

$$L^2(O_i, M^k) \simeq E^k_i \triangleq \frac{1}{J} \sum_{j=1}^{J} e^k_{i,j} \tag{10}$$

This definition of error is usually associated with the module, because the module employed here is supposed to have the ability to represent O_i. Thus, the module chosen by the user is expected to minimize E_i^k by training with D_i.

The other derivative definition is that of the *mass center of objects*, determined as

$$\bar{O}(\boldsymbol{\psi}, \mathcal{O}) \triangleq \arg\min_{O} \sum_{i=1}^{I} \psi_i L^2(O_i, O). \tag{11}$$

$\bar{O}$ is the mass center of $\mathcal{O} = \{O_i\}$ with weights $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_I)$. Since the mass center O is not affected by the scaling of weights, here ψ_i are assumed to be $\sum_i \psi_i = 1$. It means that the distance measure defined by the user also determines the definition of mass center. In other words, user should define the distance measure so that the mass center of the given objects represents 'the intermediate objects'.

The generalized algorithm of mnSOM also consists of the four processes described above for the case of MLP-modules. In the *evaluative process*, the distance of every combination of O_i and M^k is estimated by evaluating the average error E_i^k. The succeeding competitive and cooperative processes are identical to those of the MLP-module case. In the *competitive process*, the BMMs are determined by (4), and then the learning weights $\{\psi_i^k\}$ are calculated by (5) and (6) in the *cooperative process*. Then the energy function of the kth module at time t is defined as

$$E^k(t) = \sum_{i=1}^{I} \psi_i^k(t)\, E_i^k \simeq \sum_{i=1}^{I} \psi_i^k(t)\, L^2(O_i, M^k). \tag{12}$$

Finally, in the *adaptive process*, all modules are updated so as to be the mass center of the objects with the weights $\boldsymbol{\psi}^k(t) = (\psi_1^k(t), \ldots, \psi_I^k)$. Ideally, M^k is expected to be innovated as

$$M^k(t+1) = \bar{O}(\boldsymbol{\psi}^k(t), \mathcal{O}) = \arg\min_{M} \sum_{i=1}^{I} \psi_i^k(t)\, L^2(O_i, M). \tag{13}$$

Note that (13) cannot be evaluated directly due to the lack of information regarding the entity of $\{O_i\}$. Instead, it can be estimated by substituting the average error E_i^k for the square distance $L^2(O_i, M^k)$, as shown in (10). Consequently, the update algorithm is obtained as

$$\boldsymbol{\theta}^k(t+1) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{I} \psi_i^k(t) E_i^k(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} E^k(\boldsymbol{\theta}). \tag{14}$$

Though the implementation of (14) depends on the learning algorithm of the modules, in most cases it would be a natural modification of the algorithm that originated from the module type because the module algorithm is supposed to minimize E_i^k when D_i is given to the module. This point is discussed in more detail later. Finally, these four processes are iterated until the network gets to a steady state.

In this generalized mnSOM algorithm, users need to choose an appropriate distance measure, which is usually accompanied by the module architecture. Since the backbone algorithm of the SOM is kept untouched, the generalized SOM is regarded as a SOM in the object space defined by the distance measure.

2.3 How to avoid local minima

For MLP-mnSOMs, local minima cause a serious problem because the modules that are caught by local minima destroy the continuity of the map at those points. To make matters worse, the chance of falling into a local minimum increases when the number of modules increases. Therefore an additional algorithm for avoiding local minima is necessary.

As shown in (7), the task of the kth module is to minimize the energy function $E^k = \sum_i \psi_i^k E_i^k$. This means that M^k is expected to minimize E^k among all modules. Now suppose that the module M^k is replaced by the other module $M^{k'}$ temporarily. If the energy function E^k is improved by this replacement, it can be reasoned that M^k has likely fallen into a local minimum. In such a case, M^k can be 'rescued' by copying $\boldsymbol{\theta}^{k'}$ to $\boldsymbol{\theta}^k$. This algorithm is formulated as follows.

$$\boldsymbol{\theta}^k(t) := \boldsymbol{\theta}^{k'}(t) \qquad \text{if} \quad \tilde{E}^{k,k'} < \beta E^k \tag{15}$$

$$\tilde{E}^{k,k'} \triangleq \sum_{i=1}^{I} \psi_i^k E_i^{k'} \tag{16}$$

Here $\tilde{E}^{k,k'}$ is the energy function when M^k is replaced by $M^{k'}$ temporarily, and β is a safety factor which is usually around 0.8–0.9. By making such comparisons between neighbors *before the adaptive process*, the MLP module in the local minimum is reset to a better initial state given by the neighbors, and after that, the module is updated toward to its own global minimum point.

By employing this additional algorithm, the chance of falling into local minima will decrease as the number of modules increases because the chance of rescue also increases. Therefore this algorithm turns the disadvantage of having many MLPs into an advantage. In addition, this algorithm has no adverse effect, since this algorithm does not modify the energy function at all. The neighbors only lend a hand to the module in trouble, by giving a better restart point for the module. Once after escaping a local minimum, then the module can be updated toward to its own target, which is different (but similar) from its neighbors. Thus the algorithm increases the chance of success without affecting the succeeded result.

3 Simulations and results

3.1 Maps of cubic functions

A simulation study was undertaken to examine the performance of mnSOM in some examples. In this paper, the module architecture was limited to MLP cases, though some other architectures have been examined previously. The first example is concerned with a family of cubic functions $y = ax^3 + bx^2 + cx$, shown in Fig. 2. The simulations were made under two different conditions. In the first case, there was a small number of classes ($I = 6$, indicated by thick frames in Fig. 2) with a large number of data vectors per class ($J = 200$), whereas in the second condition there was a large number of classes ($I = 126$, all functions in Fig. 2) with a small number of data vectors per class ($J = 8$). Each dataset $D_i = \{(x_{i,j}, y_{i,j})\}$ was sampled randomly, with the probability density function of $p(x)$ distributed uniformly between $[-1, +1]$. In addition, Gaussian white noise was added to $\{y_{i,j}\}$ ($\sigma_{\text{noise}} = 0.04$).

Fig. 3 shows some of the datasets used in the simulations. Fig. 3 (a), (b), and (c) are the datasets used in the first case, while (d) to (f) were used for the second case. In the first case, it is easy to estimate individual functions from the given datasets. Under this condition, the mnSOM is expected to interpolate between given functions. In contrast, it is hard to estimate individual functions under the second condition due to the deficiency of the sample data. In this case, the mnSOM is expected to estimate these functions while making neighbor classes as clues for the estimation. The MLP module has three layers with one input, eight hidden, and one output units. The neighborhood function was reduced from σ_0 to σ_∞ exponentially, i.e., $\sigma(t) = \sigma_\infty + (\sigma_0 - \sigma_\infty)\exp[-t/\tau]$. Other details are presented in Table 1.

Fig.2 about here.
Fig.3 about here.
Table 1 about here.

Fig. 4 (a) and (b) show the results of the first and the second case respectively. The curve depicted in each box represents the function acquired by the corresponding module after training. In both cases, the mnSOM generated similar maps. The modules on the diagonal corners showed opposite functions, and the modules located in the center of the map showed flat functions. The modules indicated by thick frames in Fig. 4 (a) are the BMMs of the given classes. All other functions acquired by the rest of the modules were interpolated by the mnSOM so as to make a continuous map. In the second case (Fig. 4 (b)), the mnSOM also succeeded in generating a map of cubic functions, despite the small number of data vectors. The validity of these maps generated by the mnSOM is discussed later.

Fig.4(a) and Fig.4 (b) about here.

3.2 Map of meteorological dynamics

The second set of material is a meteorological dataset available in a bulletin published by the Meteorological Agency of Japan. The dataset consists of daily records of weather attributes, such as atmospheric pressure, temperature, humidity, and sunlight hours, during a period of 100 days of the year 2000 at 20 cities in the Kyushu area of western Japan. The data shown in Fig. 5 were recorded at cities ‘B’ and ‘m’. Such time-series data of 10 of the 20 cities (A–J in Fig. 7) were given to the mnSOM for training, whereas the data for the remaining 10 cities (k–t in Fig. 7) were used for testing.

Fig.5 about here.

The module architecture is shown in Fig. 6. The input data was a 12-dimensional vector that consisted of the weather data of three consecutive days, and the desired output was the weather of the fourth day. Prior to the simulation, each time series record was normalized so that the DC component was removed. Thus, the MLP module was expected to learn the weather dynamics fluctuating around the average. In addition to the MLP module, there was a four-dimensional static vector, which was expected to learn the average level, i.e., the DC componet. For this reason, this architecture was a hybrid of a conventional SOM and an mnSOM. Other details are shown in Table 2.

Fig.6 about here.
Table 2 about here.

During the learning phase, training data of the 10 cities were given to the mnSOM. When learning had succeeded, the training was terminated. Then the data from the test cities were given to the mnSOM, and the BMMs of those cities were identified. Fig. 5 shows the predicted time courses (indicated by dashed lines) of a training and a test city (cities B and m in Fig. 7, respectively). The BMMs predicted the weather of the test cities as well as the training ones. The map of the weather dynamics generated by the mnSOM is shown in Fig. 8. The labeled modules represent the BMMs determined by the training (A–J) and the test (k–t) cities of Fig. 7. In the feature map, all cities for testing were allocated to appropriate positions, in such a way as to interpolate between the BMMs of the training cities. Thus, the mnSOM succeeded in generating “*a weather map of the Kyushu area*,” preserving the topology of the geographical map (Fig. 8).

In this simulation, the mnSOM played three roles: (1) it identified the weather dynamics of the cities by training, (2) it estimated the weather dynamics of “intermediate cities,” i.e., tested but not trained cities, (3) it generated a “*self-organizing weather map*” that reflected similarities in geo-meteorological dynamics. It should be emphasized here that the map organized by the mnSOM is no longer a static map, as the mnSOM organized an assembly of weather forecast systems, each of which is specialized to a particular city in the area.

Fig.7 about here.
Fig.8 about here.

4 Discussion

4.1 Relations between mnSOM and SOM

As mentioned in the theory section, the algorithm of mnSOM is not merely a modification of a conventional SOM, but rather it is a generalization that absorbs the conventional one. Thus, an mnSOM not only inherits many properties from a conventional SOM, but it also adds several new original properties. Therefore, it is important to confirm what the similarities and differences are between an SOM and an mnSOM. For example, there is the question of whether the mnSOM inherits the statistical properties of the conventional SOM.

To this end, we shall re-visit a case of cubic functions. In the simulation, the mnSOM generated a feature map of a cubic function family from a set of input-output data. The data were sampled randomly from the given functions, and they were presented to the mnSOM in random order. Practically speaking, this learning style is very natural for real problems, such as the case of weather data. However, in an artificial situation, such as with a case of cubic functions, it is also possible to vectorize the function shapes since the experimenter knows those functions in advance (of course such information is not given to the mnSOM at all). This means that the feature map of the cubic function family can be generated by the conventional SOM as well. One question then arises: is there any difference between the features map generated by the mnSOM and that generated by the conventional SOM? To answer this question, we consider a case in which an orthonormal functional expansion is employed for vectorization. Thus, let $\{\phi_n(\cdot)\}$ be a set of orthonormal functions. Then the function $f(\cdot)$ is transformed to a coefficient vector $\boldsymbol{a} = (a_0, \cdots, a_n)$, where

$$f(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_n\phi_n(x). \tag{17}$$

The terms higher than the nth order are assumed to be negligible. Under this condition, the distance in function space is identical to that in coefficient vector space. Thus,

$$\begin{aligned} L_f^2(f_i, g^k) &= (a_{i,0} - b^{k,0})^2 + \cdots + (a_{i,n} - b^{k,n})^2 & (18) \\ &= L_v^2(\boldsymbol{a}_i, \boldsymbol{b}^k). & (19) \end{aligned}$$

Here, L_f^2 and L_v^2 are the square distances in function and vector space, and $\boldsymbol{a}_i$ and $\boldsymbol{b}^k$ denote the coefficient vectors of the given *i*th function and the *k*th module of mnSOM, respectively. In this situation, mnSOM and SOM should produce the same result since the learning algorithm, i.e., (1)–(9), becomes identical for both types of SOM. That is, the mnSOM and the conventional SOM share the same properties of the feature map. This fact means that a great amount of previous work by conventional SOMs can be imported to MLP-mnSOMs.

In the case of the cubic functions, $p(x)$ was set to be uniform between $[-1, +1]$. Therefore, normalized Legendre polynomials are best suited for the orthonormal expansion in this case. Thus, the *n*th orthonormal function $\phi_n(x)$ is described by the *n*th Legendre function $P_n(x)$ as

$$\phi_n(x) = \sqrt{n + 1/2}\, P_n(x). \tag{20}$$

By using this set of orthonormal functions, the set of six functions indicated by the thick frames in Fig. 2 were expanded to their corresponding coefficient vectors $\{\boldsymbol{a}_i\} = \{(a_{i,1}, a_{i,2}, a_{i,3})\}$. The feature map generated by a conventional SOM for $\{\boldsymbol{a}_i\}$ is shown in Fig. 9 (a). The figure shows the position of each reference vector in the coefficient vector space. The corresponding map generated by the mnSOM are shown in Fig. 9 (b) and (c). Fig. 9 (b) shows the map of the global minimum point of each MLP module evaluated by (8). In simple terms, it can be regarded as a 'map of teacher signals' generated in the mnSOM. Meanwhile, Fig. 9 (c) is a map of the actual functions acquired by the MLP-modules of the mnSOM. Thus, Fig. 9 (c) was obtained by making Legendre expansions of the functions shown in Fig. 4 (a). The map generated by a conventional SOM and that of the teacher signal of an mnSOM (Fig. 9 (a) and (b)) were identical as it is assured by the theory. The map of the actual functions of the MLP-modules (Fig. 9 (c)) was also fairly equal to the other two maps, while small errors remained that were presumably caused by the additive noise and by the limitation of approximation ability of the MLPs. The mean square error of outputs between the ideal result obtained by SOM (Fig. 9 (a)) and the actual mnSOM result with additive noise (Fig. 9 (c)) was 1.08×10^{-3}. This error approximately corresponds to 3% of the output range, while the training data contains 4% noise. This result may encourage the users of mnSOMs because the theoretical properties of the MLP-mnSOM have been assured by past works conducted on the conventional SOM (e.g., Cottrell, Fort, & Pages, 1998; Dersch, & Tavan, 1995). This fact also means that all advantages and disadvantages of the conventional SOM are inherited to mnSOM, such as the difficulty of defining the energy function (Erwin, Obermayer &

Shulten, 1992).

Finally, it is worth noting that such orthonormal expansion *before training* will not be possible in most practical cases, in which the mapping objects are unknown. Therefore, the conventional SOM is not available in such practical cases. On the other hand, the mnSOM can solve the simultaneous problem of estimating the unknown functions and generating their maps. This is another important advantage of mnSOM.

Fig.9 about here.

4.2 On designing modules

In this paper, only MLP-mnSOMs were used in the simulations, although we have tried other module types previously. For example, some preliminary results with recurrent networks such as Elman Net (Elman, 1991) and with autoassociative neural networks (Oja, 1991) have been reported (Furukawa, Tokunaga, Kaneko, Kimotsuki, & Yasui, 2004; Tokunaga, & Furukawa, 2005). Since these architectures are variations of MLP, the algorithms of the mnSOM with these modules are almost the same as that of the MLP-mnSOM. We have also developed a SOM-module-mnSOM (Furukawa, 2005a; Furukawa, 2005b). The algorithm of the SOM-module-mnSOM, which we call 'SOM^2,' looks quite different from that of the MLP-module, but the algorithm can also be derived from the generalized SOM algorithm described in this paper. The algorithm of SOM^2 will be described elsewhere.

One may then raise the question of how the algorithm of an mnSOM should be designed when a user employs a new module type. Since there are a number of existing architectures and algorithms, developing a generalized protocol seems like a hopeless task. Fortunately, however, distance measures or energy functions are often accompanied by the module architectures. Therefore what users have to do is to replace the energy function by (12). To minimize the energy function, it would be also possible to use the evolutionary algorithm as well as the gradient method (Villmann, 2002).

On designing the mnSOM module and its learning algorithm, mnSOM users need to know the following two points. First, users should be mindful of the consistency between their purpose, the module architecture, and the algorithm. To better explain this point, let us consider a case in which two mnSOM users are trying to make a map from a set of time series data. Suppose that one user tries to make a map of the waveforms, and the other wants to get a map of the dynamics. In such a case, the former user needs to define the distance measure in the signal space, whereas the latter user should define it in the system space. Furthermore, the selection of the module architecture and algorithm should be consistent with their purpose. The users need to be careful because a small change in the system dynamics causes a big difference in the waveforms. The mnSOM will generate a sort of map even if the distance and algorithm are not defined appropriately. However, there is no assurance that the generated map will fit the user's purpose in such cases. Therefore, the consistency of the definitions and the user's purpose should be checked in all cases. One of the best ways is to check whether the algorithm generates desired 'intermediate' objects for user's purpose. By giving two objects A and B to a single module with learning weights ψ_A and ψ_B ($\psi_A + \psi_B = 1$), the user can see whether the module represents an intermediate object between A and B. Only user knows what is the desired middle point, user should therefore accept these responsibilities as a compensation for the freedom given by the mnSOM.

The second point is that users are recommended to check the module parameters before it is built into an mnSOM. Thus, it is better to check the ability of one module with one class of data in advance since an mnSOM usually has a lot of parameters that require adjustment. Once the user finds a good parameter set for a single module, then the remaining parameters are almost the same as those for the conventional SOM. Therefore, users will not feel difficulties in adjusting parameters if they employ this two-stage adjustment.

In this paper, MLP-module cases were introduced as representative cases since MLP is one of the most popular architectures among neural networks and the principle of gradient descendent algorithms is also a common method. For practical tasks, however, other architectures such as RBF networks can be expected to produce better results.

5 Conclusion

An extension of SOM using a modular network structure has been presented in this paper. Our mnSOM is a natural extension of a conventional SOM stemming from Kohonen, and we describe our mnSOM in a more general framework. One of the advantages of mnSOMs is that every module in an mnSOM has the capability of information processing. Thus, an mnSOM forms a dynamical map that consists of an assembly of functional modules. Therefore, one might draw analogies between the mnSOM and the functional map of our cortex, which continuously processes the data flowing into it. The mnSOM also provides a way of fusing a supervised and an unsupervised learning algorithm. For example, an MLP-mnSOM is trained by a supervised learning algorithm, i.e., the backpropagation at the MLP module level, while the upper SOM level is described in an unsupervised manner. Thus, the mnSOM provides an ‘unsupervised supervised learning’ method. These abilities have been newly added by mnSOM, and therefore, the mnSOM is not an improvement or a modification of the conventional SOM at all, but rather it has extended and widened the ability of the conventional SOM further.

References

[1] Abe, T., Kanaya, S., Kinouchi, M., Ichiba, Y., Kozuki, T., & Ikemura, T. (2003). Informatics for unvailing hidden genome signatures. *Genome Research*, **13**(4), 693–702.

[2] Cottrell, M., Fort, J. C., & Pagès, G. (1998). Theoretical aspects of the SOM algorithm. *Neurocomputing*, **21**, 119–138.

[3] Deboeck, G., & Kohonen, T. (1998). *Visual Explorations in Finance with Self-Organizing Maps*. London: Springer.

[4] Dersch, D., & Tavan, P. (1995). Asymptotic level density in topological feature maps. *IEEE Transactions on Neural Networks*, **6**(1), 230–236.

[5] Erwin, E., Obermayer, K. & Schulten, K. (1992). Self-organizing maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, **67**, 47–55.

[6] Elman, J.L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, **7**(2–3), 195–225.

[7] Furukawa, T., Tokunaga, K., Kaneko, S., Kimotsuki, K., & Yasui, S. (2004). Generalized self-organizing maps (mnSOM) for dealing with dynamical systems. *Proceedings of 2004 International Symposium on Nonliner Theory and its Applications* (pp. 231–234). Fukuoka, Japan.

[8] Furukawa, T. (2005a). SOM^2 as "SOM of SOMs." *Proceedings of 5th Workshop On Self-Organizing Maps* (pp. 545–552). Paris, France.

[9] Furukawa, T. (2005b). SOM of SOMs: Self-organizing map which maps a group of self-organizing maps. *Lecture Notes in Computer Science*, 3696, 391–396.

[10] Kohonen, T. (1993). Generalization of the self-organizing map. *Proceedings of International Joint Conference on Neural Networks* (pp. I-457–462). Nagoya, Japan.

[11] Kohonen, T. (2001). *Self-organizing maps* (3rd ed). Berlin: Springer-Verlag.

[12] Kohonen, T., Kaski, S., & Lappalainen, H. (1997). Self-organized formation of various invariant-feature filters in the adaptive-subspace SOM. *Neural Computation*, **9**(6), 1321–1344.

[13] Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., & Saarela, A. (2000). Self-organization of a massive text document collection. *IEEE Transactions on Neural Networks*, **11**(3), 574–585.

[14] Martinetz, T., Ritter, H., & Schulten, K. (1990). Three-dimensional neural net for learning visuo-motor coordination of a robot arm. *IEEE Transactions on Neural Networks*, **1**(1), 131–136.

[15] Oja, E. (1991). Data compression, feature extraction, and autoassociation in feedforward neural networks. *Artificial neural networks* (pp.737–745). Amsterdam: Elsevier.

[16] Tokunaga, K., & Furukawa, T. (2005). Nonlinear ASSOM constituted of autoassociative neural modules. *Proceedings of 5th Workshop On Self-Organizing Maps* (pp. 637–644). Paris, France.

[17] Villmann, T. (2002). Evolutionary algorithms using a neural network like migration scheme. *Integrated Computer-Aided Engineering*, **9**(7), 25–35.

[18] Walter, J., & Ritter, H. (1996). Rapid learning with parametrized self-organizing maps. *Neurocomputing*, **12**, 131–153.

Figure Legends

Fig. 1 The architecture of an mnSOM with MLP-modules.

Fig. 2 A family of cubic functions used in the first simulation. The six functions indicated with thick frames were used in the first case, while all functions were used in the second case.

Fig. 3 Examples of datasets used in the first simulation. (a)–(c) Datasets for the first case. The number of data points is 200 for each class. (d)–(f) Datasets for the second case. The number of data points is 8 for each class.

Fig. 4 Two maps of cubic functions generated by mnSOMs. (a) The map of the first case with 6 training functions. (b) The map of the second case with 126 training functions.

Fig. 5 Examples of time series data of the four weather attributes at cities ‘B’ and ‘m’ in Fig. 7. Solid and dashed lines represent the actual and the predicted weather.

Fig. 6 The module architecture of the mnSOM for learning the meteorological dynamics.

Fig. 7 A map of Kyushu area in Japan. The weather data of cities A–J were used for training, while the data of cities k–t were used for testing.

Fig. 8 A weather map of Kyushu area generated by the mnSOM.

Fig. 9 The feature map of cubic functions plotted in the coefficient parameter space of Legendre expansion. (a) A map generated by a conventional SOM. The labels 1–6 represent the BMMs of the training fuctions indicated with thick frames in Fig. 2. (b) A ‘map of teacher signals’ generated in the mnSOM. (c) A map of the actual functions acquired by the MLP-modules in the mnSOM.

Table 1 Parameters used in the simulations with cubic functions.

Table 2 Parameters used in the simulation with the meteorological data.

Table 1.

Parameters of MLP module	
Number of units	
Input layer	1
Hidden layer	8
Output layer	1
Learning constant η	0.05
Parameters of mnSOM	
Map size K	100 (10×10)
Neighborhood function size	
σ_0	10.0
σ_∞ (case 1)	2.0
σ_∞ (case 2)	1.0
τ	300
Safety factor β	0.9

Table 2.

Parameters of MLP module	
Number of units	
Input layer	12
Hidden layer	5
Output layer	4
Bias part	4
Learning constant η	0.01
Parameters of mnSOM	
Map size K	225 (15×15)
Neighborhood function size	
σ_0	15.0
σ_∞	2.0
τ	300
Safety factor β	0.9

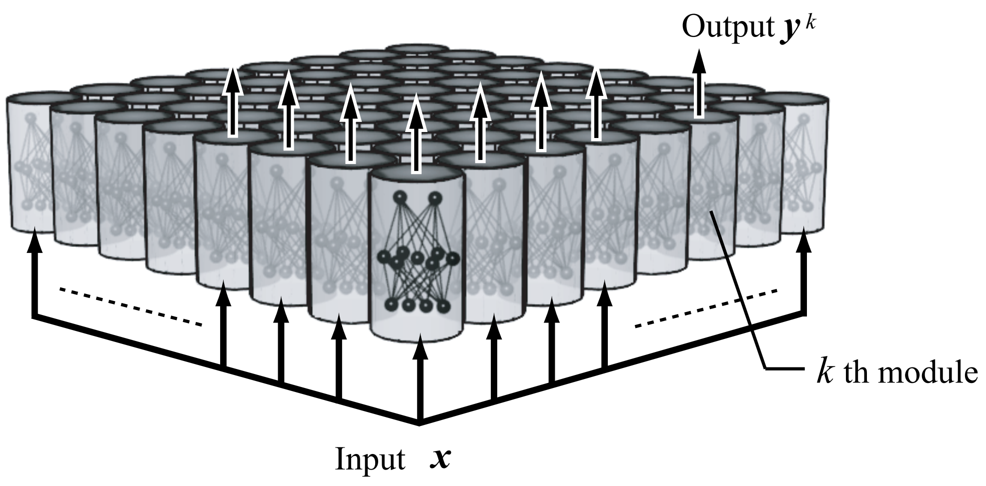


Fig. 1

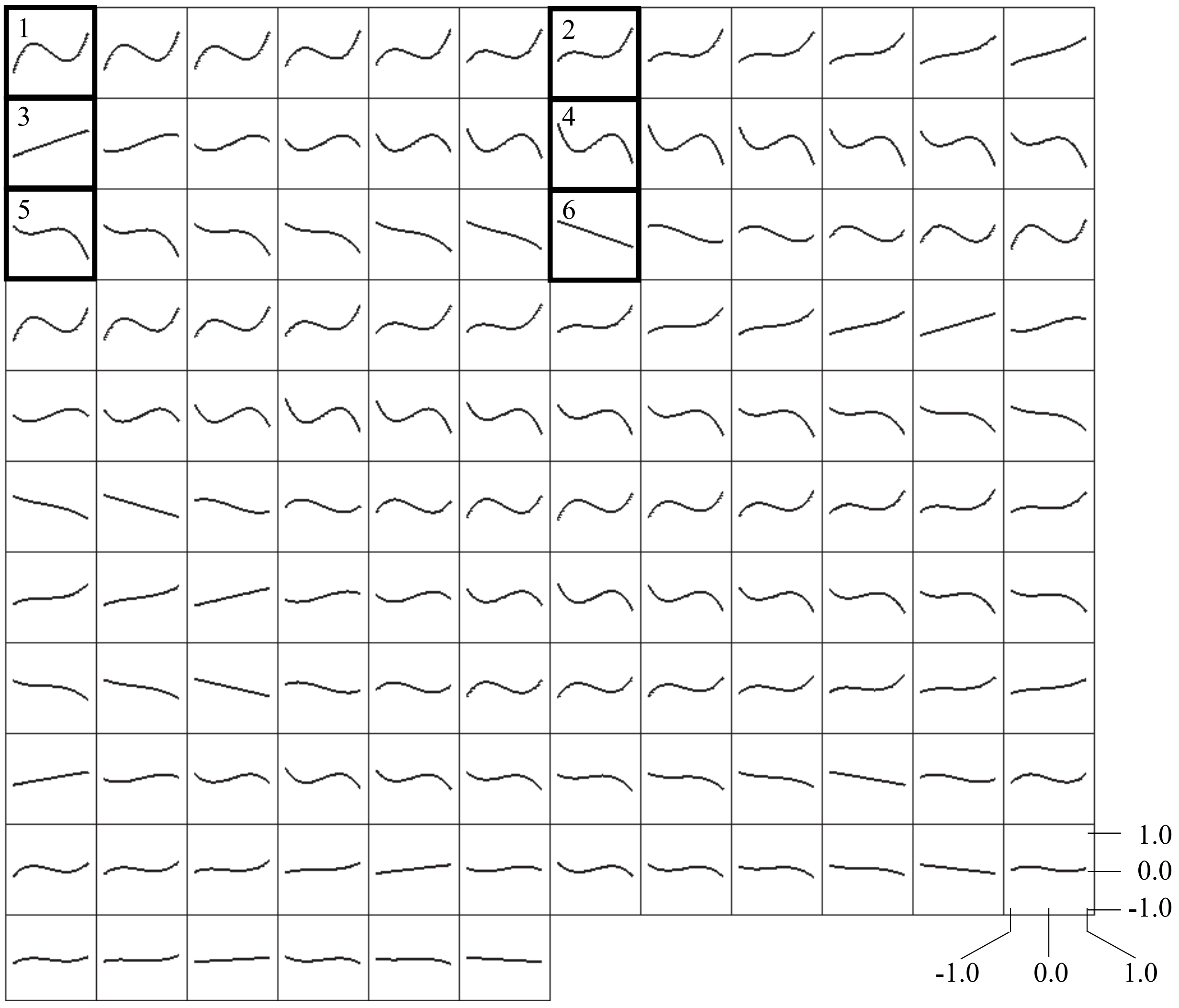


Fig. 2

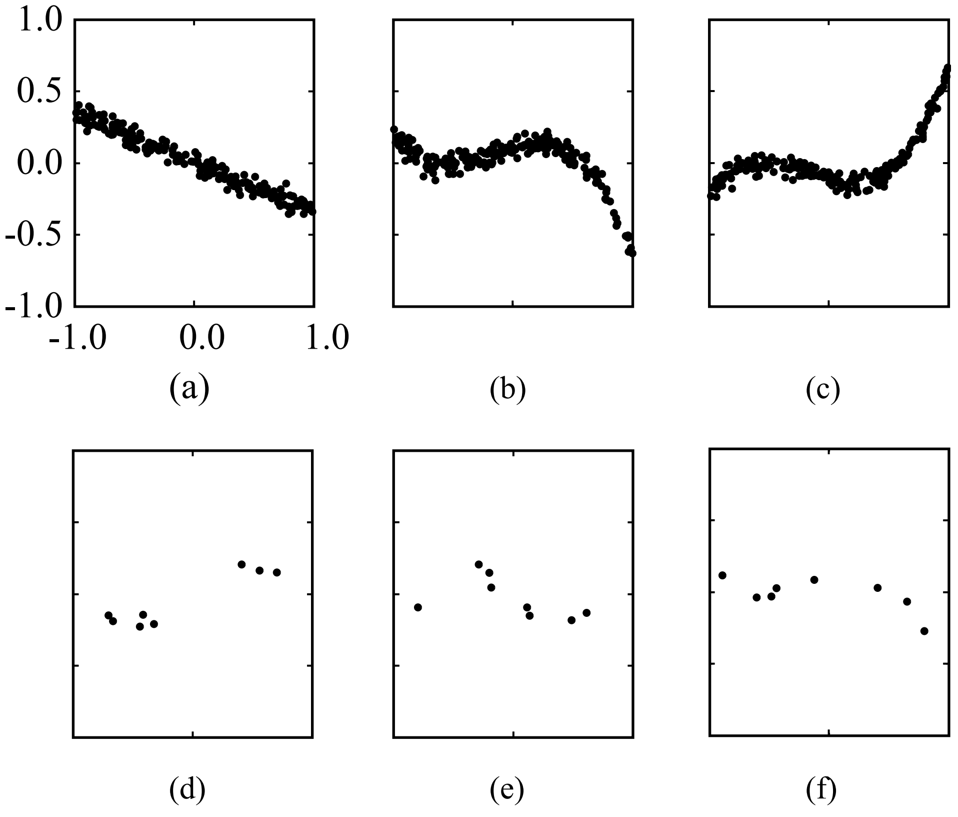


Fig. 3

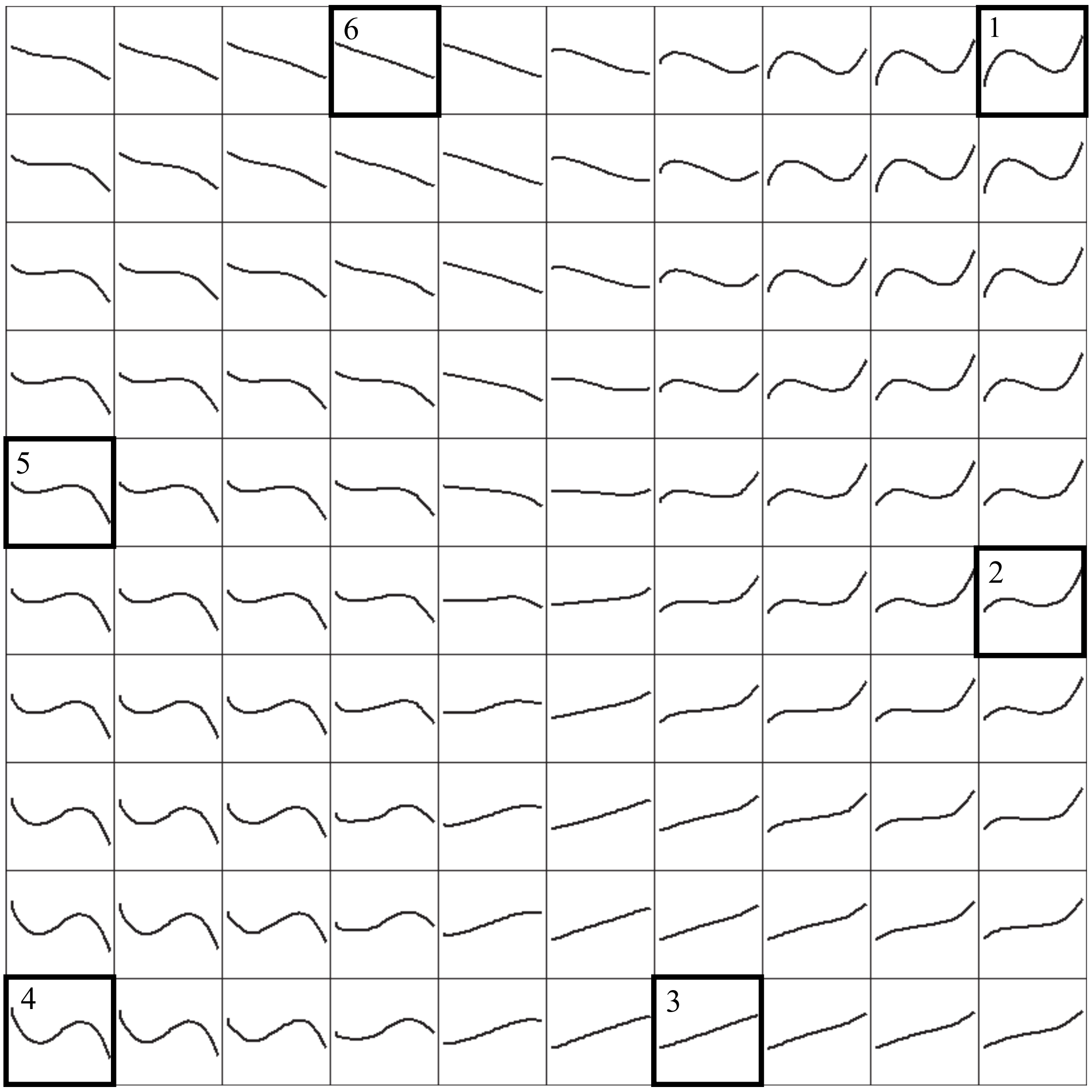


Fig. 4 (a)

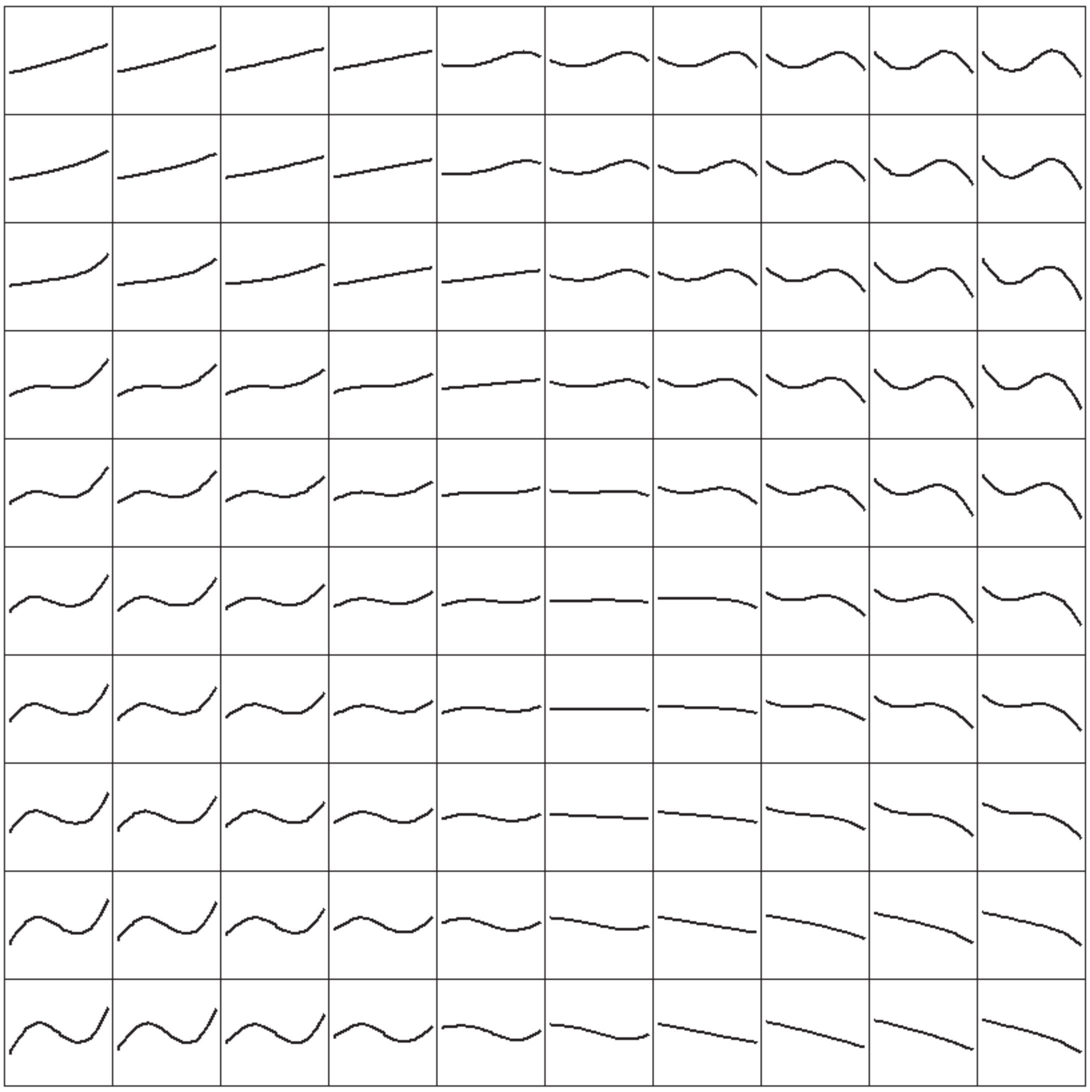
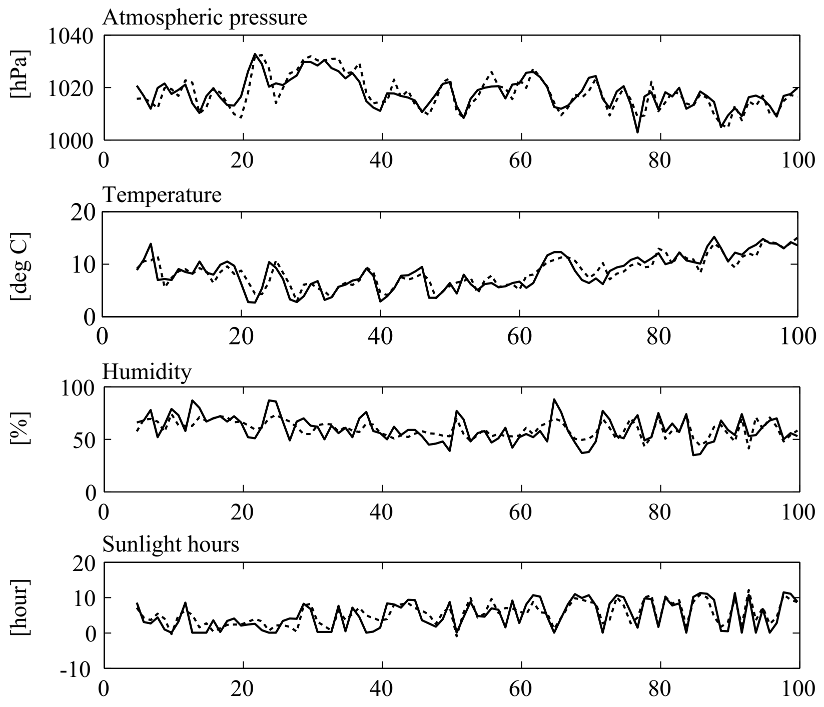
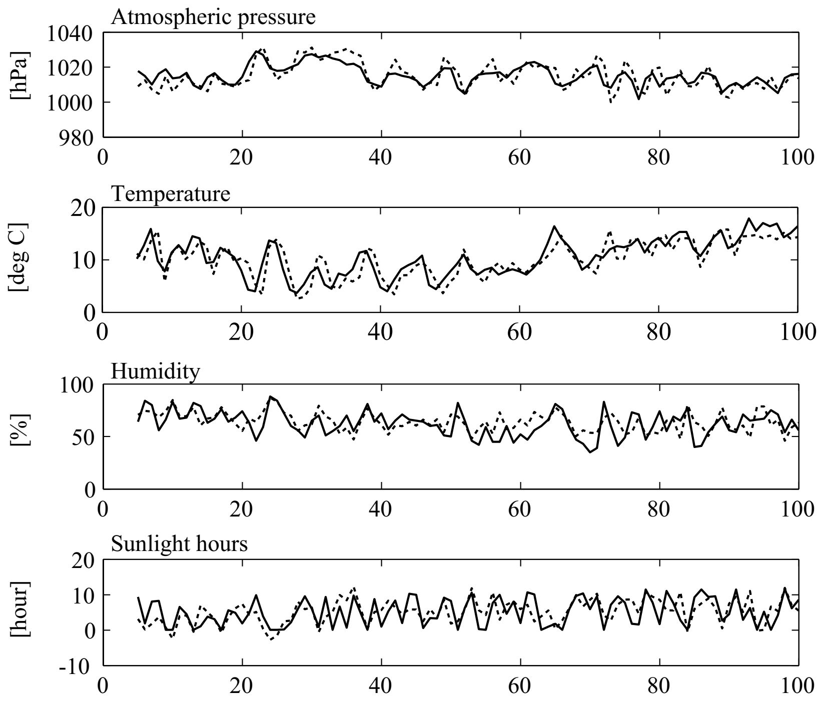


Fig. 4 (b)



City B



City m

Fig. 5

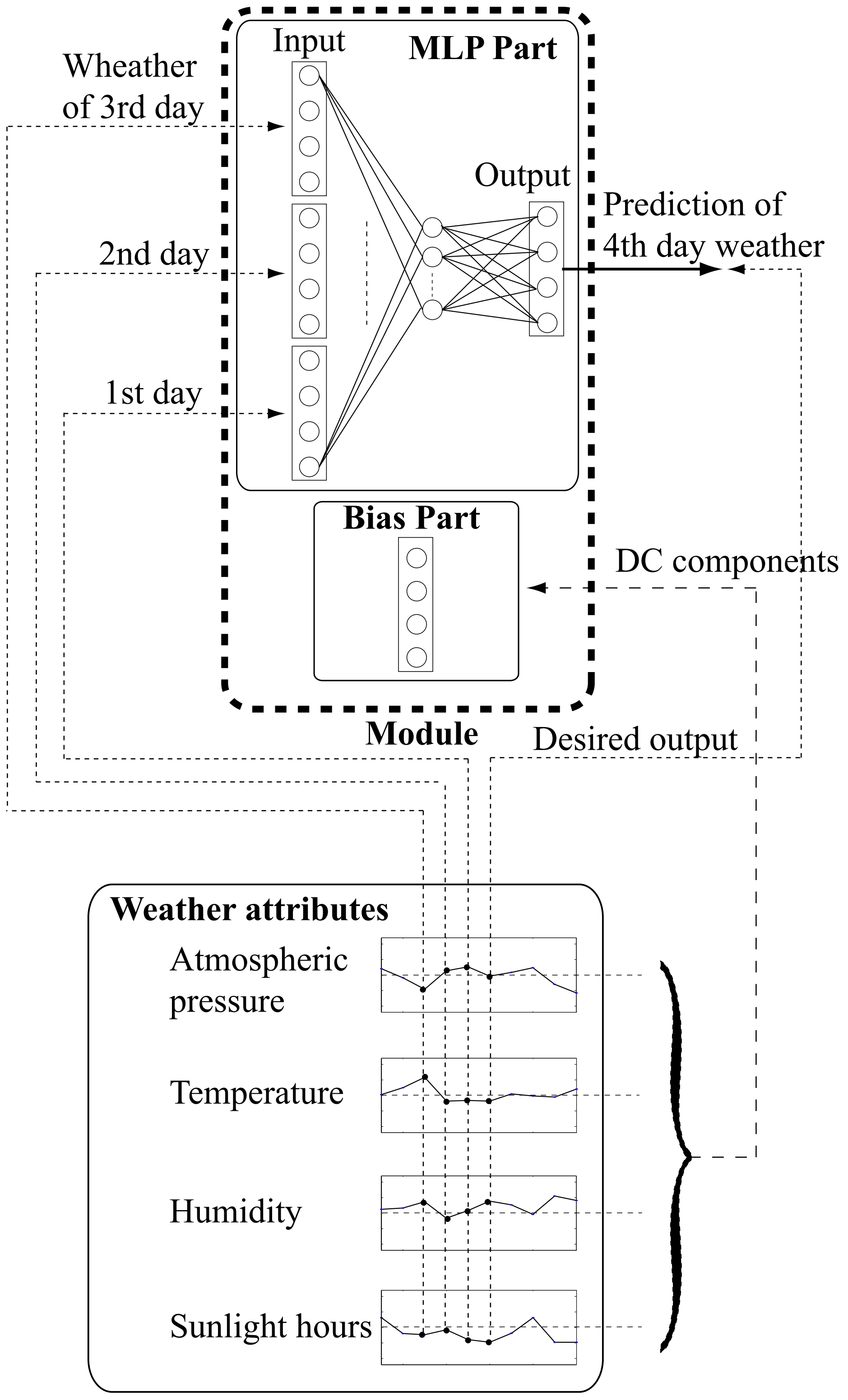

Wheather
of 3rd day
2nd day
1st day
Input
MLP Part
Output
Prediction of
4th day weather
Bias Part
DC components
Module
Desired output
Weather attributes
Atmospheric
pressure
Temperature
Humidity
Sunlight hours

Fig. 6

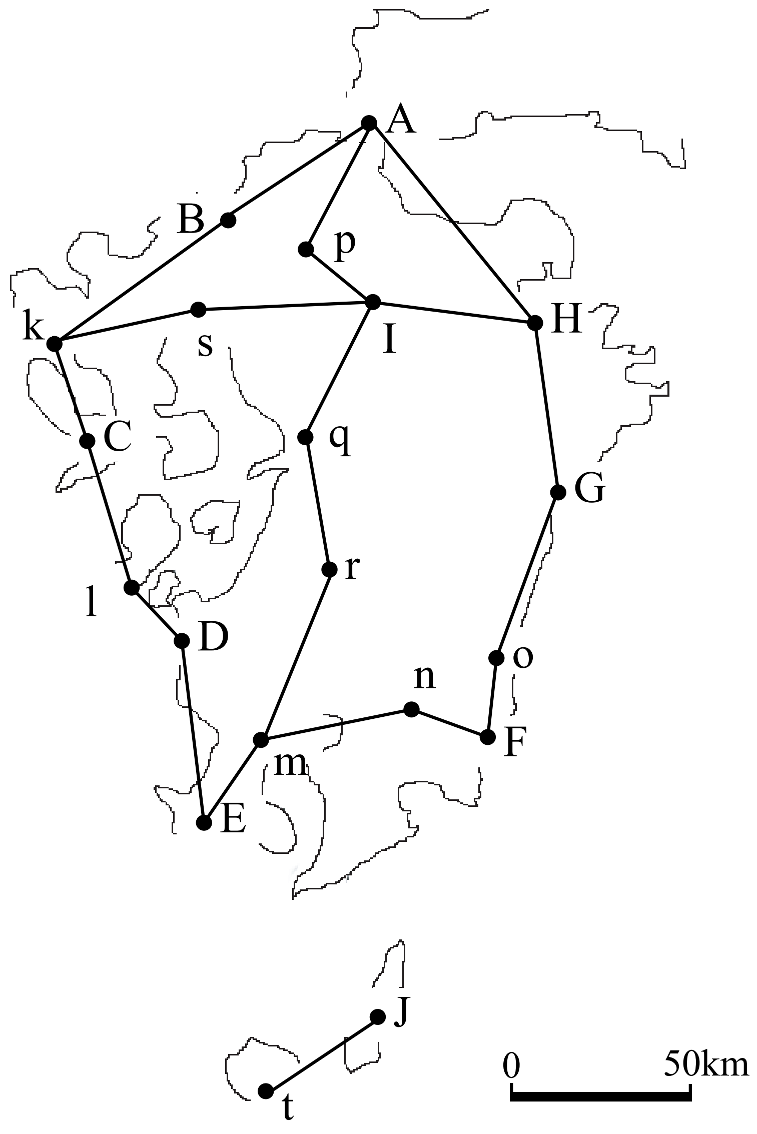


Fig. 7

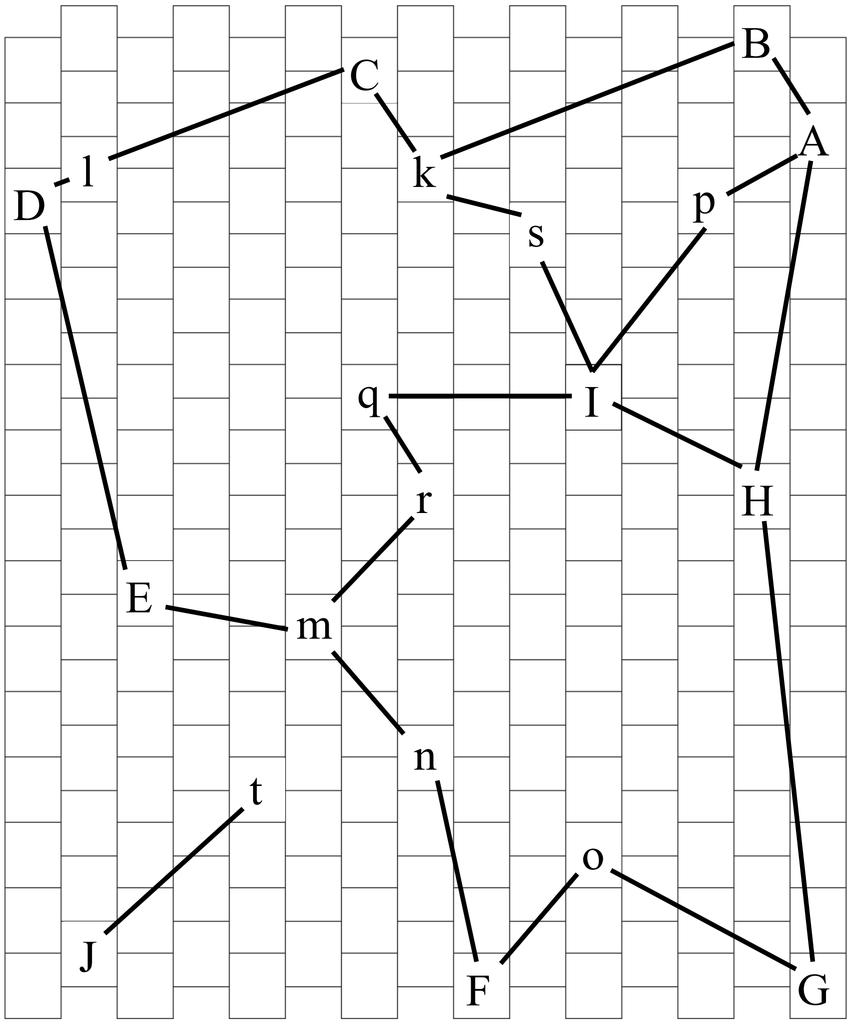
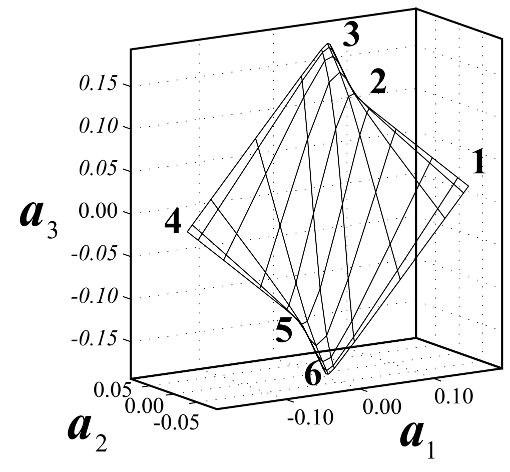
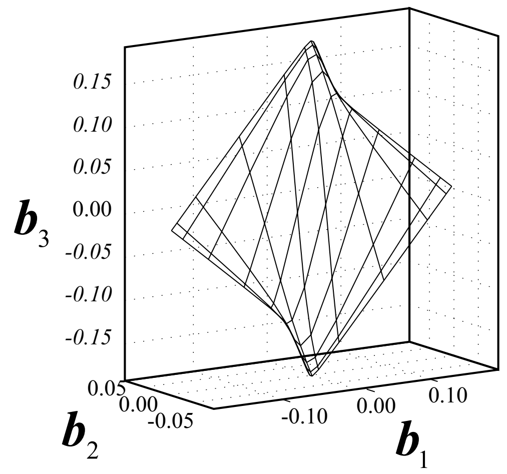


Fig. 8



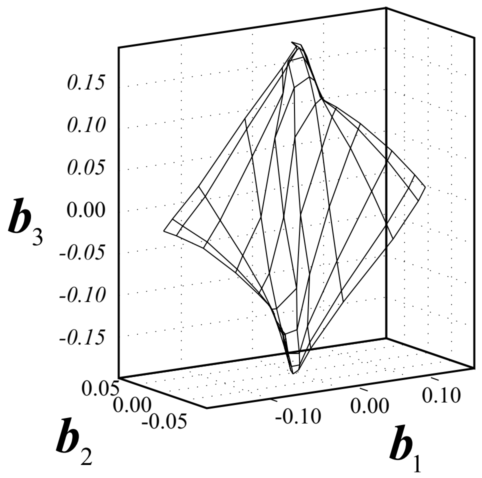
3
2
1
4
5
6
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
a_3
0.05
0.00
-0.05
a_2
-0.10
0.00
0.10
a_1

(a)



0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
b_3
0.05
0.00
-0.05
b_2
-0.10
0.00
0.10
b_1

(b)



0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
b_3
0.05
0.00
-0.05
b_2
-0.10
0.00
0.10
b_1

(c)

Fig. 9