# 1

# Neural Forecasting Systems

Takashi Kuremoto, Masanao Obayashi and Kunikazu Kobayashi
*Yamaguchi University*
*Japan*

## 1. Introduction

Artificial neural network models (NN) have been widely adopted on the field of time series forecasting in the last two decades. As a kind of soft-computing method, neural forecasting systems can be built more easily because of their learning algorithms than traditional linear or nonlinear models which need to be constructed by advanced mathematic techniques and long process to find optimized parameters of models. The good ability of function approximation and strong performance of sample learning of NN have been known by using error back propagation learning algorithm (BP) with a feed forward multi-layer NN called multi-layer perceptron (MLP) (Rumelhart et. al, 1986), and after this mile stone of neural computing, there have been more than 5,000 publications on NN for forecasting (Crone & Nikolopoulos, 2007).

To simulate complex phenomenon, chaos models have been researched since the middle of last century (Lorenz, 1963; May, 1976). For NN models, the radial basis function network (RBFN) was employed on chaotic time series prediction in the early time (Casdagli, 1989). To design the structure of hidden-layer of RBFN, a cross-validated subspace method is proposed, and the system was applied to predict noisy chaotic time series (Leung & Wang, 2001). A two-layered feed-forward NN, which has its all hidden units with hyperbolic tangent activation function and the final output unit with linear function, gave a high accuracy of prediction for the Lorenz system, Henon and Logistic map (Oliveira et. al, 2000).

To real data of time series, NN and advanced NN models (Zhang, 2003) are reported to provide more accurate forecasting results comparing with traditional statistical model (i.e. the autoregressive integrated moving average (ARIMA)(Box & Jankins, 1976)), and the performances of different NNs for financial time series are confirmed by Kodogiannis & Lolis (Kodogiannis & Lolis, 2002). Furthermore, using benchmark data, several time series forecasting competitions have been held in the past decades, many kinds of NN methods showed their powerful ability of prediction versus other new techniques, e.g. vector quantization, fuzzy logic, Bayesian methods, Kalman filter or other filtering techniques, support vector machine, etc (Lendasse et. al, 2007; Crone & Nikolopoulos, 2007).

Meanwhile, reinforcement learning (RL), a kind of goal-directed learning, has been generally applied in control theory, autonomous system, and other fields of intelligent computation (Sutton & Barto, 1998). When the environment of an agent belongs to Markov decision process (MDP) or the Partially Observable Markov Decision Processes (POMDP), behaviours of exploring let the agent obtain reward or punishment from the environment, and the policy of action then is modified to adapt to acquire more reward. When prediction



Source: Reinforcement Learning: Theory and Applications, Book edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer
ISBN 978-3-902613-14-1, pp.424, January 2008, I-Tech Education and Publishing, Vienna, Austria

error for a time series is considered as reward or punishment from the environment, one can use RL to train predictors constructed by neural networks.
In this chapter, two kinds of neural forecasting systems using RL are introduced in detail: a self-organizing fuzzy neural network (SOFNN) (Kuremoto et al., 2003) and a multi-layer perceptron (MLP) predictor (Kuremoto et al., 2005). The results of experiments using Lorenz chaos showed the efficiency of the method comparing with the results by a conventional learning method (BP).

## 2. Architecture of neural forecasting system

The flow chart of neural forecasting processing is generally used by which in Fig. 1. The *t*th step time series data $y(t)$ can be embedded into a new *n*-dimensional space $\mathbf{x}(t)$ according to Takens Theorem (Takens, 1981). Eq. (1) shows the detail of reconstructed vector space which serves input layer of NN, here $\tau$ is an arbitrary delay. An example of 3-dimensional reconstruction is shown in Fig. 2. The output layer of neural forecasting systems is usually with one neuron whose output $\hat{y}(t+1)$ equals prediction result.

$$\begin{aligned}\mathbf{x}(t) &= (x_1(t), x_2(t), \cdots, x_n(t)) \\ &= (y(t), y(t-\tau), \cdots, y(t-(n-1)\tau)\end{aligned} \tag{1}$$

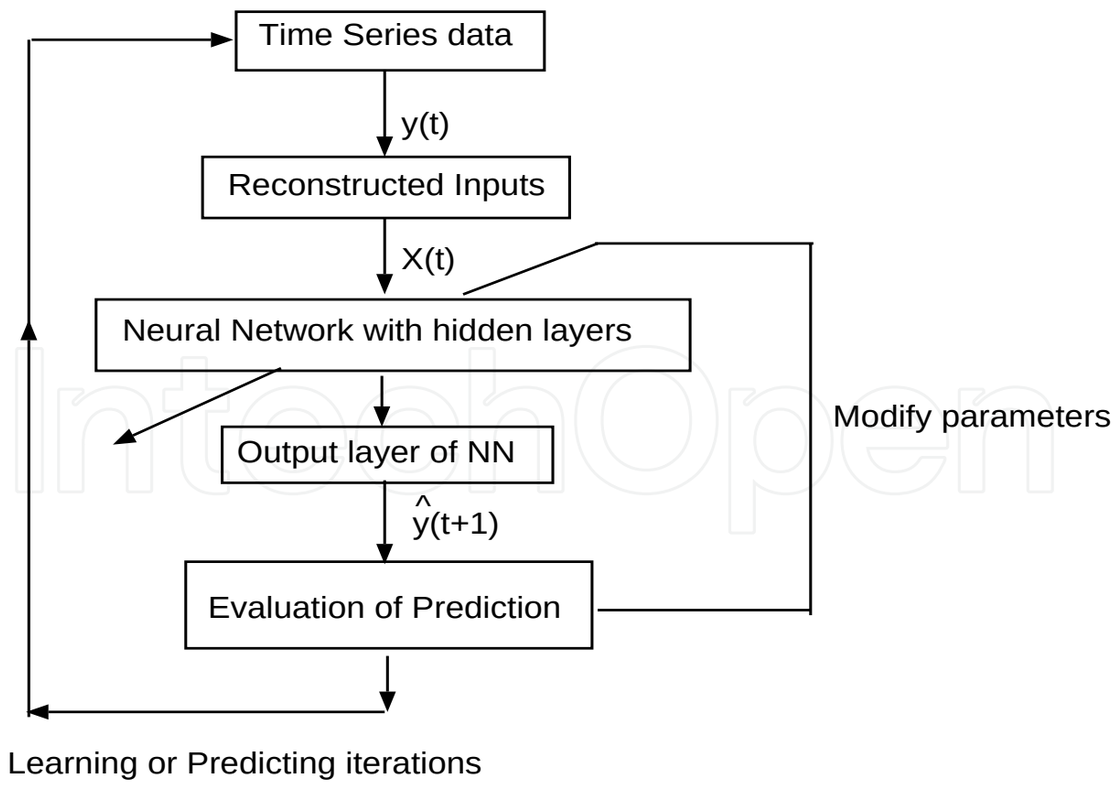


Fig. 1. Flow chart of neural forecasting methods.

There are various architectures of NN models, including MLP, RBFN, recurrent neural network (RNN), autoregressive recurrent neural network (ARNN), neuro-fuzzy hybrid network, ARIMA-NN hybrid model, SOFNN, and so on. The training rules of NNs are also very different not only well-known methods, i.e., BP, orthogonal least squares (OLS), fuzzy inference, but also evolutional computation, i.e., genetic algorithm (GA), particle swarm optimization (PSO), genetic programming (GP), RL, and so on.

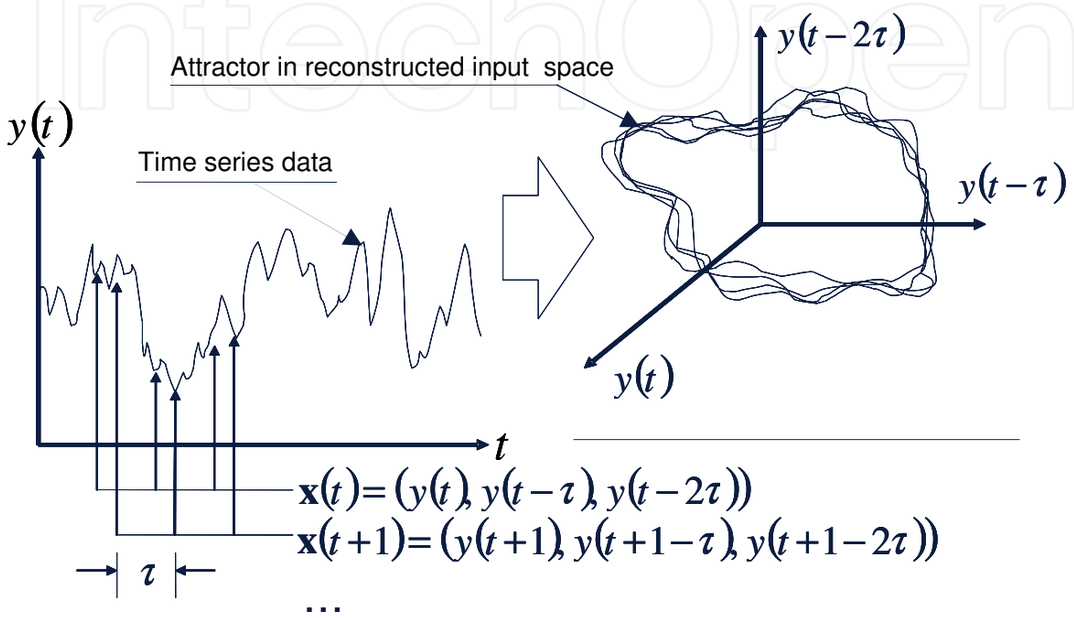


Fig. 2. Embedding a time series into a 3-dimensional space.

### 2.1 MLP with BP

MLP, a feed-forward multi-layer network, is one of the most famous classical neural forecasting systems whose structure is shown in Fig. 3. BP is commonly used as its learning rule, and the system performs fine efficiency in the function approximation and nonlinear prediction.

For the hidden layer, let the number of neurons is *K*, the output of neuron k is $H_k$, then the output of MLP is obtained by Eq. (2) and Eq. (3).

$$\hat{y}(t+1) = f(\sum_{k=1}^{K} w_{yk} H_k) \tag{2}$$

$$H_k = f(\sum_{i=1}^{n} w_{ki} x_i(t)) \tag{3}$$

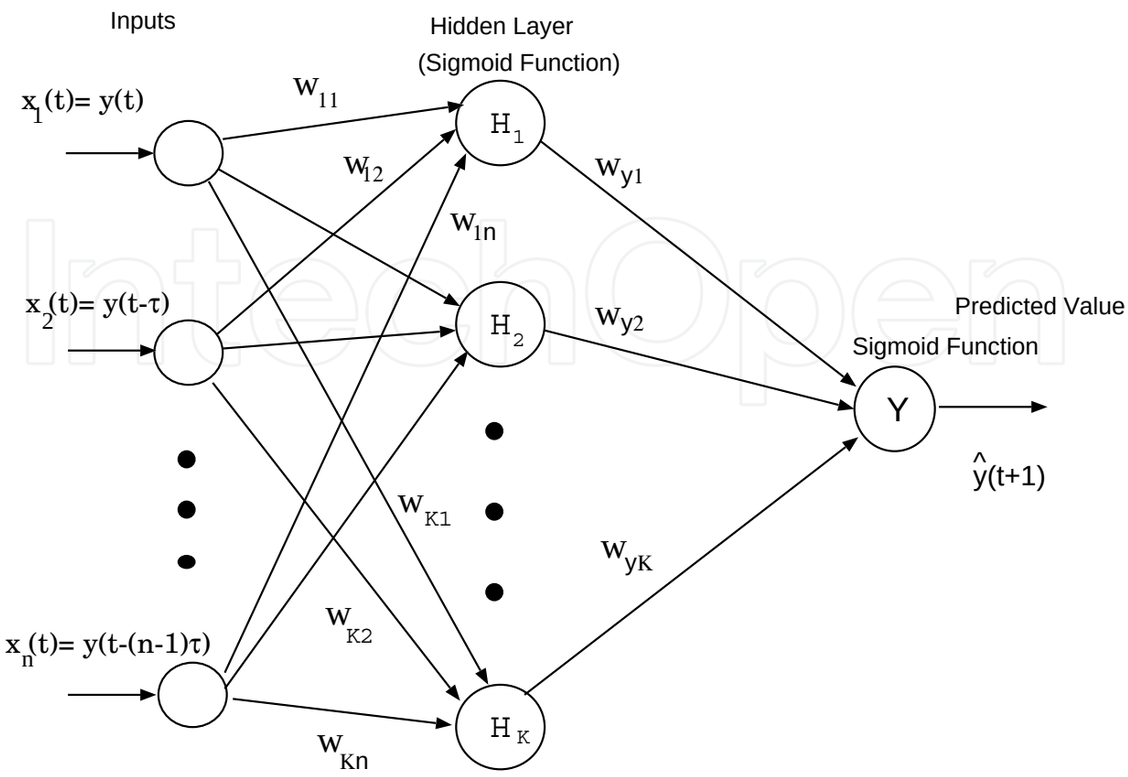


Fig. 3. A MLP with *n* input neurons, one hidden layer, and one neuron in output layer using BP training algorithm.

Here $w_{yk}$, $w_{ki}$ represent the connection of *k*th hidden neuron with output neuron and input neurons, respectively. Activation function $f(u)$ is a sigmoid function (or hyperblolic tangent function) given by Eq. (4).

$$f(u) = \frac{1}{1 + \exp(-\beta u)} \tag{4}$$

Gradient parameter β is usually set to 1.0, and to correspond to $f(u)$, the scale of time series data should be adjusted to (0.0, 1.0).
BP is a supervised learning algorithm, using sample data trains NN providing more correct output data by modifying all of connections between layers. Conventionally, the error function is given by the mean square error as Eq. (5).

$$E(W) = \frac{1}{S}\sum_{t=0}^{S-1}(y(t+1) - \hat{y}(t+1))^2 \tag{5}$$

Here $S$ is the size of train data set, $y(t+1)$ is the actual data in time series. The error is minimized by adjusting the weights according to Eq. (6), Eq. (7) and Eq. (2), Eq. (3).

$$W(w_{yk}, w_{ik})^{new} = \alpha W(w_{yk}, w_{ik})^{old} - \eta \Delta W(w_{yk}, w_{ik}) \tag{6}$$

$$\Delta W\ (w_{yk}, w_{ik}) = (\partial E / \partial w_{yk}, \partial E / \partial w_{ik}) \tag{7}$$

Here $\alpha$ is a discount parameter ($0.0 < \alpha \leq 1.0$), $\eta$ is the learning rate ($0.0 < \eta \leq 1.0$). The training iteration keeps to be executed until the error function converges enough.

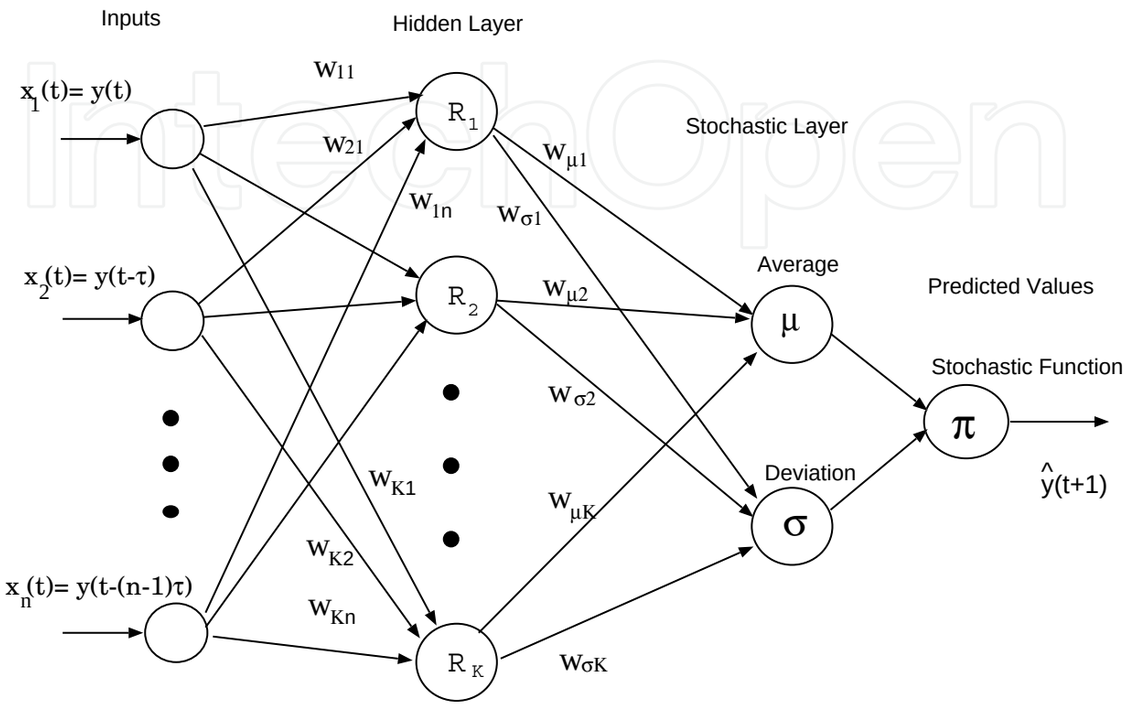


Fig. 4. A MLP with *n* input neurons, two hidden layers, and one neuron in output layer using RL training algorithm.

### 2.2 MLP with RL

One important feature of RL is its statistical action policy, which brings out exploration of adaptive solutions. Fig. 4 shows a MLP which output layer is designed by a neuron with Gaussian function. A hidden layer consists of variables of the distribution function is added. The activation function of units in each hidden layer is still sigmoid function (or hyperbolic tangent function) (Eq. (8)-(10)).

$$\mu = \frac{1}{1 + \exp(-\beta_1 \sum R_k w_{\mu k})} \tag{8}$$

$$\sigma = \frac{1}{1 + \exp(-\beta_2 \sum R_k w_{\sigma k})} \tag{9}$$

$$R_k = \frac{1}{1 + \exp(-\beta_3 \sum x_i(t) w_{ki})} \tag{10}$$

And the prediction value is given according to Eq. (11).

$$\pi(\hat{y}(t+1), \mathbf{w}, \mathbf{x}(t)) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{(\hat{y}(t+1)-\mu)^2}{2\sigma^2} \right\} \tag{11}$$

Here $\beta_1, \beta_2, \beta_3$ are gradient constants, $\mathbf{w}$ ($w_{\mu k}$, $w_{\sigma k}$, $w_{ki}$) represents the connection of $k$th hidden neuron with neuron μ,σ in statistical hidden layer and input neurons, respectively. The modification of $\mathbf{w}$ is calculated by RL algirthm which will be described in section 3.

### 2.3 SOFNN with RL

A neuro-fuzzy hybrid forecasting system, SOFNN, using RL training algorithm is shown in Fig. 5. A hidden layer consists of fuzzy membership functions $B_{ij}(x_i(t))$ is designed to categorize input data of each dimension in $\mathbf{x}(x_1(t), x_2(t),..., x_n(t))$, $t$ = 1, 2, ..., S (Eq. (12)). The fuzzy reference $\lambda_k$, which calculates the fitness for an input set $\mathbf{x}(t)$, is executed by fuzzy rules layer (Eq. 13).

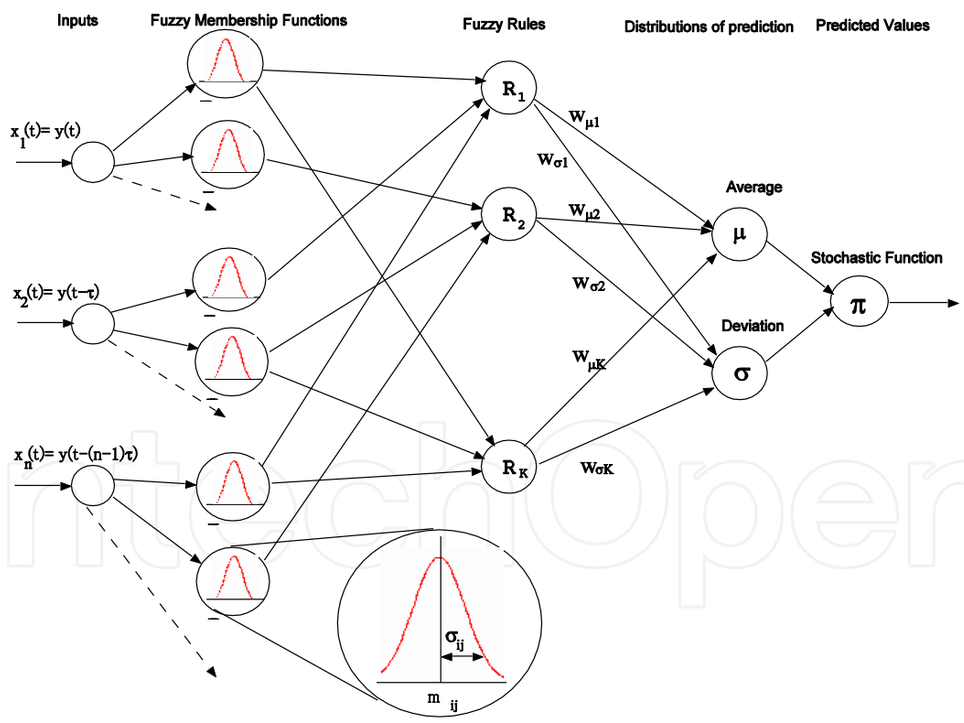


Fig. 5. A SOFNN with *n* input neurons, three hidden layers, and one neuron in output layer using RL training algorithm.

$$B_{ij}\left(x_i(t)\right) = \exp\left\{-\frac{\left(x_i(t) - m_{ij}\right)^2}{2\sigma_{ij}^{\ 2}}\right\} \tag{12}$$

$$\lambda_k\left(X(t)\right) = \prod_{i=1}^{n} B_{ic}\left(x_i(t)\right) \tag{13}$$

Where $i$ = 1, 2, ..., $n$, $j$ means the number of membership function which is 1 initially, $m_{ij}, \sigma_{ij}$ are the mean and standard deviation of $j$th membership function for input $x_i(t)$, $c$ means each of membership function which connects with $k$th rule, respectively. $c \in j$, ($j$ = 1, 2, ..., $l$ ), and $l$ is the maximum number of membership functions. If an adaptive threshold of $B_{ij}\left(x_i(t)\right)$ is considered, then the multiplication or combination of membership functions and rules can be realized automatically, the network owns self-organizing function to deal with different features of inputs.

The output of neurons $\mu, \sigma$ in stochastic layer is given by Eq. (14), Eq. (15) respectively.

$$\mu = \frac{\sum_k \lambda_k \ \ w_{\mu k}}{\sum_k \lambda_k} \tag{14}$$

$$\sigma = \frac{\sum_k \lambda_k \ \ w_{\sigma k}}{\sum_k \lambda_k} \tag{15}$$

Where $w_{\mu k}$, $w_{\sigma k}$ are the connections between $\mu, \sigma$ and rules, and $\mu, \sigma$ are the mean and standard deviation of stochastic function $\pi\left(\hat{y}(t+1), \mathbf{w}, \mathbf{x}(t)\right)$ whose description is given by Eq. (11). The output of system can be obtained by generating a random data according this probability function.

## 3. SGA of RL

### 3.1 Algorithm of SGA

A RL algorithm, Stochastic Gradient Ascent (SGA), is proposed by Kimura and Kobayashi (Kimura & Kobayashi, 1996, 1998) to deal with POMDP and continuous action space. Experimental results reported that SGA learning algorithm was successful for cart-pole control and maze problem. In the case of time series forecasting, the output of predictor can be considered as an action of agent, and the prediction error can be used as reward or punishment from the environment, so SGA can be used to train a neural forecasting system by renewing internal variable vector of NN (Kuremoto et. al, 2003, 2005).

The SGA algorithm is given below.

Step 1. Observe an input $\mathbf{x}(t)$ from training data of time series.

Step 2. Predict a future data $\hat{y}(t+1)$ according to a probability $\pi(\hat{y}(t+1), \mathbf{w}, \mathbf{x}(t))$.

Step 3. Receive the immediate reward $r_t$ by calculating the prediction error.

$$r_t = \begin{cases} r & if \ \left|\hat{y}(t+1) - y(t+1)\right| \le \varepsilon \\ -r & if \ \left|\hat{y}(t+1) - y(t+1)\right| > \varepsilon \end{cases} \tag{16}$$

Here $r$, $\varepsilon$ are evaluation constants greater than or equal to zero.

Step 4. Calculate characteristic eligibility $e_i(t)$ and eligibility trace $\overline{D_i}(t)$.

$$e_i(t) = \frac{\partial}{\partial w_i} \ln\{\pi(\hat{y}(t+1), \mathbf{w}, \mathbf{x}(t))\} \tag{17}$$

$$\overline{D_i}(t) = e_i(t) + \gamma \overline{D_i}(t-1) \tag{18}$$

Here $\gamma (0 \le \gamma < 1)$ is a discount factor, $w_i$ denotes *i*th internal variable vector.

Step 5. Calculate $\Delta w_i(t)$ by Eq. (19).

$$\Delta w_i(t) = (r_t - b)\overline{D_i}(t) \tag{19}$$

Here *b* denotes the reinforcement baseline.

Step 6. Improve policy by renewing its internal variable $\mathbf{w}$ by Eq. (20).

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_s \Delta \mathbf{w}(t) \tag{20}$$

Here $\Delta \mathbf{w}(t) = (\Delta w_1(t), \Delta w_2(t), \cdots, \Delta w_i(t), \cdots)$ denotes synaptic weights, and other internal variables of forecasting system, $\alpha_s$ is a positive learning rate.

Step 7. For next time step *t*+1, return to step 1.

Characteristic eligibility $e_i(t)$, shown in Eq. (17), means that the change of the policy function is concerning with the change of system internal variable vector (Williams, 1992). In fact, the algorithm combines reward/punishment to modify the stochastic policy with its internal variable renewing by step 4 and step 5. The finish condition of training iteration is also decided by the enough convergence of prediction error of sample data.

### 3.2 SGA for MLP

For the MLP forecasting system described in section 2.2 (Fig. 4), the characteristic eligibility $e_i(t)$ of Eq. (21)-(23) can be derived from Eq. (8)-(11) with the internal viable $w_{\mu k}$, $w_{\sigma k}$, $w_{ki}$ respectively.

$$\begin{aligned} e_{w_{\mu k}} &= \frac{\partial}{\partial w_{\mu k}} \ln(\pi) = \frac{\partial\{\ln(\pi)\}}{\partial \mu} \frac{\partial \mu}{\partial w_{\mu k}} \\ &= \frac{\beta_1 R_k \mu(1-\mu)(\hat{y}(t+1)-\mu)}{\sigma^2} \end{aligned} \tag{21}$$

$$\begin{aligned} e_{w_{\sigma k}} &= \frac{\partial}{\partial w_{\sigma k}} \ln(\pi) = \frac{\partial\{\ln(\pi)\}}{\partial \sigma} \frac{\partial \sigma}{\partial w_{\sigma k}} \\ &= \beta_2 R_k (1-\sigma) \frac{1}{\sigma} \left( \frac{(\hat{y}(t+1)-\mu)^2}{\sigma^2} - 1 \right) \end{aligned} \tag{22}$$

$$\begin{aligned} e_{w_{ki}} &= \frac{\partial}{\partial w_{ki}} \ln(\pi) = \frac{\partial\{\ln(\pi)\}}{\partial \mu} \frac{\partial \mu}{\partial R_k} \frac{\partial R_k}{\partial w_{ki}} + \frac{\partial\{\ln(\pi)\}}{\partial \sigma} \frac{\partial \sigma}{\partial R_k} \frac{\partial R_k}{\partial w_{ki}} \\ &= \beta_3 x_i(t)(1-R_k)(w_{\mu k} e_{w_{\mu k}} + w_{\sigma k} e_{w_{\sigma k}}) \end{aligned} \tag{23}$$

The initial values of $w_{\mu k}, w_{\sigma k}, w_{ki}$ are random numbers in (0, 1) at the first iteration of training. Gradient constants $\beta_1, \beta_2, \beta_3$ and reward parameters *r*, $\varepsilon$ denoted by Eq. (16) have empirical values.

### 3.3 SGA for SOFNN

For the SOFNN forecasting system described in section 2.3 (Fig. 5), the characteristic eligibility $e_i(t)$ of Eq. (24)-(27) can be derived from Eq. (11)-(15) with the internal viable $w_{\mu k}, w_{\sigma k}, m_{ij}, \sigma_{ij}$ respectively.

$$\begin{aligned} e_{w_{\mu k}} &= \frac{\partial}{\partial w_{\mu k}} \ln(\pi) = \frac{\partial\{\ln(\pi)\}}{\partial \mu} \frac{\partial \mu}{\partial w_{\mu k}} \\ &= \frac{\hat{y}(t+1)-\mu}{\sigma^2} \frac{\lambda_k}{\sum_k \lambda_k} \end{aligned} \tag{24}$$

$$\begin{aligned} e_{w_{\sigma k}} &= \frac{\partial}{\partial w_{\sigma k}} \ln(\pi) = \frac{\partial\{\ln(\pi)\}}{\partial \sigma} \frac{\partial \sigma}{\partial w_{\sigma k}} \\ &= \frac{1}{\sigma} \left( \frac{(\hat{y}(t+1)-\mu)^2}{\sigma^2} - 1 \right) \frac{\lambda_k}{\sum_k \lambda_k} \end{aligned} \tag{25}$$

$$
\begin{aligned}
e_{m_{ij}} &= \frac{\partial}{\partial m_{ij}} \ln(\pi) \\
&= \sum_k \left\{ \left( \frac{\partial \{\ln(\pi)\}}{\partial \mu} \frac{\partial \mu}{\partial \lambda_k} + \frac{\partial \{\ln(\pi)\}}{\partial \sigma} \frac{\partial \sigma}{\partial \lambda_k} \right) \frac{\partial \lambda_k}{\partial B_{ij}} \right\} \frac{\partial B_{ij}}{\partial m_{ij}} \\
&= \sum_k \left\{ \left( \begin{array}{l} \dfrac{\hat{y}(t+1)-\mu}{\sigma^2} \dfrac{w_{\mu k}-\mu}{\sum_k \lambda_k} + \\ \dfrac{1}{\sigma} \left( \dfrac{(\hat{y}(t+1)-\mu)^2}{\sigma^2} - 1 \right) \dfrac{w_{\sigma k}-\sigma}{\sum_k \lambda_k} \end{array} \right) \frac{\lambda_k}{B_{ij}} \right\} \frac{x_i - m_{ij}}{\sigma_{ij}^2} B_{ik}
\end{aligned}
\tag{26}
$$

$$
\begin{aligned}
e_{\sigma_{ij}} &= \frac{\partial}{\partial \sigma_{ij}} \ln(\pi) \\
&= \sum_k \left\{ \left( \frac{\partial \{\ln(\pi)\}}{\partial \mu} \frac{\partial \mu}{\partial \lambda_k} + \frac{\partial \{\ln(\pi)\}}{\partial \sigma} \frac{\partial \sigma}{\partial \lambda_k} \right) \frac{\partial \lambda_k}{\partial B_{ij}} \right\} \frac{\partial B_{ij}}{\partial \sigma_{ij}} \\
&= \sum_k \left\{ \left( \begin{array}{l} \dfrac{\hat{y}(t+1)-\mu}{\sigma^2} \dfrac{w_{\mu k}-\mu}{\sum_k \lambda_k} \\ + \dfrac{1}{\sigma} \left( \dfrac{(\hat{y}(t+1)-\mu)^2}{\sigma^2} - 1 \right) \dfrac{w_{\sigma k}-\sigma}{\sum_k \lambda_k} \end{array} \right) \frac{\lambda_k}{B_{ij}} \right\} \frac{(x_i - m_{ij})^2}{\sigma_{ij}^3} B_{ik}
\end{aligned}
\tag{27}
$$

Here membership function $B_{ik}$ is described by Eq. (12), fuzzy inference $\lambda_k$ is described by Eq. (13). The initial values of $w_{\mu k}, w_{\sigma k}, m_{ij}, \sigma_{ij}$ are random numbers included in (0, 1) at the first iteration of training. Reward *r*, threshold of evaluation error $\varepsilon$ denoted by Eq. (16) have empirical values.

## 4. Experiments

A chaotic time series generated by Lorenz equations was used as benchmark for forecasting experiments which were MLP using BP, MLP using SGA, SOFNN using SGA. Prediction precision was evaluated by the mean square error (MSE) between forecasted values and time series data.

### 4.1 Lorenz chaos

A butterfly-like attractor generated by the three ordinary differential equations (Eq. (28)) is very famous on the early stage of chaos phenomenon study (Lorenz, 1969).

$$\begin{cases} \dot{o}(t) = \delta p(t) - \delta o(t) \\ \dot{p}(t) = -o(t)q(t) + \phi o(t) - p(t) \\ \dot{q}(t) = o(t)p(t) - \varphi q(t) \end{cases} \tag{28}$$

Here $\delta, \phi, \varphi$ are constants. The chaotic time series was obtained from dimension $o(t)$ of Eq. (29) in forecasting experiments, where $\Delta t = 0.005$, $\delta = 16.0$, $\phi = 45.92$, $\varphi = 4.0$.

$$\begin{cases} o(t+1) = o(t) + \Delta t \sigma (p(t) - o(t)) \\ p(t+1) = p(t) - \Delta t (o(t)q(t) - \phi o(t) + p(t)) \\ q(t+1) = q(t) + \Delta t (o(t)p(t) - \varphi q(t)) \end{cases} \tag{29}$$

The size of sample data for training is 1,000, and the continued 500 data were served as unknown data for evaluating the accuracy of short-term (i.e. one-step ahead) prediction.

### 4.2 Experiment of MLP using BP

It is very important and difficult to construct a good architecture of MLP for nonlinear prediction. An experimental study (Oliveira et. al, 2000) showed the different prediction results for Lorenz time series by the architecture of $n : 2n : n : 1$, where $n$ denotes the embedding dimension and the cases of $n$ = 2, 3, 4 were investigated for different term predictions (long-term prediction).

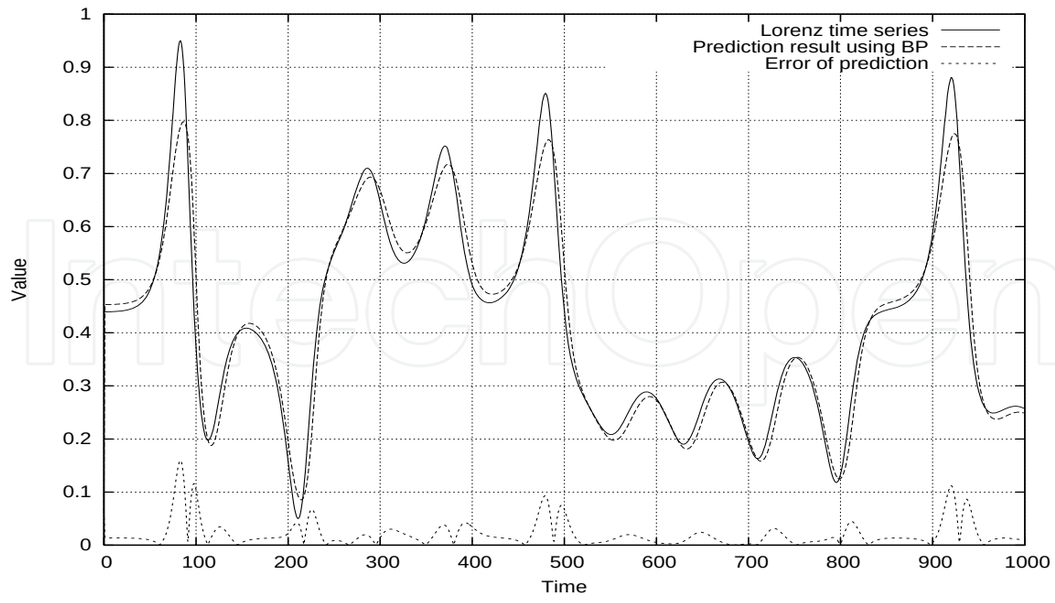


Fig. 6. Prediction results after 2,000 iterations of training by MLP using BP.

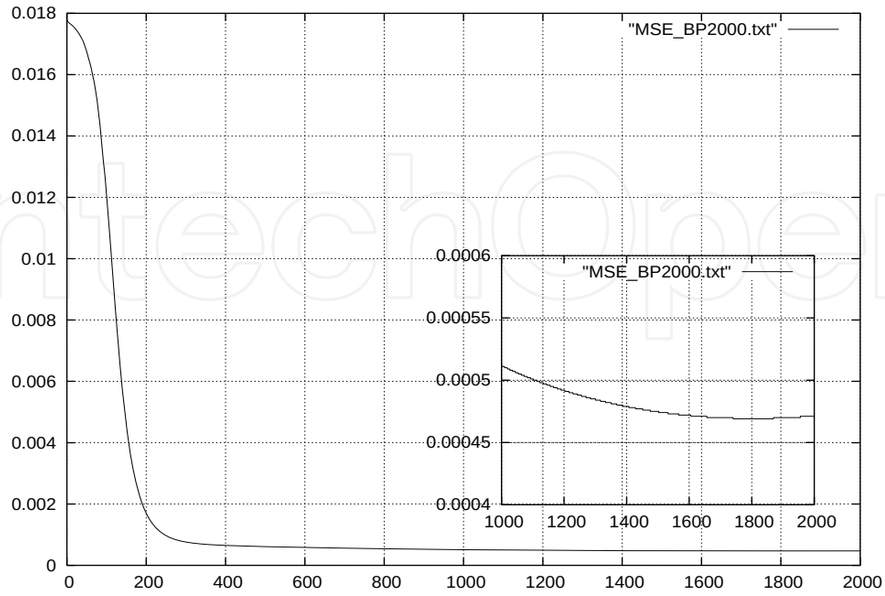


Fig. 7. Prediction error (MSE) in training iteration of MLP using BP.

For short-term prediction here, a three-layer MLP using BP and *3* : *6* : *1* structure shown in Fig. 3 was used in experiment, and time delay $\tau$ =1 was used in embedding input space. Gradient constant of sigmoid function $\beta$ = 1.0, discount constant $\alpha$ = 1.0, learning rate $\eta$ = 0.01,

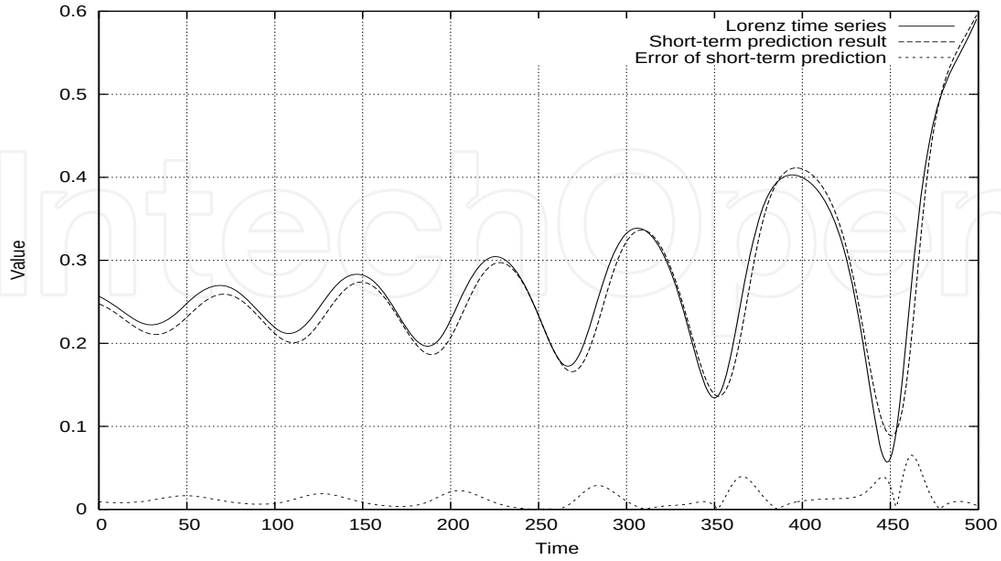


Fig. 8. One-step ahead forecasting results by MLP using BP.

and the finish condition of training was set to $E(W) < 5.0\times10^{-4}$. The prediction results after training 2,000 times are shown in Fig. 6, and the change of prediction error according to the iteration of training is shown in Fig. 7. The one-step ahead prediction results are shown in Fig. 8. The 500 steps MSE of one-step ahead forecasting by MLP using BP was 0.0129.

### 4.3 Experiment of MLP using SGA

A four-layer MLP forecasting system with SGA and *3* : *60* : *2* : *1* structure shown in Fig. 4 was used in experiment, and time delay $\tau$ =1 was used in embedding input space. Gradient

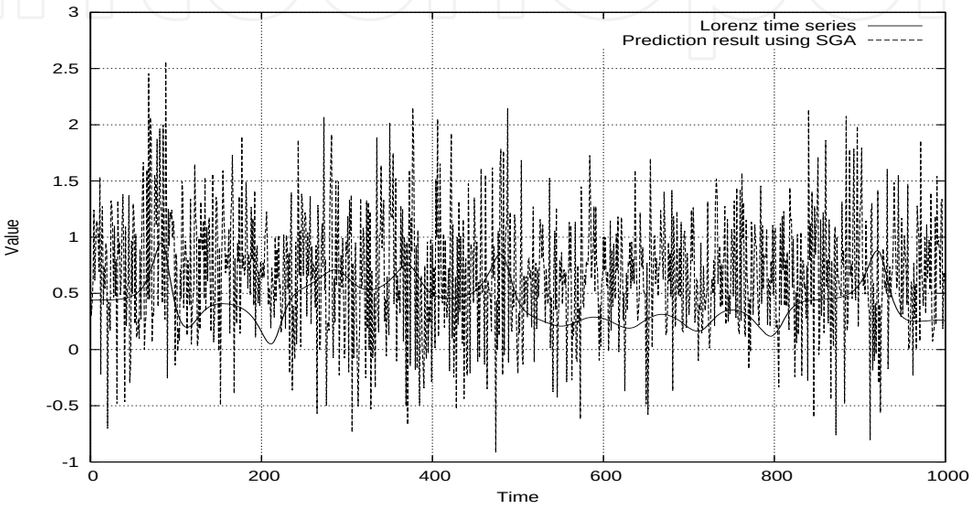


Fig.9. Prediction results before iteration by MLP using SGA.

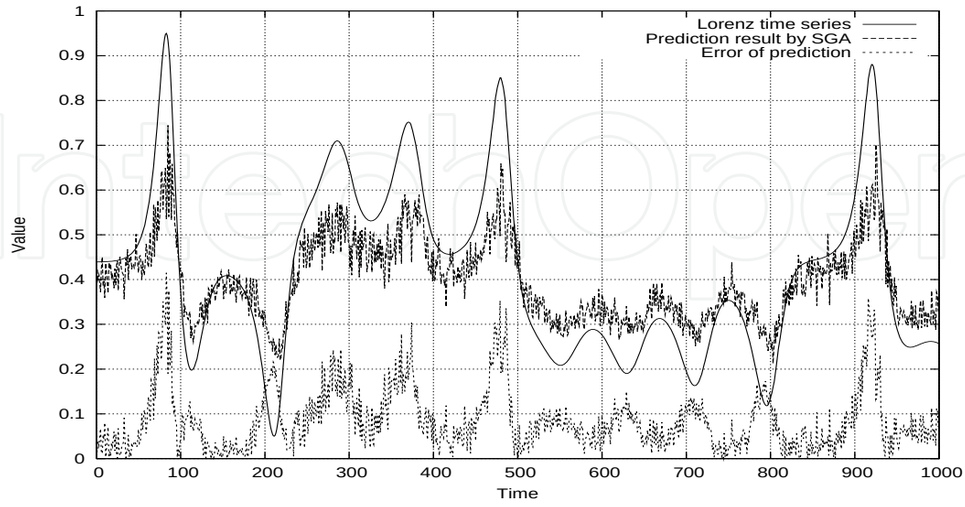


Fig. 10. Prediction results after 5,000 iterations of training by MLP using SGA.

constants of sigmoid functions $\beta_1 = 8.0, \beta_2 = 18.0, \beta_3 = 10.0$, discount constant $\gamma$ = 0.9, learning rate $\alpha_{wij} = \alpha_{wjk} = 2.0 \times 10^{-6}, \alpha_{wkl} = 2.0 \times 10^{-5}$, the reward was set by Eq. (30), and the finish condition of training was set to 30,000 iterations where the convergence $E(W)$ could be observed. The prediction results after 0, 5,000, 30,000 iterations of training are shown in Fig. 9, Fig. 10 and Fig. 11 respectively. The change of prediction error during training is shown in Fig. 12. The one-step ahead prediction results are shown in Fig. 13. The 500 steps MSE of one-step ahead forecasting by MLP using SGA was 0.0112, forecasting accuracy was 13.2% upped than MLP using BP.

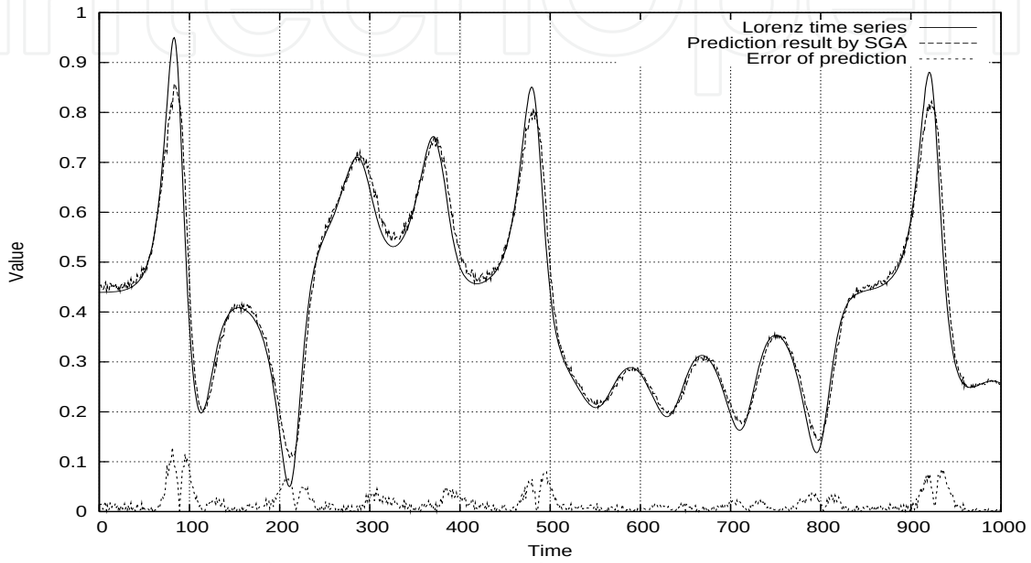


Fig. 11. Prediction results after 30,000 iterations of training by MLP using SGA.

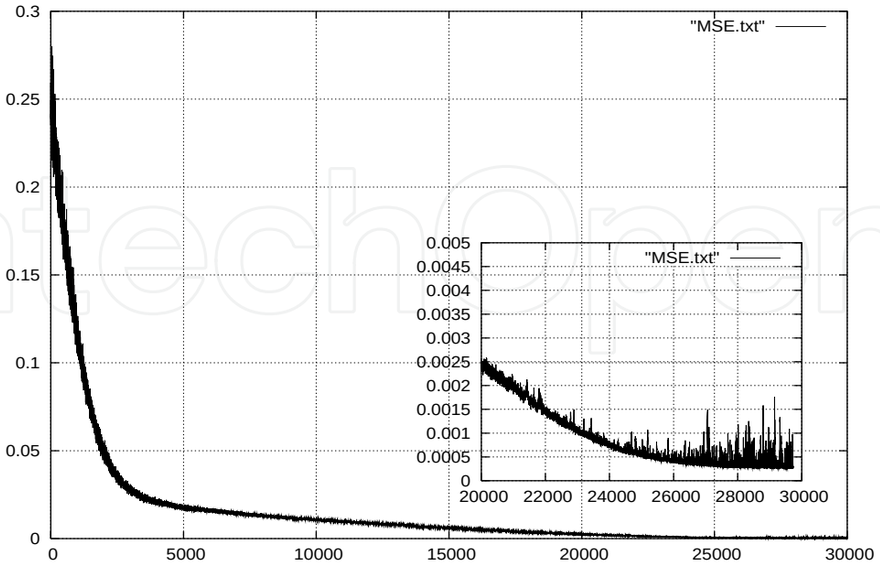


Fig. 12. Prediction error (MSE) in training iteration of MLP using SGA.

$$r_t = \begin{cases} 4.0E-4 & if\ \left|\hat{y}(t+1) - y(t+1)\right| \le 0.1 \\ -4.0E-4 & if\ \left|\hat{y}(t+1) - y(t+1)\right| > 0.1 \end{cases} \tag{30}$$

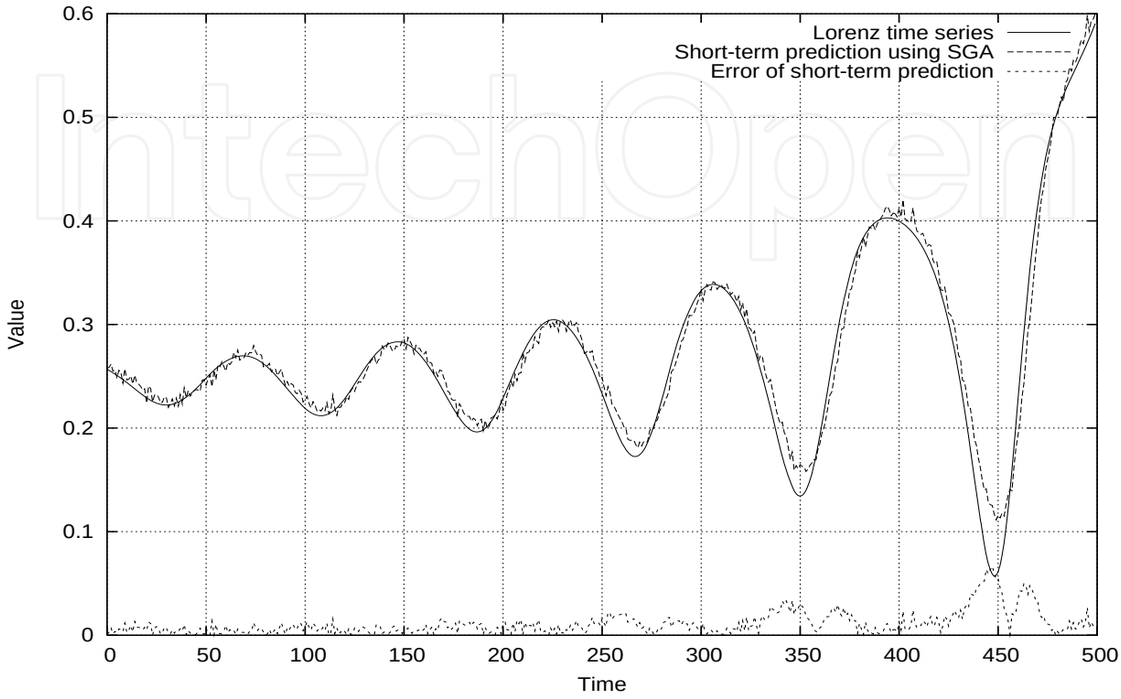


Fig. 13. One-step ahead forecasting results by MLP using SGA.

### 4.4 Experiment of SOFNN using SGA

A five-layer SOFNN forecasting system with SGA and structure shown in Fig. 5 was used in experiment, time delay $\tau$=2 was used in 3, 4, or 5-dimensional embedding input spaces. Initial value of weight $w_{\mu k}$ had random values in (0.0, 1.0), $w_{\sigma k} = 0.5, m_{ij} = 0.0, \sigma_{ij} = 15.0$ and discount $\gamma$ = 0.9, learning rate $\alpha_{mij} = \alpha_{w\sigma ij} = \alpha_{w\sigma k} = 3.0\times10^{-6}, \alpha_{w\mu k} = 2.0\times10^{-3}$, the reward *r* was set by Eq. (31), and the finish condition of training was also set to 30,000 iterations where the convergence *E*(*W*) could be observed. The prediction results after training are shown in Fig. 14, where the number of input neurons was 4 and data scale of results was modified into (0.0, 1.0). The change of prediction error during the training is shown in Fig. 15. The one-step ahead prediction results are shown in Fig. 16. The 500 steps MSE of one-step ahead forecasting by SOFNN using SGA was 0.00048, forecasting accuracy was 95.7% and 96.3% upped than the case by MLP using BP and by MLP using SGA respectively.

$$r_t = \begin{cases} 1.5 & if\ \left|\hat{y}(t+1) - y(t+1)\right| \le 1.5 \\ -1.5 & if\ \left|\hat{y}(t+1) - y(t+1)\right| > 1.5 \end{cases} \tag{31}$$

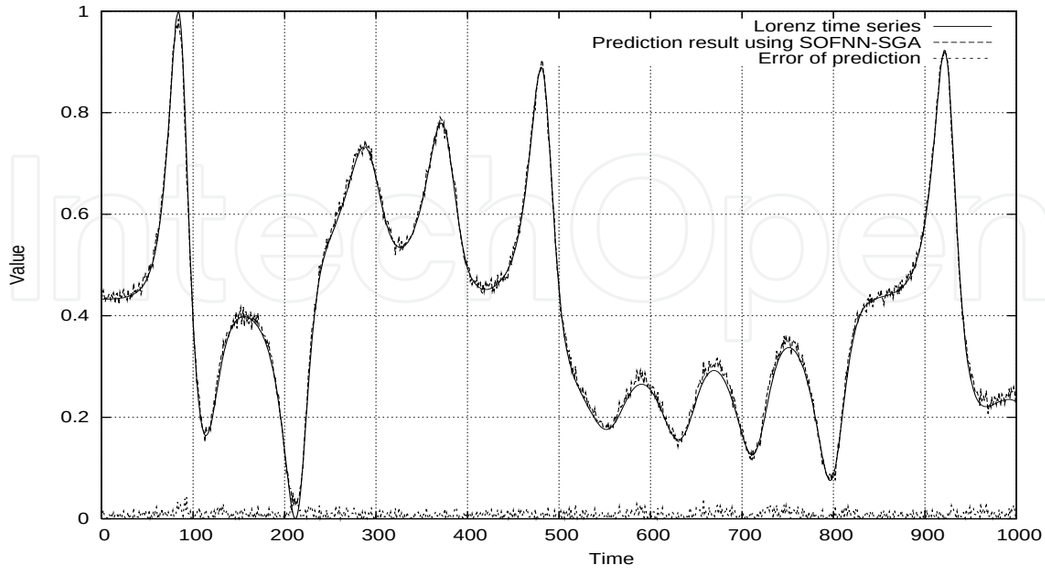


Fig. 14. Prediction results after 30,000 iterations of training by SOFNN using SGA.

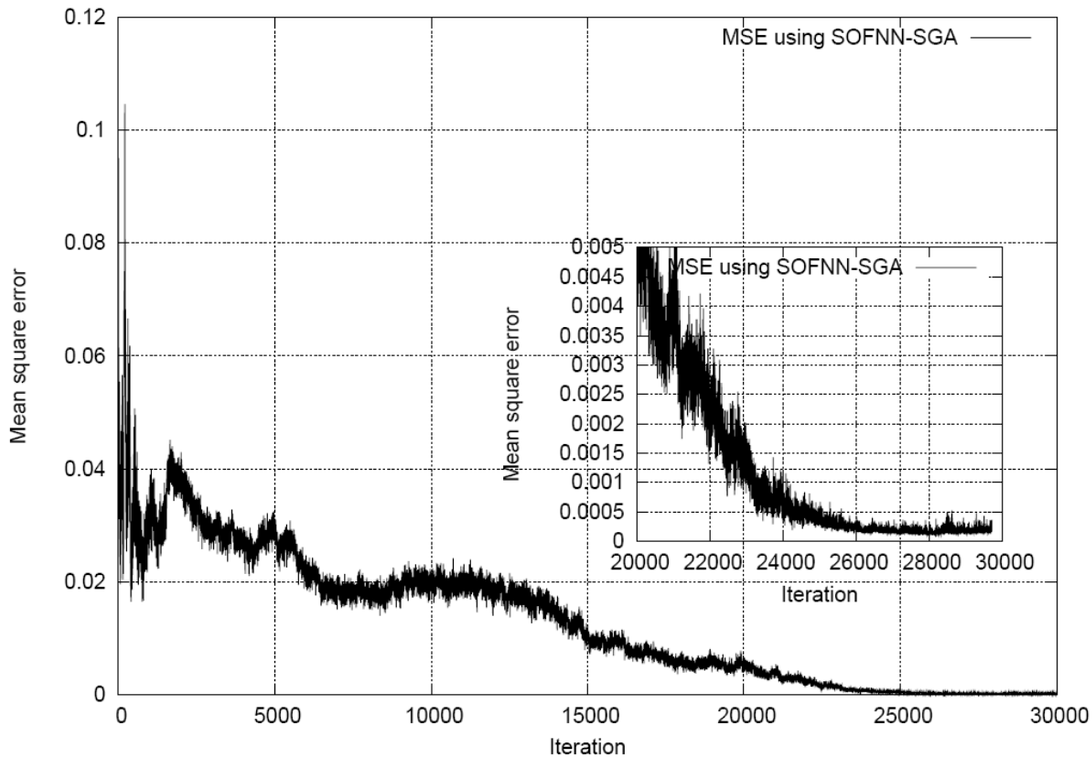


Fig. 15. Prediction error (MSE) in training iteration of SOFNN using SGA.

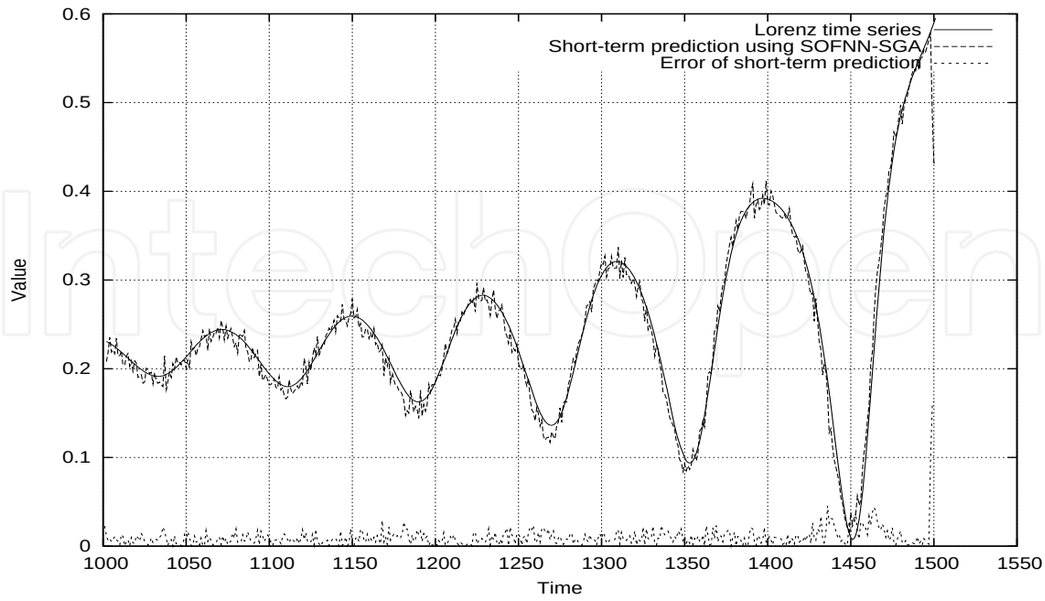


Fig. 16. One-step ahead forecasting results by SOFNN using SGA.

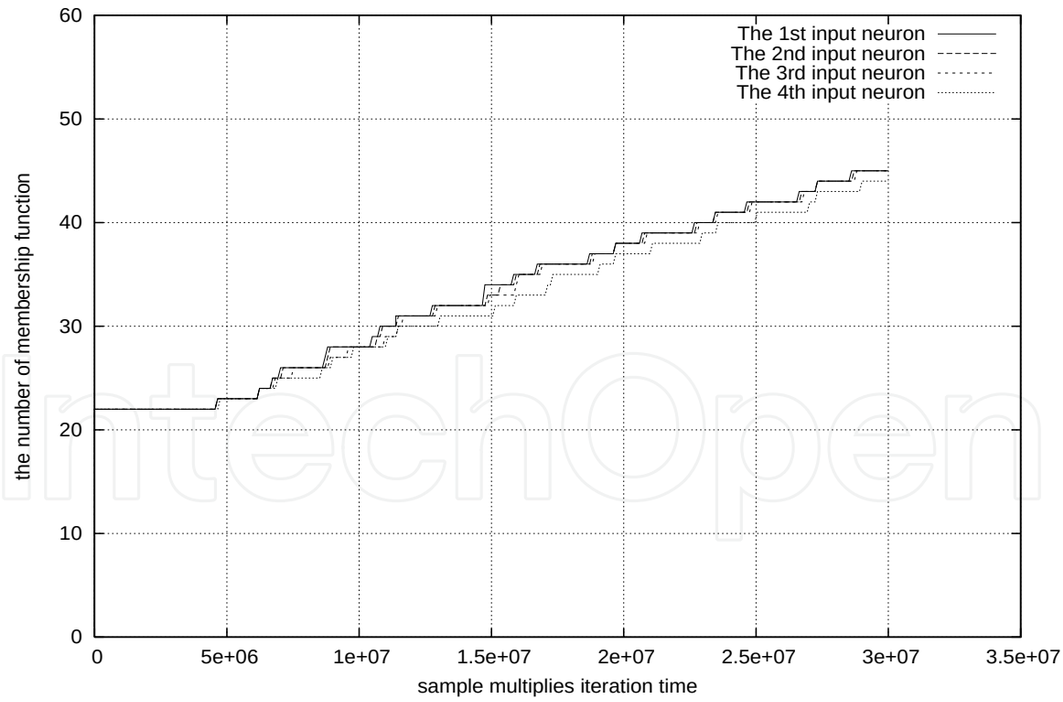


Fig. 17. The number of membership function neurons of SOFNN using SGA increased in training experiment.

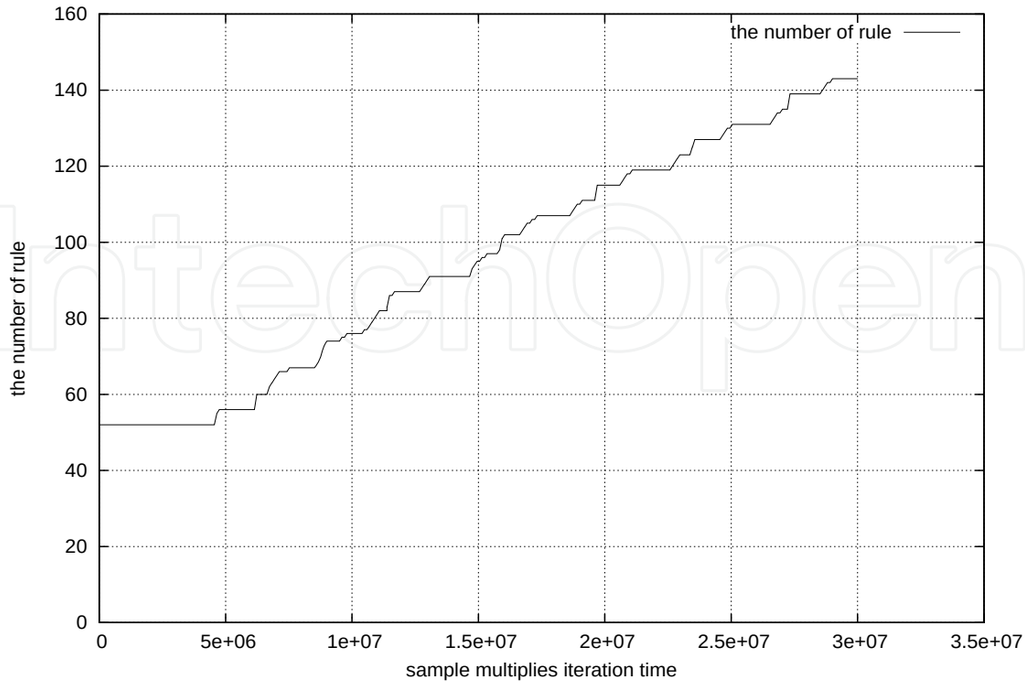


Fig. 18. The number of rules of SOFNN using SGA increased in training experiment.

One advanced feature of SOFNN is its data-driven structure building. The number of membership function neurons and rules increased with samples (1,000 steps in training of experiment) and iterations (30,000 times in training of experiment), which can be confirmed by Fig. 17 and Fig. 18. The number of membership function neurons for the 4 input neurons was 44, 44, 44, 45 respectively, and the number of rules was 143 when the training finished.

## 5. Conclusion

Though RL has been developed as one of the most important methods of machine learning, it is still seldom adopted in forecasting theory and prediction systems. Two kinds of neural forecasting systems using SGA learning were described in this chapter, and the experiments of training and short-term forecasting showed their successful performances comparing with the conventional NN prediction method. Though the iterations of MLP with SGA and SOFNN with SGA in training experiments took more than that of MLP with BP, both of their computation time were not more than a few minutes by a computer with 3.0GHz CPU. A problem of these RL forecasting systems is that the value of reward in SGA algorithm influences learning convergence seriously, the optimum reward should be searched experimentally for different time series. Another problem of SOFNN with SGA is how to tune up initial value of deviation parameter in membership function and the threshold those were also modified by observing prediction error in training experiments. In fact, when SOFNN with SGA was applied on an neural forecasting competition "NN3" where 11 time series sets were used as benchmark, it did not work sufficiently in the long-term prediction comparing with the results of other methods (Kuremoto et. al, 2007; Crone & Nikolopoulos,

2007). All these problems remain to be resolved, and it is expected that RL forecasting systems will be developed remarkably in the future.

## Acknowledgements

We would like to thank Mr. Yamamoto A. and Mr. Teramori N. for their early work in experiments, and a part of this study was supported by MEXT-KAKENHI (15700161) and JSPS-KAKENHI (18500230).

## 6.References

Box, G. E. P. & Jenkins, G. (1970). *Time series analysis: Forecasting and control*. Holden-Day, ISBN-10 0816211043, San Francisco

Casdagli, M. (1989). Nonlinear prediction of chaotic time series. *Physica D: Nonlinear Phenomena*. Vol. 35, pp. 335-356

Crone, S. & Nikolopoulos, K. (2007). Results of the NN3 neural network forecasting competition. *The 27th International Symposium on Forecasting*. Program, pp. 129

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of U. K. inflation. *Econometrica*. Vol. 50, pp. 987-1008

Kimura, H., Yamamura, M. & Kobayashi S. (1996). Reinforcement learning in partially observable Markov decision process: A stochastic gradient ascent (in Japanese). *Journal of Japanese Society for Artificial Intelligent,* pp. 761-768

Kimura, H. & Kobayashi S. (1998). Reinforcement learning for continuous action using stochastic gradient ascent. *Intelligent Autonomous Systems*, pp. 288-295

Kodogiannis, V. & Lolis, A. (2002). Forecasting financial time series using neural network and fuzzy system-based techniques. *Neural computing & applications*. Vol. 11, pp. 90-102

Kuremoto, T., Obayashi, M., Yamamoto, A. & Kobayashi, K. (2003). Predicting chaotic time series by reinforcement learning. *Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems* (CIRAS '03), Singapore

Kuremoto, T., Obayashi, & Kobayashi, K. (2005). Nonlinear prediction by reinforcement learning. In: *Lecture Notes in Computer Science,* Vol. 3644, pp.1085-1094, Springer, ISBN 0302-9743 (Print) 1611-3349 (Online), Berlin

Kuremoto, T., Obayashi, & Kobayashi, K. (2007). Forecasting time series by SOFNN with reinforcement learning. *The 27th International Symposium on Forecasting*. Program, pp. 99

Lendasse, A., Oja, E., Simula, O. & Verleysen, M. (2007). Time series prediction competition: The CATS benchmark. *Neurocomputing*. Vol. 70, pp. 2325-2329

Leung, H., Lo, T., & Wang S. (2001). Prediction of noisy chaotic time series using an optimal radial basis function. *IEEE Transaction on Neural Networks*. Vol. 12, pp.1163-1172

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the atmosphere Sciences*. Vol. 20, pp. 130-141

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, Vol. 261, pp. 459-467

Oliveira, K. A., Vannucci, A. & Silva, E. C. (2000). Using artificial neural networks to forecast chaotic time series. *Physica A*. Vol. 284, pp. 393-404

Rumelhart, D. E., Hinton, G. E. & R. J. Williams, R. J. (1986). Learning representation by back-propagating errors. *Nature*. Vol. 232, No. 9, pp. 533-536

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: an introduction*. The MIT Press, ISBN 0-262-19398-1, Cambridge

Takens., F. (1981). Detecting strange attractor in turbulence. *Lecture Notes in Mathematics*, Vol. 898, pp. 366-381, Springer-Verlag, Berlin

Williams., R. J. (1992). Simple statistical gradient following algorithms for connectionist reinforecement learning. *Machine Learning*, Vol. 8, pp. 229-256

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, Vol. 50, pp. 159-175


**Reinforcement Learning**

Edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer

ISBN 978-3-902613-14-1
Hard cover, 424 pages
**Publisher** I-Tech Education and Publishing
**Published online** 01, January, 2008
**Published in print edition** January, 2008

Brains rule the world, and brain-like computation is increasingly used in computers and electronic devices. Brain-like computation is about processing and interpreting data or directly putting forward and performing actions. Learning is a very important aspect. This book is on reinforcement learning which involves performing actions to achieve a goal. The first 11 chapters of this book describe and extend the scope of reinforcement learning. The remaining 11 chapters show that there is already wide usage in numerous fields. Reinforcement learning can tackle control tasks that are too complex for traditional, hand-designed, non-learning controllers. As learning computers can deal with technical complexities, the tasks of human operators remain to specify goals on increasingly higher levels. This book shows that reinforcement learning is a very dynamic area in terms of theory and applications and it shall stimulate and encourage new research in this field.



INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen