Efficient Learning Algorithm Based On Maximum Likelihood Principle for Multilayer Feedforward Stochastic Neural Network

Shigeo Kamitsuji

Keio University

Ritei Shibata

Keio University

Abstract. Efficient learning algorithm based on maximum likelihood principle is proposed for *multilayer feedforward stochastic neural network*, in which each neuron emits 0 or 1 according to a conditional probability determined by a linear combination of given inputs. To reduce the burden of computation, the log likelihood is approximated by an average of the log conditional likelihoods, which are evaluated by the Monte Carlo simulations. Also the back-propagation algorithm for such a stochastic neural network has been developed. It is proved that the connection weights among neurons updated by this algorithm converge to the weights which maximize the likelihood of the whole network, as the number of iterations increases. The advantages of using the multilayer feedforward stochastic neural network over deterministic neural network or Boltzmann machine are shown by giving a practical example of predicting the fall or the rise of the Tokyo Stock Price Index.

Keywords: Autoregressive model, Boltzmann machine, Deterministic neural network, Maximum likelihood, Multilayer feedforward stochastic neural network, Monte Carlo simulation, Stochastic neuron, Tokyo stock price index.

Abbreviations: KAP – Kluwer Academic Publishers; compuscript – Electronically submitted article

JEL codes: D24, L60, 047

Nomenclature:

KAP – Kluwer Academic Publishers; compuscript – Electronically submitted article

1. Introduction

Stochastic neural networks are networks in which each neuron emits 0 or 1 according to a conditional probability determined by a linear combination of given inputs. Stochastic neural networks are powerful tools for non-linear modeling of stochastic phenomena. Elementary introductions to the theory of stochastic neural networks can be found in Amari (1993) or in Haykin (1999). Stochastic neural networks have been used in a wide variety of applications. Such an application can be found in Baba (1998) where a stochastic neural network is used for predicting TOPIX (Tokyo Stock Price Index). Given the stochastic

nature of stock indices, one might expect that a well-chosen stochastic neural network has the potential to learn the underlying stochastic mechanism better than a conventional deterministic neural network.

Stochastic neural network we apply here is a hierarchical network in which the neurons are stochastic neurons except neurons in the input layer, that is, multilayer feedforward stochastic neural network. The connections in the stochastic neural network are uni-directional, that is, the neurons in a layer are only connected to the neurons in the next layer. Each connection has a connection weight, and connection weights control the behavior of outputs of the stochastic neural network. Multilayer feedforward stochastic neural network is an input-output system and learns a number of patterns from input-output pairs.

To estimate the weights so as to learn a number of patterns from given pairs of inputs and outputs, it is necessary to select a suitable optimization criteria for the stochastic neural network. In this paper, we introduce the maximum likelihood principle, which is a most natural way of making full use of stochastic nature of the network. However, the likelihood function of the stochastic neural network is generally more complicated to compute, differently from the squared error function in deterministic neural network or an approximation of the outputs of stochastic neuron in stochastic neural network, since we apply the exact likelihood function of the stochastic neural network Neverthe-

less the importance of the stochastic neural network, as non-linear stochastic models for input-output systems, outweighs any additional computational complexity and cost.

The development of a computationally efficient algorithm is the key to the practical application of any stochastic neural networks, except in the case where the network is very simple, for example with only one hidden layer as in Kurita (1990). In this paper, we succeeded in reducing the burden of computation by introducing two techniques, the Monte Carlo simulation and the back-propagation algorithm (Rumelhart et al., 1986). The likelihood function of the stochastic neural network can be evaluated by performing the Monte Carlo simulations, and the idea of the well known back-propagation algorithm for deterministic neural network is employed to our algorithm. Convergence of the back-propagation algorithm is not guaranteed because of the complexity of the criterion function, but convergence of our learning algorithm for the stochastic neural network can be proved with several assumptions.

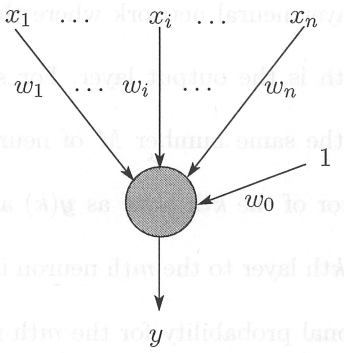
The advantages of using multilayer feedforward stochastic neural network also include making possible more flexible and adaptive. The stochastic neuron plays a role by not only describing and approximating the random mechanism generating the data, but also by providing a formal statistical understanding of the discrepancies between the data

and the model. A practical example given in Section 6 demonstrates the latter.

In Section 2, the multilayer feedforward stochastic neural network is formally defined and an explicit representation of the likelihood and the maximum likelihood method for the stochastic neural network are given in Section 3. A learning algorithm for the stochastic neural network are given in Section 4. The convergence of learning algorithm is proved in Section 5. In Section 6, a practical application for predicting the fall or the rise of TOPIX is given.

2. Multilayer Feedforward Stochastic Neural Network

With the exception of the deterministic neurons in the input layer, the neurons in MFSNN, called stochastic neurons, emits 1 with probability $f(z)$ and 0 with probability $1 - f(z)$ for a given level of the input z. Figure 1 depicts a stochastic neuron influenced by binary inputs x_i $(i = 1, \ldots, n)$ which are outputs from neurons in the previous layer, including the input $x_0 = 1$ from a deterministic neuron. In Figure 1, the stochastic neuron is distinguished from the non-stochastic neuron by hatching. The probability of the output $y = 1$ is determined by the value of an activation function f for a linear combination $z = \sum_{i=0}^{n} w_i x_i$



$$P(y = 1) = f(\textstyle\sum_{i=1}^{n} w_i x_i + w_0)$$

Figure 1. Stochastic neuron.

of inputs $\boldsymbol{x} = (x_0, \ldots, x_n)$ where z measures the "level of activity" of the system and $\boldsymbol{w} = (w_0, \ldots, w_n)$ are the constant connection weights. A natural choice of the activation function $f(z)$ would be a monotone increasing function, because a neuron is more likely to activate as the value of the input z increases. Sigmoid activation function,

$$f(z) = \frac{1}{1 + \exp(-z)}, \tag{1}$$

is frequently used as an activation function. For alternative choice, we may use the probit function

$$f(z) = \Phi(z) \tag{2}$$

instead, where $\Phi(\cdot)$ is the standard normal probability distribution function. Note that the value of $f(z)$ should range over $[0, 1]$ since $f(z)$ is a probability of output being 1.

Now consider a K layer neural network where the first layer is the input layer and the Kth is the output layer. For simplicity, we also assume that there are the same number M of neurons in each layer. Denote the output vector of the kth layer as $\boldsymbol{y}(k)$ and the connection weight vector from the kth layer to the mth neuron in the $k+1$th layer as $\boldsymbol{w}_m(k)$. The conditional probability for the mth neuron to activate is then given by $f(\boldsymbol{w}_m(k)^T\boldsymbol{y}(k))$. Figure 2 is an illustration of such a MFSNN. The meaning of the two boxes in the figure will be explained later in Section 4.3. The joint probability of M neurons taking the value $\boldsymbol{y}(k+1)$ given the value $\boldsymbol{y}(k)$ is by the function

$$p(\boldsymbol{y}(k+1)|\boldsymbol{y}(k)) = \prod_{m=1}^{M} \left\{f(\boldsymbol{w}_m(k)^T\boldsymbol{y}(k))\right\}^{y_m(k+1)} \times \left\{1 - f(\boldsymbol{w}_m(k)^T\boldsymbol{y}(k))\right\}^{1-y_m(k+1)}, \quad (3)$$
$$k = 1, 2, \ldots, K-1.$$

Here $\boldsymbol{y}(1)$ and $\boldsymbol{y}(K)$ stand for the input $\boldsymbol{x}$ and the output $\boldsymbol{t}$, respectively.

This is a kind of Markovian property of the multilayer feedforward stochastic neural network. This property makes easier to develope maximization algorithm of the likelihood. In fact, the probability $P(\boldsymbol{t}|\boldsymbol{x})$ of the output $\boldsymbol{t}$ of the neural network given the input $\boldsymbol{x}$ is written as

$$P(\boldsymbol{t}|\boldsymbol{x}) = \sum_{\boldsymbol{y}(K-1),\ldots,\boldsymbol{y}(2)} p(\boldsymbol{t}|\boldsymbol{y}(K-1)) \times p(\boldsymbol{y}(K-1)|\boldsymbol{y}(K-2)) \cdots p(\boldsymbol{y}(2)|\boldsymbol{x}), \quad (4)$$

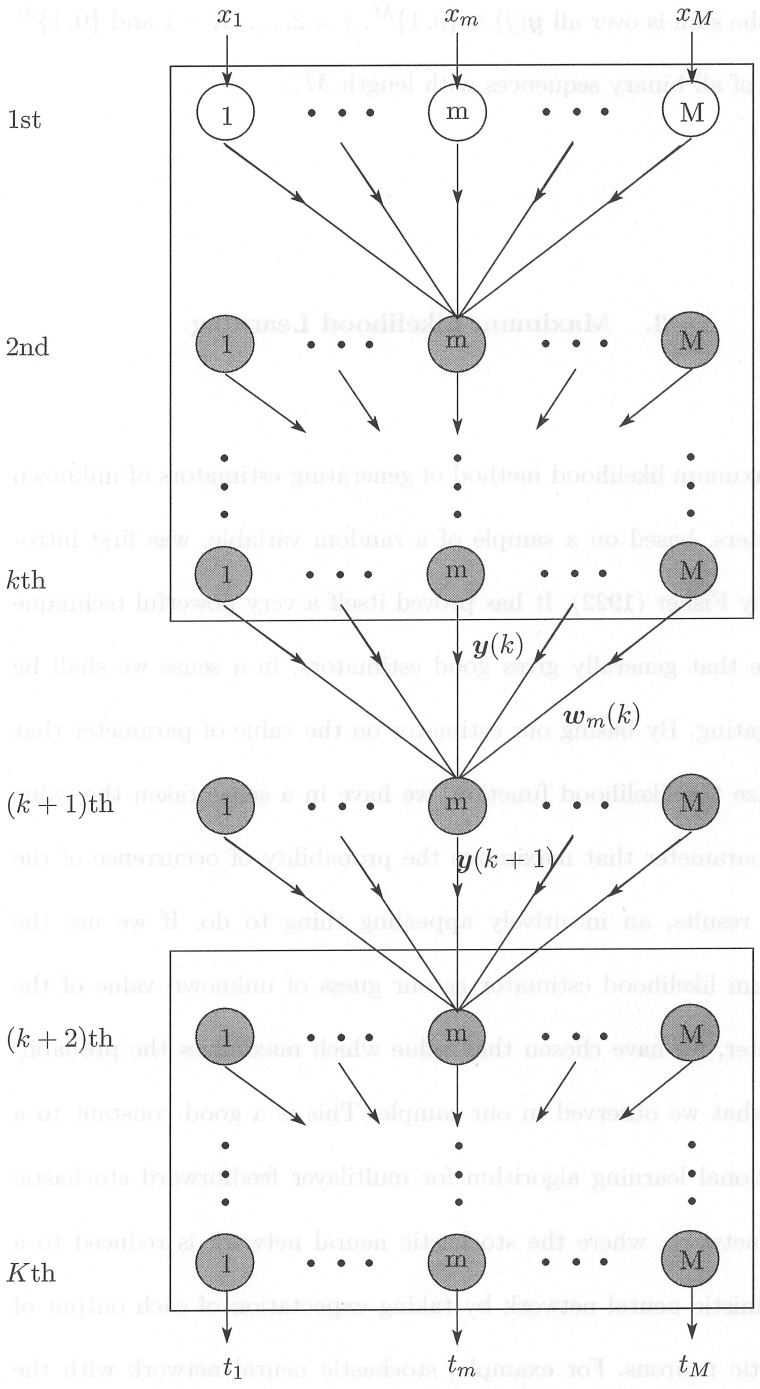


Figure 2. Multilayer feedforward neural network with K layer.

where the sum is over all $\boldsymbol{y}(j) \in \{0,1\}^M$, $j = 2, \ldots, K-1$ and $\{0,1\}^M$ is a set of all binary sequences with length M.

3. Maximum Likelihood Learning

The maximum likelihood method of generating estimators of unknown parameters, based on a sample of a random variable, was first introduced by Fisher (1922). It has proved itself a very powerful technique and one that generally gives good estimators, in a sense we shall be investigating. By basing our estimator on the value of parameter that maximize the likelihood function, we have in a sense taken the value of the parameter that maximizes the probability of occurrence of the sample results, an intuitively appealing thing to do. If we use the maximum likelihood estimator as our guess of unknown value of the parameter, we have chosen that value which maximizes the probability of what we observed in our sample. This is a good constant to a conventional learning algorithm for multilayer feedforward stochastic neural network, where the stochastic neural network is reduced to a deterministic neural network by taking expectation of each output of stochastic neurons. For example, stochastic neural network with the

sigmoid activation function $f(z)$ is reduced to a deterministic neural network with activation function $\tanh(z)$ (Hassoun, 1995).

For an independent training sample of input-output pairs $(\boldsymbol{x}^{(1)}, \boldsymbol{t}^{(1)})$, $\ldots$, $(\boldsymbol{x}^{(n)}, \boldsymbol{t}^{(n)})$ the likelihood of the weight vectors $\boldsymbol{w}(1), \ldots, \boldsymbol{w}(K-1)$ is now given by

$$L(\boldsymbol{w}(1), \ldots, \boldsymbol{w}(K-1)) = \prod_{i=1}^{n} P(\boldsymbol{t}^{(i)}|\boldsymbol{x}^{(i)}), \tag{5}$$

where $\boldsymbol{w}(k) = (\boldsymbol{w}_1(k), \ldots, \boldsymbol{w}_m(k))^T$, $k = 1, \ldots, K-1$ is the vector of all connection weights from the kth later to the $(k+1)$th layer.

Now, we want to estimate the weights $\boldsymbol{w}(1), \ldots, \boldsymbol{w}(K-1)$ so as to maximize the likelihood (5) of the stochastic neural network. However, it is tedious and inefficient to estimate the weights all at once because that the likelihood function (5) is generally very complicated. Then, in this paper we propose a learning algorithm which the weights $\boldsymbol{w}(k)$, $k = 1, \ldots, K-1$ are updated one by one, like the back-propagation algorithm. For that purpose, it is better to rewrite the likelihood function (5) by using the recursive formula of the conditional probability $P(\boldsymbol{t}|\boldsymbol{x})$ in (4). To update the weights $\boldsymbol{w}(k)$ when the target layer is the kth layer, it is convenient to work with the log of the likelihood L and

$$l = \log L = \sum_{i=1}^{n} \log \sum_{\boldsymbol{y}(k) \in \{0,1\}^M} P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k)) P(\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}), \tag{6}$$

holds true for any $k = 1, \ldots, K-1$. Here we note that the weights $\boldsymbol{w}(k)$ is involved only in the conditional probability $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$ for the

sub-network which consists of all layers from the $(k+1)$th to the Kth. This implies that the weights $\boldsymbol{w}(k)$ can be updated layer by layer.

Then, the vector of weights $\boldsymbol{w}(k)$ is updated by the Newton method, that is, updated to

$$\boldsymbol{w}(k) - \eta \left(\nabla_k^2 l\right)^{-1} \nabla_k l. \tag{7}$$

Here η is a learning rate, $\nabla_k l$ is the M^2 dimensional gradient vector,

$$\nabla_k l = \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{w}_1(k)} l \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{w}_M(k)} l \end{pmatrix} \tag{8}$$

and $\nabla_k^2 l$ is the $M^2 \times M^2$ dimensional Hessian matrix,

$$\nabla_k^2 l = \left(\frac{\partial^2}{\partial \boldsymbol{w}_m(k) \partial \boldsymbol{w}_{m'}(k)^T} l;\ 1 \leq m, m' \leq M \right). \tag{9}$$

An alternative way of updating weight vector is by Natural Gradient method (Amari, 1998; Amari et al., 2000), in which the expectation $\mathrm{E}(\nabla_k^2 l)$ is used in place of $\nabla_k^2 l$, or EM algorithm (Amari, 1995). However, we concentrate our attention into Newton method for simplicity in this paper.

It is however tedious to evaluate all gradient vectors $\nabla_k l$, $k = 1, \ldots, K-1$ and Hessian matrices $\nabla_k^2 l$ for each $k = 1, \ldots, K-1$. Such a burden is resolved by introduction of the notion of conditional likelihood $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$. Then the log likelihood l in (6) can be regarded as the sum of expectation of the conditional likelihood $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$ for

each $i = 1, \ldots, n$. The same discussion holds true for the gradient and the Hessian, and leads us to the idea of employing the Monte Carlo simulations for the evaluation of such expectations.

4. Derivation of An Efficient Learning Algorithm

In this section, we describe our algorithm into details, including the back-propagation and the Monte Carlo simulations implemented in.

4.1. Back-propagation Algorithm for Multilayer Feedforward Stochastic Neural Network

We will show that the weights $\boldsymbol{w}(k)$ for the kth layer can be updated on the basis of the conditional likelihood for layers below the kth layer. To derive a back-propagation algorithm for the multilayer feedforward stochastic neural network, we rewrite the log likelihood l in (6) to

$$l = \sum_{i=1}^{n} \log E^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}} \left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k)) \right], \tag{10}$$

where $E^{\boldsymbol{y}|\boldsymbol{x}}$ denotes the conditional expectation with respect to $\boldsymbol{y}$ for a given $\boldsymbol{x}$. Note that $\boldsymbol{w}(k)$ is involved only in $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$ not in the

expectation. The gradient $\nabla_k l$ is then rewritten as

$$\nabla_k l = \sum_{i=1}^{n} \frac{\nabla_k \, \mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]}{\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]} = \sum_{i=1}^{n} \frac{\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[\nabla_k \, P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]}{\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]}. \tag{11}$$

The gradient $\nabla_k P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$ which appears on the right hand side of (11) can be rewritten as

$$\begin{aligned}
\nabla_k P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k)) &= \nabla_k \sum_{\boldsymbol{y}(k+1)\in\{0,1\}^M} P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1)) \; p(\boldsymbol{y}(k+1)|\boldsymbol{y}(k)) \\
&= \sum_{\boldsymbol{y}(k+1)\in\{0,1\}^M} P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1)) \; \nabla_k \, p(\boldsymbol{y}(k+1)|\boldsymbol{y}(k)) \\
&= \sum_{\boldsymbol{y}(k+1)\in\{0,1\}^M} P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1)) \\
&\quad \times \left\{ \boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k)) \otimes \boldsymbol{y}(k) \right\} \; p(\boldsymbol{y}(k+1)|\boldsymbol{y}(k)) \\
&= E^{\boldsymbol{y}(k+1)|\boldsymbol{y}(k)} \left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1)) \right. \\
&\quad \left. \times \left\{ \boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k)) \otimes \boldsymbol{y}(k) \right\} \right],
\end{aligned} \tag{12}$$

where

$$\boldsymbol{g}_{\boldsymbol{z}\boldsymbol{y}}(\boldsymbol{w}) = \left(g_{z_1}\boldsymbol{y}(\boldsymbol{w}_1), \ldots, g_{z_M}\boldsymbol{y}(\boldsymbol{w}_M)\right)^T$$

is the M dimensional vector and

$$g_{z}\boldsymbol{y}(\boldsymbol{w}) = \frac{\{z - f(\boldsymbol{w}^T\boldsymbol{y})\}f'(\boldsymbol{w}^T\boldsymbol{y})}{f(\boldsymbol{w}^T\boldsymbol{y})\{1 - f(\boldsymbol{w}^T\boldsymbol{y})\}}. \tag{13}$$

Here "$\otimes$" is the Kronecker product (See in Appendix). Therefore, the numerator $\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[\nabla_k P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]$ on the right hand side of (11) can be written

as

$$\mathrm{E}^{\boldsymbol{y}(k),\boldsymbol{y}(k+1)|\boldsymbol{x}^{(i)}}\left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))\ \left\{\boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k))\otimes\boldsymbol{y}(k)\right\}\right], \tag{14}$$

where $\mathrm{E}^{\boldsymbol{y},\boldsymbol{z}|\boldsymbol{x}}$ denotes the conditional expectation with respect to the pair $(\boldsymbol{y},\boldsymbol{z})$ for a given $\boldsymbol{x}$. The denominator $\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]$ on the right hand side of (11) is similarly rewritten as

$$\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]=\mathrm{E}^{\boldsymbol{y}(k),\boldsymbol{y}(k+1)|\boldsymbol{x}^{(i)}}\left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))\right]. \tag{15}$$

In our algorithm, the expecatation $\mathrm{E}^{\boldsymbol{y}(k),\boldsymbol{y}(k+1)|\boldsymbol{x}^{(i)}}$ is replaced by the Monte Carlo simulations as described in Section 4.2. A problem here is how to know the function $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))$ of $\boldsymbol{y}(k+1)$. Fortunately, we can make use of a recursive formula for such conditional likelihood

$$P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))=\mathrm{E}^{\boldsymbol{y}(k+2)|\boldsymbol{y}(k+1)}\left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+2))\right]. \tag{16}$$

This implies that it is better to save the function $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+2))$ in the step when the target layer is the $k+1$th layer. It is no other than the use of idea of back-propagation.

4.2. Use of Monte Carlo Simulations

Combining the results in the previous section, we see that the gradient $\nabla_k l$ is written as

$$\sum_{i=1}^{n} \frac{\mathrm{E}^{\boldsymbol{y}(k),\boldsymbol{y}(k+1)|\boldsymbol{x}^{(i)}}\left[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))\ \left\{\boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k))\otimes\boldsymbol{y}(k)\right\}\right]}{\mathrm{E}^{\boldsymbol{y}(k),\boldsymbol{y}(k+1)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))]}. \tag{17}$$

It would be better to employ the Monte Carlo simulations to evaluate the expectations which appear on the right hand side of (17). In the Monte Carlo simulations, enough number of realizations of $(\boldsymbol{y}(k), \boldsymbol{y}(k+1))$ are generated as the outputs of the kth layer and the $k+1$th layer for given input $\boldsymbol{x}^{(i)}$, $i = 1, \ldots, n$. Stochastic neuron in each layer up to the $k+1$th layer is simulated as binary random variables with probability $f(z)$ for the input z. The same principle can be applied for the evaluation of the Hessian $\nabla_k^2 l$. The details are given in Appendix 1.

4.3. Efficient Learning Algorithm

We are now ready to propose a learning algorithm based on the maximum likelihood principle.

STEP0

Set $k = K - 1$.

STEP1

Generate N realizations of $(\boldsymbol{y}(k), \boldsymbol{y}(k+1))$ for each $\boldsymbol{x}^{(i)}$, $i = 1, \ldots, n$. By making use of the saved values $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))$, $i = 1, \ldots, n$ in the previous STEP2 for $k > K-1$ and 1 otherwise, approximate $\nabla_k l$ and $\nabla_k^2 l$ by the results $\widetilde{\nabla_k l}$ and $\widetilde{\nabla_k^2 l}$ of the Monte Carlo simulations. Save the vector $\boldsymbol{w}(k) - \eta(\widetilde{\nabla_k^2 l})^{-1}\,\widetilde{\nabla_k l}$ for later use. If $k = 1$, update all weights $\boldsymbol{w}(k)$, $k = 1, \ldots, K-1$ by using the saved vectors, and exit when the convergence is obeserved, otherwise go to STEP0. Otherwise, go to STEP2.

STEP2

Generate N realizations of $\boldsymbol{y}(k+1)$ for all possible given values of $\boldsymbol{y}(k)$. Save the average of $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k+1))$ over the realizations as an approximation to $P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))$.

Decriment k to $k-1$ and go to STEP1.

Figure 2 may help the reader's understanding of our algorithm. The subnetwork in the top box is used for evaluating the conditional likelihood for the subnetwork in the bottom box. A practical problem would be the choice of the number N of the Monte Carlo simulations. Table I is the result of numerical experiments for the case of a multilayer feedforward stochastic neural network with $K = 7$ and $M = 4$. Test samples $(\boldsymbol{x}^{(i)}, \boldsymbol{t}^{(i)})$, $i = 1, \ldots, 500$ are generated by giving random

inputs, which are distributed as the standard normal distirbution, to the stochastic neural network with $K = 7$ and $M = 4$. The weights

Table I. The number of iterations needed for the convergence.

N: Number of Monte Carlo simulations	200	400	600	800	1000	1200
Number of iterations	328	291	184	185	183	183

$\boldsymbol{w}(1), \boldsymbol{w}(2), \ldots, \boldsymbol{w}(6)$ are also set by normal random numbers. The actual values of the weights are given in Appendix 2. As is seen from Table I, the number of iterations stays almost the same stable if $N \geq 600$. This observation suggests the choice $N = 600$ would be an appropriate choice at least for similar stochastic neural networks.

5. Mathematical Proof of Local Convergence of the Algorithm

To prove the convergence of the weight updated by the algorithm in Section 4.3, introduce a global gradient vector $d(W) = (\nabla_1 l(W)^T, \ldots, \nabla_{K-1} l(W)^T)^T$ and a global Hessian matrix $J(W) = (\nabla_i \nabla_j l(W);\ 1 \leq i, j \leq K-1)$ for whole weight vector $W = (\boldsymbol{w}(1)^T, \ldots, \boldsymbol{w}(K-1)^T)^T$. Here the notation $l(W)$ is used to emphasize

the dependency on W and

$$\nabla_i \nabla_j l(W) = \left(\frac{\partial^2}{\partial \boldsymbol{w}_m(i) \partial \boldsymbol{w}_{m'}(j)^T} l(W) \, ; \, 1 \leq m, m' \leq M \right).$$

By using a notation $H(W)$ for the diagonal block matrix with the same blocks as those of $J(W)$, the update formula (7) for each layer k is combined into

$$W - \eta \, H(W)^{-1} \, d(W). \tag{18}$$

In fact, our updating algorithm for each step is not exactly the same as Newton method, rather a diagonalized Newton method.

Assumption 1

All diagonal blocks in $H(W)$ are bounded above zero for any W, that is, $\| \nabla_k^2 l(W) \| > h$, $k = 1, \ldots, K-1$ for a positive constant h. Here

$$\| A \| = \sup_{\|\boldsymbol{x}\| \neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \tag{19}$$

is a matrix norm of a matrix A, which is defined through Euclidean norm $\|\boldsymbol{x}\| = (\sum_i x_i^2)^{\frac{1}{2}}$ for vector $\boldsymbol{x}$.

If the activation function $f(x)$ is a sigmoid function, Assumption 1 is always fulfilled. Otherwise, we may restrict a range of W to satisfy Assumption 1.

We need two more assumptions. One is to ensure the convexity of $l(W)$ in the neighborhood of W^* which is a solution of $d(W) = \mathbf{0}$. Another is to ensure the convergence of our algorithm. It might be

difficult to check if such assumptions hold true in practice. But we may make use of those assumptions as a check if the convergence is satisfactory or not.

Assumption 2

The matrix $J(W)$ is negative definite for any W.

This is a natural assumption at least in the neighborhood of the solution, since we are dealing with maximization problem of the likelihood function.

Assumption 3

The matrix $H(W)^{-1}J(W)$ is bounded above zero, that is,
$\| H(W)^{-1}J(W) \| > \lambda_0$ *for a positive constant* λ_0.

The following Theorem 5.1 ensures that the weights $W^{[t]} = (\boldsymbol{w}^{[t]}(1)^T, \ldots, \boldsymbol{w}^{[t]}(K-1)^T)^T$ updated in the tth iteration of the algorithm proposed in the previous section converges to the vector W^*. In the proof, we implicitly assume that the number N of Monte Carlo simulations are large enough, so that approximations $\widetilde{\nabla_k l}$ and $\widetilde{\nabla_k^2 l}$ can be replaced by $\nabla_k l$ and $\nabla_k^2 l$ respectively, due to law of large numbers.

THEOREM 5.1. *Under Assumptions 1, 2 and 3, $W^{[t]}$ converges to W^* as t tends to infinity for any $0 < \eta < \frac{1}{\lambda_0}$ as far as an initial weight*

$W^{[0]}$ is chosen so as

$$\sup_{\{W:\|W-W^*\|\leq\|W^{[0]}-W^*\|\}} \rho(W) < \frac{2h\lambda_0}{\|W^{[0]}-W^*\|}. \tag{20}$$

Here $\rho(W) = \sum_{i=1}^{M^2(K-1)} \left\| \frac{\partial}{\partial W_i} J(W) \right\|$.

Proof

To prove a convergence, we first show that

$$\begin{aligned} \|W^{[t+1]} - W^*\| \leq \quad & (1-\eta\,\lambda_0)\,\|W^{[t]} - W^*\| \\ & +\eta\,\frac{\rho((1-\theta)W^* + \theta W^{[t]})}{2h}\,\|W^{[t]} - W^*\|^2 \end{aligned} \tag{21}$$

holds true for a constant $0 \leq \theta \leq 1$. Since

$$W^{[t+1]} = W^{[t]} - \eta\,H(W^{[t]})^{-1}\,d(W^{[t]}), \tag{22}$$

the difference between $W^{[t+1]}$ and W^* is rewritten as

$$\begin{aligned} W^{[t+1]} - W^* = \quad & W^{[t]} - W^* \\ & -\eta H(W^{[t]})^{-1}\left\{-J(W^{[t]})(W^* - W^{[t]})\right. \\ & \left.+J(W^{[t]})(W^* - W^{[t]}) + d(W^{[t]})\right\} \\ = \quad & \left\{I - \eta H(W^{[t]})^{-1} J(W^{[t]})\right\}(W^{[t]} - W^*) \\ & -\eta H(W^{[t]})^{-1}\left\{J(W^{[t]})(W^* - W^{[t]}) + d(W^{[t]})\right\}. \end{aligned}$$

From the definition of λ_0, we see that

$$\left\| I - \eta\,H(W^{[t]})^{-1}\,J(W^{[t]}) \right\| \leq 1 - \eta\,\lambda_0. \tag{23}$$

On the other hand, in the Taylor expansion,

$$\mathbf{0} = d(W^*) = d(W^{[t]}) + J(W^{[t]})\,(W^* - W^{[t]}) + \text{Residual},$$

the ith element of the residual

$$J(W^{[t]})(W^* - W^{[t]}) + d(W^{[t]}) \tag{24}$$

is precisely written as

$$-\frac{1}{2}(W^* - W^{[t]})^T \left.\frac{\partial}{\partial W_i} J(W)\right|_{W=(1-\theta)W^*+\theta W^{[t]}} (W^* - W^{[t]}) \tag{25}$$

for a constant $0 \leq \theta \leq 1$. Therefore,

$$\left\| J(W^{[t]})(W^* - W^{[t]}) + d(W^{[t]}) \right\| \leq \frac{1}{2}\,\rho((1-\theta)W^*+\theta W^{[t]})\,\|W^{[t]}-W^*\|^2 \tag{26}$$

holds true and inequality (21) follows.

To prove a convergence of the weight vector $W^{[t]}$ to the vector W^*, we rewrite (21) as

$$\frac{\|W^{[t+1]} - W^*\|}{\|W^{[t]} - W^*\|} \leq 1 - \eta\lambda_0 \left\{ 1 - \rho((1-\theta)W^* + \theta W^{[t]})\,\frac{\|W^{[t]} - W^*\|}{2h\lambda_0} \right\}. \tag{27}$$

We can find an $\varepsilon > 0$ such that

$$\sup_{\{W\,:\,\|W-W^*\| \leq \|W^{[0]}-W^*\|\}} \rho(W)\,\frac{\|W^{[0]} - W^*\|}{2h\lambda_0} < 1 - \varepsilon, \tag{28}$$

from the condition (20), so that

$$\rho((1-\theta)W^* + \theta W^{[0]})\frac{\|W^{[0]} - W^*\|}{2h\lambda_0} < 1 - \varepsilon$$

follows from the fact that

$$\|(1-\theta)\,W^* + \theta\,W^{[0]} - W^*\| = \theta\,\|W^{[0]} - W^*\| \leq \|W^{[0]} - W^*\|. \quad (29)$$

Therefore

$$\frac{\|W^{[1]} - W^*\|}{\|W^{[0]} - W^*\|} < 1 - \eta\lambda_0\varepsilon \quad (30)$$

follows. In a similar way, we can show that

$$\frac{\|W^{[t+1]} - W^*\|}{\|W^{[t]} - W^*\|} < 1 - \eta\lambda_0\varepsilon \quad (31)$$

holds true for any $t \geq 1$. In fact, suppose that the inequality (31) holds true for any $0 \leq t \leq s$, then

$$\begin{aligned} & \rho((1-\theta)W^* + \theta W^{[s+1]})\,\frac{\|W^{[s+1]} - W^*\|}{2h\lambda_0} \\ < & \sup_{\{W\,:\,\|W-W^*\| \leq \|W^{[0]}-W^*\|\}} \rho(W)\,\frac{\|W^{[0]} - W^*\|}{2h\lambda_0} \\ < & \ 1-\varepsilon. \end{aligned}$$

This implies that the inequality (31) holds true for $t = s+1$. We now established Theorem 5.1. □

The proof of Theorem 5.1 also implies that the rate of convergence is $1-\eta\lambda_0\varepsilon$, which depends on the choice of the initial weight $W^{[0]}$. In (20), the left hand side of the inequality is a monotone increasing function of $\|W^{[0]} - W^*\|$ and the right hand side is a monotone decreasing function of $\|W^{[0]} - W^*\|$, so that the condition is fulfilled as far as the initial weight $W^{[0]}$ is not far from W^*. Of course, a closer choice of $W^{[0]}$ to W^* yields a quicker convergence.

6. A Practical Application

We will show an example of practical application, the prediciton of the fall or the rise of the closing values $\{s_k\}$ of TOPIX for seven years from January 1st of 1994 to December 31st of 2000. We interpolated the missing values for market holidays by linear interpolation. The data used here is available from

http://www.stat.math.keio.ac.jp/DandDIII/Examples/TOPIX1991-2002.dad

together with the interpolated data. We will use the first 3 years' data as a training data and the rest as a test data. In this example, we concentrate our attention into predicting the fall or the rise of the return, in other word, positiveness or negativeness of the log return,

$$x_k = \log \frac{s_k}{s_{k-1}}. \tag{32}$$

A traditional model used for prediction is an autoregressive model (AR),

$$x_k = \beta_0 + \sum_{j=1}^{p} \beta_j x_{k-j} + \varepsilon_k. \tag{33}$$

The best predictor obtained by fitting an AR model to the training data is

$$\widehat{x}_k = -0.00002 + 0.42931 x_{k-1} - 0.23617 x_{k-2} + 0.04125 x_{k-3}. \tag{34}$$

The fall or the rise is predicted by the sign of this predictor with 70.357 % accuracy.

For such a time series, time-delay feedback type neural network is frequently used (for example in Beltrametti et al, 1997). Also applications of time-delay feedback type Boltzmann machine (Hinton et al., 1983) are reported in Darbellay et al. (2000) or in Kaizoji (2000). The Boltzmann machine looks like the multilayer feedforward stochastic neural network, and all neurons in Boltzmann machine are stochastic and connected to each other in a bi-directional way.

We now compare theses models with the multilayer feedforward stochastic neural network (MFSNN). It would be natural to train a time-delay feedback type 3-3-1 non-stochastic neural network (NN) in view of the fitted AR model above. The same type 3-3-1 MFSNN is trained for comparison, although the three input neurons are deterministic in this case. Similarly we train Boltzmann machine with seven neurons, in which three neurons accept the delayed inputs and one neuron emits the output. The unit 0 neuron is always affixed to each neuron. For convenience we assume that each neuron in BM or MFSNN emits 1 or -1 instead of 1 or 0. The output is then compared with the sign of the actual log return. Since the output of NN is the predicted value of the log return, the sign is compared with that of the actual log return. In our experiments, the weights are converged after 119

iterations for NN and 257 iterations for MFSNN with the learning rate $\eta = 1 \,/\, \left(\left\| H(W^{[t]})^{-1} J(W^{[t]}) \right\| + 0.001 \right)$. The convergence has been observed after 1907 iterations for BM. The tolerance for $||W^{[t+1]} - W^{[t]}||$ is taken the same 10^{-7} for all three networks. For BM, the criterion function to be minimized is the potential function and the stochastic behavior of each neuron is controlled by a global constant T, called temperature, given by $T = c \,/\, \log(t+1)$ as an increase of where T decreases as t, the number of iterations increases. In our experiments we have taken c to be 5.

The results are summarized in Table II. The accuracy of BM and

Table II. Accuracy of prediction of the fall or the rise of the training data.

Model	Accuracy (%)	
	Mean	Standard deviation
AR	70.357	0
NN	72.914	0
BM	71.769	0.509
MFSNN	69.212	0.658

MFSNN does not exactly remain the same for each experiments because of its stochastic nature, so that the mean and the standard deviation of 200 times experiments are shown. The standard deviation for NN

is always 0 because of deterministic nature of NN. In terms of the accuracy, NN is the best, and BM follows. It is worthy of noting that NN or BM a little bit improves the accuracy of AR but not so much.

The mean accuracy of MFSNN can be improved to 70.011 % with the standard deviation 0.416 % by disconnecting some connections in the network. The disconnected MFSNN is illustrated in Figure 4, which reflects time lags of the input. The first neuron in the hidden layer collects all values, the second collects the values of the lag 1 and 2, and the third does the value of the lag 1 only. In fact, such modification of the network does not much improve the accuracy, but it would be better for understanding the meaning of the fitted model. We will use such a modified MFSNN in the following comparison. We now show the accuracies for the test data in Table III. The order of the accuracies is now reversed. The mean accuracy of NN is poor, not only worse

Table III. Accuracy of prediction of the fall or the rise of the test data.

Model	Accuracy (%)	
	Mean	Standard deviation
AR	51.377	0
NN	49.841	0
BM	55.474	0.392
MFSNN	60.193	0.489

than the 51.377 % of AR, but also worse than the 50.093 % of the coin-tossing. By disconnecting some connections in NN as same as in MFSNN, the accuracy is a little bit improved but only up to 50.029 %. This result suggests that the improvement of the accuracy is not only due to the way of connection or to the non-linearity of the networks, but to the stochastic nature of MFSNN or of BM. In other words, stochastic nature of MFSNN or of BM is indispensable to get a stable prediction of the fall or the rise of TOPIX. The BM is stochastic but too complicated to get a simple prediction like the fall or the rise. In fact, it is hard to understand the role of the weights of BM shown in Figure 3, but easier to understand the weights of MFSNN in Figure 4.

Roughly speaking, in Figure 4 the hidden neuron 3 tends to emit 1 as an increase of the log return of yesterday and the hidden neuron 2 tends to emit 1 as a decrease of the log return of before yesterday. The hidden neuron 1 plays a role of global compensation since all inputs are connected to it. The outputs of the neurons in the hidden layer are then combined into an output 1 or -1 of the network. The probability of the output being 1 is

$$f\left(1.1746\ y_1(2) + 1.2054\ y_2(2) + 1.9541\ y_3(2) + 0.0001\right). \qquad (35)$$

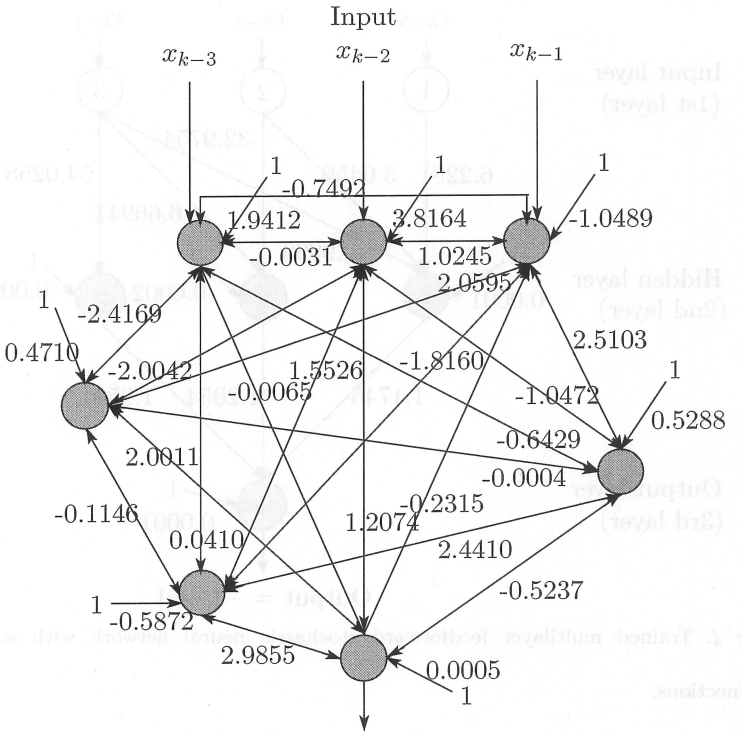


Figure 3. Trained Boltzmann machine at the convergence.

7. Concluding Remarks

We have proposed the learning algorithm for the multilayer feedforward stochastic neural network based on the maximum likelihood principle. The likelihood principle would be a most natural way of making full use of stochastic nature of the network. It is a key principle already well known in a long history of statistics. Unfortunately, no direct application of the ordinary back-propagation algorithm is possible because

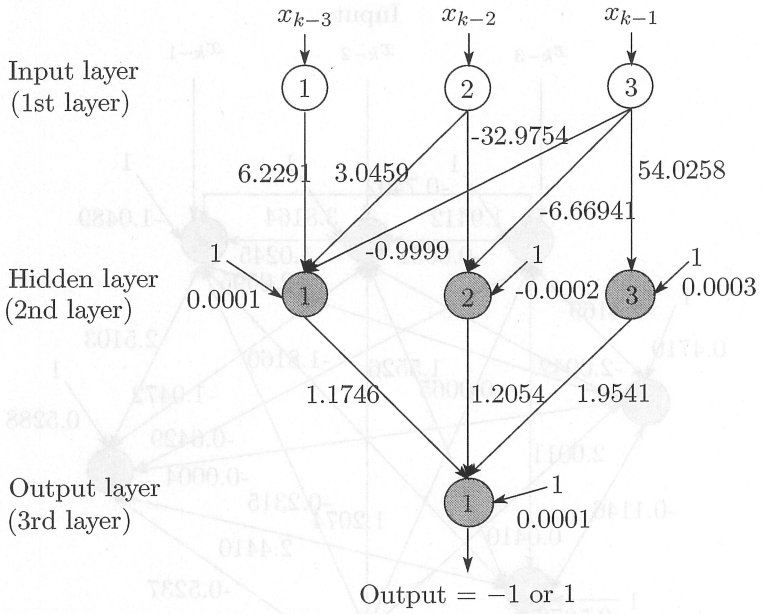


Figure 4. Trained multilayer feedforward stochastic neural network with some disconnections.

of the stochastic nature of the network. What we have shown here is that a similar but different algorithm can be constructed by making use of conditional likelihood, which is evaluated by the Monte Carlo simulations. The simplicity of the proof of the convergence in Theorem 5.1 is dismissed if the weight were updated layer by layer in STEP1, but the convergence itself still holds true under the same assumptions and conditions as in Theorem 5.1.

A practical application of the multilayer feedforward stochastic neural network to TOPIX data shows that our stochastic neural network is advantageous over other networks, non-stochastic neural network or the

Boltzmann machine. The reason is that non-stochastic neural network or the Boltzmann machine tends to overfit to the data, so that it works well for the training data but does not for the test data. The stochastic neural network works better for the test data, since it learns well only a stable stochastic mechanism behind the TOPIX data.

Appendix

Evaluation of Hessian $\nabla_k^2 l$

The Hessian $\nabla_k^2 l$ is rewritten as

$$\nabla_k^2 l = \sum_{i=1}^{n} \left\{ H_k^{(i)} - G_k^{(i)} \left(G_k^{(i)} \right)^T \right\}, \tag{36}$$

where

$$G_k^{(i)} = \frac{\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[\nabla_k \, P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]}{\mathrm{E}^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}[P(\boldsymbol{t}^{(i)}|\boldsymbol{y}(k))]} \quad \text{and} \quad H_k^{(i)} = \frac{E^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}\left[\nabla_k^2 L_{\boldsymbol{y}(k)}^{(i)}\right]}{E^{\boldsymbol{y}(k)|\boldsymbol{x}^{(i)}}\left[L_{\boldsymbol{y}(k)}^{(i)}\right]}. \tag{37}$$

Note that $G_k^{(i)}$ is a formula which appears the gradient vector $\nabla_k l$ in the formula (11) in Section 4.1.

The denominator for the right hand side of (37) is the same denominator of (11) in Section 4.1, then the same discussion is available in view of Section 4.1.

The numerator for the right hand side of (37) becomes

$$\mathrm{E}^{(\boldsymbol{y}(k),\boldsymbol{y}(k+1))|\boldsymbol{x}^{(i)}} \left[L^{(i)}_{\boldsymbol{y}(k+1)} \cdot \left\{ \boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k))\, \boldsymbol{g}_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k))^T + H_{\boldsymbol{y}(k+1)\boldsymbol{y}(k)}(\boldsymbol{w}(k)) \right\} \otimes \boldsymbol{y}(k)\boldsymbol{y}(k)^T \right],$$

where $H_{\boldsymbol{z}\boldsymbol{y}}(\boldsymbol{w}) = \mathrm{diag}\,(h_{z_1\boldsymbol{y}}(\boldsymbol{w}_1), \ldots, h_{z_M\boldsymbol{y}}(\boldsymbol{w}_M))$ with

$$h_{z\boldsymbol{y}}(\boldsymbol{w}) = \begin{cases} (\log f)''(\boldsymbol{w}^T\boldsymbol{y}), & \text{for} \quad z = 0, \\ (\log(1-f))''(\boldsymbol{w}^T\boldsymbol{y}), & \text{for} \quad z = 1. \end{cases} \tag{38}$$

Here $A \otimes B$ is the Kronecker product of $m \times n$ matrix $A = (a_{ij})$ and matrix $B = (b_{kl})$, that is,

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}. \tag{39}$$

From these results, it is avairable to update the weights $\boldsymbol{w}(k)$ layer by layer, and the Hessian can be also evaluated by performing the Monte Carlo simulations in order to evaluate the conditional expectation $\mathrm{E}^{(\boldsymbol{y}(k),\boldsymbol{y}(k+1))|\boldsymbol{x}^{(i)}}$, in which enough number of realizations are generated $(\boldsymbol{y}(k), \boldsymbol{y}(k+1))$ for given input $\boldsymbol{x}^{(i)}$, $i = 1, \ldots, n$ to the network.

The values of weights $\boldsymbol{w}(1), \ldots, \boldsymbol{w}(6)$ used for the experiments in Section 4.3.

$w(1)$	$w(2)$	$w(3)$	$w(4)$	$w(5)$	$w(6)$
-0.1394612	-0.31984159	-0.3676139	0.6240228	-0.5533617	-0.4619993
0.4074307	0.44929164	0.3323727	0.5571762	-0.4331631	-0.9625873
1.0929253	-0.21606808	1.8233537	0.7944775	1.1434630	0.3762323
0.8587742	-0.06870566	0.8729137	2.3449648	1.6318256	1.4207603
-1.5765343	0.05035160	-0.5275496	0.7631042	-0.1080997	-0.7926528
-1.5010325	0.33342176	-0.7479537	0.2530770	1.7911563	-1.5552566
0.2928568	0.15244506	0.5286420	-0.1386473	-1.1828238	0.7196766
0.3518486	-0.22013318	-1.1501037	-0.9877117	-1.5925307	-0.3265978
-1.9359147	-1.19125343	-0.7772961	-0.4056741	1.5179803	0.6359205
0.9102182	-0.55452458	0.7302553	-2.2274971	-0.3563535	0.4050059
1.1954429	0.21735754	-2.0034462	0.9667034	-1.2605193	-0.4129074
-0.3104301	0.22177094	-0.9292310	0.1050608	-1.4063022	-2.0700562
-1.8160673	1.37068625	1.5671901	0.6085276	0.4272885	-0.5374074
1.6310159	0.78176526	-1.9807646	1.3572921	0.4342217	0.2454906
-0.5665352	-1.74834072	-0.2512323	1.7693943	0.6685098	-0.1412052
-0.6258691	-1.66897517	-1.0507641	0.2500595	-1.0334830	-1.4387635

References

Amari, S. *Mathematical methods of neurocomputing*, O. E. Barndorff-Nielsen, J. L. Jensen and W. S. Kendall, editors, *Networks and Chaos-Statistical and Probabilistic Aspects*, (pp. 1–39), Chapman&Hall, 1993.

Amari, S. The EM algorithm and information geometry in neural network learning, *Neural Computation*, 7(1):13–18, 1995.

Amari, S. Natural gradient works efficiently in learning, *Neural Computation*, 10:251–276, 1998.

Amari, S., Park, H. and Fukumizu, K. Adaptive method of realizing natural gradient learning, *Neural Computation*, 12:1399–1409, 2000.

Baba, N. Construction of a decision support system for dealing stocks; Utilizing neural networks. *MTEC Journal*, 11:3–41, 1998.

Beltrametti, L., Fiorentini, R., Marengo, L and Tamborini, R. (1997). A learning-to-forecast experiment on the foreign exchange market with a classifier system, *Journal of Economic Dynamics and Control*, 21:1543–1575, 1997.

Chen, S., Cowan C.F.N., and Grant, P. M. Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Transactions on Neural*

Networks, 2(2):302-309, 1991.

Darbellay, G. and Slama, M. Forecasting the short-term demand for electricity, *International Journal of Forecasting*, 16:71–83, 2000.

Fisher, R. A. On the Mathematical Foundations of Theoretical Statistics, *Phil. Trans. Royal Soc.*, Series A, 222–326, 1922.

Goldberg, D. E. *Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

Hassoun, M. H. *Fundamentals of Artificial Neural Networks*, MIT Press, 1995.

MacKay, D. J. C. *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003.

Medgassy, P. *Decomposition of superposition of distributed functions*, Hungarian Academy of Science, Budapest, 1961.

Micchelli, C. A. Interpolation of scattered data: Distance and conditionally positive definite functions, *Constructive Approximation*, 2:11–22, 1986.

Haykin, S. *Neural Networks-A Comprehensive Foundation,* Second Edition, (pp. 309–316), IEEE Press, 1999.

Hinton, G. E. and Sejnowski, T. J. *Optimal Perceptual Inference*, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (Washington 1983), 448–453, 1984.

Kaizoji, T. Speculative bubbles and crashes in stock markets: an interacting-agent model of speculative activity, *Physica A*, 287:493–506, 2000.

Kurita, T. A method to determine the number of hidden units of three layers networks by information criteria. *IEICE*, D-II, J73-D-II, 1872–1878, 1990.

Larson, J. H. *Introduction of probability theory and statistical inference*, John-Wiely & Sons, USA, 1969.

Montanari, U. Networks of constraints, fundamental properties and applications to picture processing, *Information Science*, 7:95–132, 1974.

Pearl, J. *Probabilistic reasoning in intelligent systems*, (pp. 239–288), Morgan Kaufmann publishers, 1988.

Powel, M. J. D. Radial basis functions for multivariate intepolation: A review, in *Algorithms for the Approximation of Functions and Data*, J. C. Mason and M. G. Cox, eds., Clarendon Press, Oxford, England, 1987.

Rumelhart, D. E., Hinston, G. E. and Williams, R. J. *Learning internal representations by error propagation,* Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1, (pp. 318–362),

MIT Press, 1986.

Address for Offprints: Department of Mathematics, Keio University,

3-14-1 Hiyoshi Kohoku Yokohama,

223-8522,

Japan

abstract

Efficient learning algorithm based on maximum likelihood principle is proposed for {\it multilayer feedforward stochastic neural network}, in which each neuron
emits 0 or 1 according to a conditional probability determined by a linear combination of given inputs. To reduce the burden of computation, the log likelihood is approximated by an average of the log conditional likelihoods, which are evaluated by the Monte Carlo simulations. Also the back-propagation algorithm for such a stochastic neural network has been developed. It is proved that the connection weights among neurons updated by this algorithm converge to the weights which maximize the likelihood of the whole network, as the number of iterations increases.
The advantages of using the multilayer feedforward stochastic neural network over deterministic neural network or Boltzmann machine are shown by giving a practical example of predicting the fall or the rise of the Tokyo Stock Price Index.