Journal of the Operations Research Society of Japan
Vol. 43, No. 4, December 2000

© 2000 The Operations Research Society of Japan

A NEW STOCHASTIC LEARNING ALGORITHM FOR NEURAL NETWORKS

Masato Koda
University of Tsukuba

Hiroyuki Okano
IBM Research, Tokyo Research Laboratory

(Received October 6, 1999; Final July 7, 2000)

Abstract A new stochastic learning algorithm using Gaussian white noise sequence, referred to as Subconscious Noise Reaction (SNR), is proposed for a class of discrete-time neural networks with time-dependent connection weights. Unlike the back-propagation-through-time (BTT) algorithm, SNR does not require the synchronous transmission of information backward along connection weights, while it uses only ubiquitous noise and local signals, which are correlated against a single performance functional, to achieve simple sequential (chronologically ordered) updating of connection weights. The algorithm is derived and analyzed on the basis of a functional derivative formulation of the gradient descent method in conjunction with stochastic sensitivity analysis techniques using the variational approach.

1. Introduction

Development of computer algorithms for artificial neural networks started with a fundamental idea borrowed from studies of the brain. Based on a computing model similar to the underlying structure of the brain, neural networks share the brain's ability to learn and adapt in reaction to external and/or internal inputs, and, over time, the neural network has itself become an accepted metaphor for how the brain actually works. Sophisticated back-propagation algorithms [17] are widely used for many practical applications, and the technology has now reached a level at which neural computing (computing based on a brain-like model) is both possible and practical.

Neural networks, for instance, have also been considered to provide existential support for some models of the organized mind; Hebb's assumption about the plasticity of synapses [5] was verified on neural networks before actual biological evidence was found in the brain [7]. However, although neural networks have demonstrated emergent intelligent behavior, they continue to face major criticism from neuro-scientists. Francis Crick, for example, pointed out that back-propagation, which requires transmission of information backward along axons, is highly unrealistic [2]. What the researchers seem to have overlooked, in the authors' opinion, is the possible existence of a human-learning or information-preprocessing mechanism that operates largely at a subconscious level without a (conscious) back-propagation mechanism.

This paper proposes a new noise-based learning algorithm that does not require the backward transmission of sensitivity information. The algorithm takes advantage of ubiquitous noise inherent in each node (neuron) and updates connection weights using only the noise, the signal from each node, and the overall objective of the network. The process of updating connection weights may be referred to as *reaction*, because it resembles and emulates an autonomous reaction of each neuron in response to noise signals and the network objective. A major characteristic of the algorithm is that updating is done locally without

synchronous transmission of information backward along network connections. The proposed algorithm, therefore, is referred to as *Subconscious Noise Reaction (SNR)*[1] , since it may offer some analogy to a possible information preprocessing mechanism in the brain that operates largely at a subconscious level.

There has been an earlier effort to design and analyze stochastic neural networks that function in the presence of random perturbations. Kosko [14], for example, extended the standard deterministic stability theory for recurrent networks (e.g., [1, 4] and references therein) to continuous networks whose dynamics are driven by additive Brownian motion. An attempt by Matsuoka [15] to design a stochastic learning method showed the technical potential of noise-based learning for a class of layered networks.

In the stochastic sensitivity analysis area, on the other hand, a sensitivity theory for a general class of stochastic kinetic systems, employing Green's function method, has been described in Dacol and Rabitz [3]. Similarly, Koda [9] proposed a likelihood ratio method to compute the sensitivity information on a temporal evolution of the probability density function by using a diffusionless Fokker-Planck equation associated with the dynamic systems. These studies reveal that the stochastic sensitivity analysis can be a powerful tool for estimating the gradient information of stochastic neural networks.

In the prior exposition, Koda [10, 11, 12] derived a comprehensive framework of stochastic sensitivity analysis for the neural learning of a class of continuous recurrent networks whose dynamics are driven by an additive Gaussian white noise process. The purpose of this paper is to extend the earlier continuous results to general cases of discrete-time stochastic neural networks with time-dependent connection weights, and derive Subconscious Noise Reaction (SNR) algorithm.

The derived SNR algorithm does not involve differentiation of a *signal function* or summation over connection weights, where the signal function is a transmitting function at each node. The simplicity of the SNR gives it practical importance; a non-differentiable signal function, such as a step (threshold) function, may be used, and there is no need to propagate error sensitivities along connection weights. These properties make it suitable for hardware (chip) implementations, and applicable to dynamic networks in which network connections are generated and deallocated dynamically at each node independent of a learning algorithm. Note that the standard back-propagation-through-time (BTT) algorithm requires a differentiable signal function and a static network.

Numerical experiments using the SNR and BTT algorithms based on the three-layer feedforward networks, $N(3;3;1)$ and $N(7;n;1)$, reveal that the SNR has basically the same characteristics (in a stochastic sense) as the BTT. The experiments also show that the SNR achieves a learning with a step function, a simple non-differentiable function, with which the BTT does not work. Note that the SNR algorithm proposed in this paper is applicable to most kinds of empirical learning for both layered or recurrent neural networks, which can model types of problems in operations research; for example, clustering problems or combinatorial optimization problems [6].

In Section 2, a discrete-time stochastic neural network system with additive Gaussian white noise is defined. In Section 3, a functional derivative formulation is proposed for the gradient descent learning of time-dependent connection weights. The variational approach to stochastic sensitivity analysis is also described, and a fundamental sensitivity relation is obtained by using the concept of discrete-time functional derivative sensitivity coefficients. In Section 4, a comprehensive theoretical framework for stochastic neural learning is

[1] SNR is also referred to as Stochastic Noise Reaction in other publications.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

obtained and the SNR algorithm is derived. In Section 5, the application of the SNR algorithm to general layered networks is illustrated, and numerical experiments on the learning of OR, XOR, and mirror symmetry (SYM) patterns are reported. The results are compared with those obtained by using the standard back-propagation algorithm. Finally, conclusions are summarized in Section 6.

2. Discrete-Time Stochastic Neural Networks

In this study, a class of discrete-time recurrent neural networks with n units is considered, which is described by the following difference equations (with time treated as an integer variable):

$$x_i(t+1) = \sum_j w_{ij}(t) S_j(t) + \xi_i(t+1), \quad i,j = 1,2,\ldots,n, \tag{1}$$

where $x_i(t)$ denotes the internal state of the i-th unit at time t, $w_{ij}(t)$ is the weight of the time-dependent connection between the i-th and the j-th units, $S_j(t)$ is the signal at the j-th unit, and $\xi_i(t+1)$ is the noise associated with the i-th unit at time $t+1$. The initial data, $x_i(0) = x_i^o$ at $t = 0$, is assumed to be given. For ease of exposition, the non-linear signal function is specified as

$$S_i(t) = \tanh[x_i(t)] = \frac{1 - \exp(-2x_i(t))}{1 + \exp(-2x_i(t))}, \tag{2}$$

where tanh is the sigmoid-shaped function on $[-1,+1]$-interval.

In the subsequent development, a discrete-time model is consistently assumed; however, many of the techniques described in this study can also be applied and carried over to continuous-time recurrent networks [4]. Note that the discrete-time model, in principle, can be obtained directly from a continuous model by using standard (temporal) discretization techniques or by (spatially) unfolding a continuous recurrent network into a multi-layered feedforward network that evolves by one layer at each time step.

Injection of Gaussian white noise (with zero mean and unit variance) is assumed in (1); i.e.,

$$E[\xi_i(t)] = 0, \quad E[\xi_i(t)\xi_j(s)] = \delta_{ij}\delta_{ts}, \tag{3}$$

where $E[\cdot]$ denotes the expectation operator and δ_{ij} and δ_{ts} denote the Kronecker delta. The noise $\xi_i(t)$ can be considered either as a network bias, an (externally controlled) artificial noise signal, or an intrinsic system noise (e.g., a thermal noise). Hence, the (n-dimensional) internal state vector $x(t)$ becomes a stochastic variable, since a degree of uncertainty is induced by the additive noise sequence $\xi(s)$ ($s = 1,2,\ldots,t$). It should be noted that the stochastic network defined by (1)-(3) reduces to the ordinary multi-layered network if the stochastic state variables are replaced by their expected values. The present approach is in contrast to simulated annealing [8], where the Boltzmann distribution is used to implement stochastic algorithms.

3. Stochastic Sensitivity Analysis

In this section, a functional derivative formulation is proposed for the gradient descent learning of time-dependent connection weights. A variational approach for stochastic sensitivity analysis of the network is then described, and a concept of discrete-time functional derivatives for obtaining fundamental sensitivity relations is proposed.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

3.1. Functional derivative formulation of gradient method

Let the performance functional for a network to be minimized be $L[x(T)]$, where $t = T$ denotes the time of interest, which is usually some final time, although this final time could itself be arbitrary. Since the connection weights $w_{ij}(t)$ are time-dependent, and they are correlated against a single performance functional $L[x(T)]$ at time T, a relevant gradient descent method may be formulated as follows:

$$w_{ij}^{\tau+1}(t) = w_{ij}^{\tau}(t) - \mu \frac{\delta L[x(T)]}{\delta w_{ij}^{\tau}(t)}, \tag{4}$$

where τ denotes the index for the learning time and μ is the learning rate, which is usually a small positive number ($\mu > 0$). In (4), $\frac{\delta L[x(T)]}{\delta w_{ij}^{\tau}(t)}$ denotes the functional derivative (see, e.g., [13]) since the gradient computation in weight space involves multiple time variables t and T ($t < T$).

Thus, the central role in this study is played by functional derivative sensitivity coefficients, defined formally as first-order derivatives of the performance functional $L[x(T)]$ with respect to the time-dependent parameters $w_{ij}^{\tau}(t)$, which are simply denoted by $\frac{\delta L[x(T)]}{\delta w_{ij}(t)}$. Further, assuming that the performance functional $L[x(T)]$ has an implicit dependence on $w_{ij}(t)$ only through $x(T)$, $\frac{\delta L[x(T)]}{\delta w_{ij}(t)}$ can be expressed as

$$\frac{\delta L[x(T)]}{\delta w_{ij}(t)} = \sum_k \frac{\partial L[x(T)]}{\partial x_k(T)} \frac{\delta x_k(T)}{\delta w_{ij}(t)}. \tag{5}$$

Equation (5) implies that the state sensitivity coefficients, which are defined as first-order functional derivatives of the internal state $x_k(T)$ with respect to the connection weights $w_{ij}(t)$, i.e., $\frac{\delta x_k(T)}{\delta w_{ij}(t)}$, contain all the information needed for the neural learning of $w_{ij}(t)$.

In the present stochastic sensitivity analysis, network performance is evaluated by the average value of $L[x(T)]$:

$$E[L[x(T)]] = \int_X L(x) p(x, T) dx, \tag{6}$$

where $p(x, T)$ denotes the probability density function and X is a set of random variables associated with the underlying stochastic sequence $\xi(t)$. Thus, a gross performance index for the stochastic network can be defined and given by (6). In practice, however, a suitably defined statistical average of $L[x(T)]$ may be used for (6).

Taking the functional derivative of (6) with respect to $w_{ij}(t)$, the performance sensitivity coefficient is obtained as follows:

$$\frac{\delta E[L[x(T)]]}{\delta w_{ij}(t)} = E\left[\frac{\delta L[x(T)]}{\delta w_{ij}(t)}\right] = E\left[\sum_k \frac{\partial L[x(T)]}{\partial x_k(T)} \frac{\delta x_k(T)}{\delta w_{ij}(t)}\right], \tag{7}$$

where (5) has been used, and it is assumed that the operations of (functional derivative) differentiation and expectation are interchangeable. Therefore, in the functional derivative formulation of the gradient descent method (4), it is needed to determine the state sensitivity coefficients, $\frac{\delta x_k(T)}{\delta w_{ij}(t)}$.

3.2. Variational approach

Sensitivity problems for stochastic neural networks can be solved by using the variational approach developed and typically demonstrated for continuous-time neural networks by one

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

of the present authors (e.g., [10, 11]). In this study, a discrete-time neural network defined by the stochastic difference equations (1) and (2) is considered by using the same approach.

Simultaneous perturbations of the connection weights, including noise signals, cause perturbations in the system states. Thus variations, $\delta w_{jk}(t)$ and $\delta\xi_j(t+1)$, result in a variation $\delta x_j(t+1)$ for $t = 0, 1, 2, \ldots, T-1$. To first-order, relations for these variations are obtained as

$$\begin{aligned} \delta x_j(t+1) &= \textstyle\sum_k w_{jk}(t) S'_k(t) \delta x_k(t) \\ &+ \textstyle\sum_k \delta w_{jk}(t) S_k(t) + \delta\xi_j(t+1), \end{aligned} \tag{8}$$

where

$$S'_k(t) = \frac{\partial S_k(t)}{\partial x_k} = \{1 + S_k(t)\}\{1 - S_k(t)\}. \tag{9}$$

Adjoint variables $\phi_i(t)$ $(i = 1, 2, \ldots, n)$ are now introduced, and (8) is multiplied by $\phi_j(t+1)$ to obtain

$$\begin{aligned} \textstyle\sum_j \phi_j(t+1)\delta x_j(t+1) &= \textstyle\sum_j \sum_k w_{jk}(t)\phi_j(t+1) S'_k(t)\delta x_k(t) \\ &+ \textstyle\sum_j \sum_k \delta w_{jk}(t)\phi_j(t+1) S_k(t) \\ &+ \textstyle\sum_j \phi_j(t+1)\delta\xi_j(t+1). \end{aligned} \tag{10}$$

Summation of (10) from $t = 0$ to $t = T-1$ yields

$$\begin{aligned} \textstyle\sum_{t=0}^{T-1}\sum_j \phi_j(t+1)\delta x_j(t+1) &= \textstyle\sum_{t=0}^{T-1}\sum_j \sum_k w_{jk}(t)\phi_j(t+1) S'_k(t)\delta x_k(t) \\ &+ \textstyle\sum_{t=0}^{T-1}\sum_j \sum_k \delta w_{jk}(t)\phi_j(t+1) S_k(t) \\ &+ \textstyle\sum_{t=0}^{T-1}\sum_j \phi_j(t+1)\delta\xi_j(t+1). \end{aligned} \tag{11}$$

Further manipulation of (11) and an appropriate arrangement of terms lead to

$$\begin{aligned} \textstyle\sum_j \phi_j(T)\delta x_j(T) = &- \textstyle\sum_{t=1}^{T-1}\sum_j [\phi_j(t) - S'_j(t)\sum_k w_{kj}(t)\phi_k(t+1)]\delta x_j(t) \\ &+ \textstyle\sum_j \sum_k \sum_{t=0}^{T-1} \delta w_{jk}(t)\phi_j(t+1) S_k(t) \\ &+ \textstyle\sum_j \sum_{t=1}^{T} \phi_j(t)\delta\xi_j(t), \end{aligned} \tag{12}$$

where the relation $\delta x_j(0) = 0$ and the formal translation $\sum_j \sum_k w_{jk}\phi_j(t+1)S'_k(t)\delta x_k(t) = \sum_k \sum_j w_{kj}\phi_k(t+1)S'_j(t)\delta x_j(t)$ have been used.

Let $\phi_j(t)$ be governed formally by the following adjoint equations:

$$\phi_j(t) = S'_j(t)\sum_k w_{kj}(t)\phi_k(t+1), \quad t = T-1, T-2, \ldots, 1 \tag{13}$$

subject to the terminal data

$$\phi_j(T) = \delta_{ij}, \quad i, j = 1, 2, \ldots, n, \tag{14}$$

where i represents an output node. Note that the adjoint equations, i.e., $n \times (T-1)$ linear difference equations (13) with terminal data (14), must generally be solved backwards in time.

Substitution of (13) and (14) into (12) yields

$$\begin{aligned} \delta x_i(T) &= \textstyle\sum_j \sum_k \sum_{t=0}^{T-1} \delta w_{jk}(t)\phi_j(t+1) S_k(t) \\ &+ \textstyle\sum_j \sum_{t=1}^{T} \phi_j(t)\delta\xi_j(t). \end{aligned} \tag{15}$$

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

It is observed that the total variation of the internal state at time T, $\delta x_i(T)$, can be decomposed into two contributing terms. The first term in the right-hand side of (15) shows the total sum over time of variations in each of the connection weights $w_{jk}(t)$, and the second term shows the temporal sum of the effect of changes in each of the noise sequences $\xi_j(t)$, respectively. In the continuous limit, the temporal sums in (15) can be equivalently replaced by the corresponding integrals with respect to the (continuous) time t.

Hence, following the continuous-time arguments [10, 11], a discrete-time functional derivative can be formally defined based on (15):

$$\frac{\delta x_i(T)}{\delta \xi_j(t)} = \phi_j(t), \tag{16}$$

where $\frac{\delta x_i(T)}{\delta \xi_j(t)}$ denotes the discrete-time functional derivative sensitivity coefficient, which gives a relevant stochastic sensitivity measure for the present analysis. Equation (16) implies that the adjoint function $\phi_j(t)$ can be viewed as the sensitivity of the internal state at the i-th unit at time T, $x_i(T)$, with respect to a variation in the noise signal at the j-th unit at time t, $\xi_j(t)$.

The variational approach described above leads to the following state sensitivity lemma:

Lemma 1 (State Sensitivity): For the discrete-time neural network defined by (1)-(3), the state functional derivative sensitivity coefficient, $\frac{\delta x_i(T)}{\delta w_{jk}(t)}$, can be expressed as follows:

$$\frac{\delta x_i(T)}{\delta w_{jk}(t)} = \frac{\delta x_i(T)}{\delta \xi_j(t+1)} S_k(t), \tag{17}$$

for $t = 0, 1, 2, \ldots, T-1$.

Proof: From (15), the state functional derivative sensitivity coefficient (17) is derived as follows:

$$\frac{\delta x_i(T)}{\delta w_{jk}(t)} = \phi_j(t+1) S_k(t) = \frac{\delta x_i(T)}{\delta \xi_j(t+1)} S_k(t), \tag{18}$$

where the discrete-time functional derivative defined by (16) has been used. ■

Remark 1: It is important to observe that the adjoint function, $\phi_j(t+1)$, is not explicitly involved in (17). This contrasts with the deterministic theory based on the back-propagation-through-time (BTT) algorithm, where computation of the adjoint equation is needed [17]. Lemma 1, therefore, implies that the desired sensitivity coefficients may be obtained directly from the functional derivative sensitivities, $\frac{\delta x_i(T)}{\delta \xi_j(t+1)}$, without the need for actually computing the back-propagation equation. Similar results for continuous-time recurrent networks can be found in Koda [10, 11, 12].

4. Stochastic Learning of Discrete-Time Neural Networks

In this section, a stochastic learning algorithm in terms of the functional derivative formulation of the gradient descent method is proposed.

4.1. Main theorem

The algorithm uses the following identity (Novikov's Theorem):

$$E\Big[H(\xi)\xi_i\Big] = E\Big[\frac{\delta H(\xi)}{\delta \xi_i}\Big], \tag{19}$$

where $H(\xi)$ is an arbitrary functional of the Gaussian stochastic process $\xi(t)$, and $\frac{\delta H(\xi)}{\delta \xi_i}$ denotes the functional derivative [3].

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

Using in (7) Lemma 1 from Subsection 3.2 and the above Novikov's Theorem, the following performance sensitivity lemma is obtained:

Lemma 2 (Performance Sensitivity): For the discrete-time neural network defined by (1)-(3), the performance functional derivative sensitivity coefficient, $\frac{\delta L[x(T)]}{\delta w_{ij}(t)}$, can be expressed as follows:

$$\frac{\delta L[x(T)]}{\delta w_{ij}(t)} = L[x(T)]\xi_i(t+1)S_j(t), \tag{20}$$

which is valid in a statistical sense (i.e., as a mean or average).

Proof: Using (17) and (19) in (7) yields

$$\begin{aligned} \frac{\delta E[L[x(T)]]}{\delta w_{ij}(t)} &= E[\frac{\delta L[x(T)]}{\delta w_{ij}(t)}] = E[\sum_k \frac{\partial L[x(T)]}{\partial x_k(T)} \frac{\delta x_k(T)}{\delta w_{ij}(t)}] \\ &= E[\sum_k \frac{\partial L[x(T)]}{\partial x_k(T)} \frac{\delta x_k(T)}{\delta \xi_i(t+1)} S_j(t)] = E[\frac{\delta L[x(T)]}{\delta \xi_i(t+1)} S_j(t)] \\ &= E[\frac{\delta \{L[x(T)]S_j(t)\}}{\delta \xi_i(t+1)}] = E[L[x(T)]\xi_i(t+1)S_j(t)], \end{aligned} \tag{21}$$

where the following fact is used that $S_j(t)$ and $\xi_i(t+1)$ are independent stochastic variables (i.e., $\frac{\delta S_j(t)}{\delta \xi_i(t+1)} = 0$), because of the temporal causality of the network system (1). Further arrangement of the terms in (21) yields

$$E\left[\frac{\delta L[x(T)]}{\delta w_{ij}(t)} - L[x(T)]\xi_i(t+1)S_j(t)\right] = 0, \tag{22}$$

which leads to (20). ■

Using Lemma 2 in the functional derivative formulation of the gradient descent method described in Subsection 3.1, i.e., inserting (20) into (4), a stochastic implementation of the gradient descent method for a discrete-time neural network can be obtained.

Theorem: For the discrete-time stochastic neural network system defined by (1)-(3), the learning algorithm

$$w_{ij}^{\tau+1}(t) = w_{ij}^{\tau}(t) - \mu L[x(T)]\xi_i(t+1)S_j(t) \tag{23}$$

guarantees a monotonic decrease in the average value of the given performance functional $L[x(T)]$.

Proof: From the formal definition of discrete-time functional derivatives, the basic variational relationship is obtained as follows:

$$\delta L[x(T)] = \sum_i \sum_j \sum_{t=0}^{T-1} \frac{\delta L[x(T)]}{\delta w_{ij}(t)} \delta w_{ij}(t). \tag{24}$$

Using the learning algorithm (23) to update $w_{ij}^{\tau}(t)$ at time t, while the weights at other times are kept unchanged, the variations induced by the learning sequence are derived as follows:

$$\delta w_{ij}(t) = w_{ij}^{\tau+1}(t) - w_{ij}^{\tau}(t) = -\mu L^{\tau}[x(T)]\xi_i(t+1)S_j(t). \tag{25}$$

The corresponding variation of the performance functional is

$$\delta L[x(T)] = L^{\tau+1}[x(T)] - L^{\tau}[x(T)], \tag{26}$$

where $L^{\tau}[x(T)]$ denotes the value of $L[x(T)]$ computed by using $w_{ij}^{\tau}(t)$.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

Then, by taking the expectation of (26) and using (24), (25), and (20):

$$\begin{aligned} E[L^{\tau+1}[x(T)] - L^{\tau}[x(T)]] &= \sum_i \sum_j \sum_{t=0}^{T-1} E[\tfrac{\delta L[x(T)]}{\delta w_{ij}(t)}\{w_{ij}^{\tau+1}(t) - w_{ij}^{\tau}(t)\}] \\ &= \sum_i \sum_j \sum_{t=0}^{T-1} E[\tfrac{\delta L[x(T)]}{\delta w_{ij}(t)}] E[w_{ij}^{\tau+1}(t) - w_{ij}^{\tau}(t)] \\ &= -\mu \sum_i \sum_j \sum_{t=0}^{T-1} C_{ij}^{\tau}(T,t)^2 < 0, \end{aligned} \tag{27}$$

where $C_{ij}^{\tau}(T,t)$ is the correlation defined by

$$C_{ij}^{\tau}(T,t) = E[L^{\tau}[x(T)]\xi_i(t+1)S_j(t)]. \tag{28}$$

In establishing (27), it has been assumed the relations $\delta w_{ij}(s) = 0$ for $s \neq t$ and that the variations of the learning sequence, (25), and sensitivities, $\frac{\delta L[x(T)]}{\delta w_{ij}(t)}$, are statistically independent. Finally, from (27), the following relation is obtained:

$$E[L^{\tau+1}[x(T)]] < E[L^{\tau}[x(T)]], \tag{29}$$

which leads to the claim stated in the main theorem. ■

Remark 2: The assumption of unit variance in the Gaussian white noise sequence, (3), does not impose any limitations on the practical applications of the Theorem. In actual implementation of the algorithm, the variance of the noise does not need to be estimated, because it is never really used for gradient computation in weight space. The magnitude of the noise variance can be embedded into the learning rate μ (> 0) in (23), which could itself be a function of the learning time, i.e., $\mu = \mu(\tau)$. An appropriate cooling schedule that mimics simulated annealing may be considered for $\mu(\tau)$.

Remark 3: Equation (23) contains two time sequences, i.e., the system time t and the learning time τ. The network system (1) and the learning algorithm (23) may operate on different time scales. As a network evolves in system time t, (23) can be sequentially executed in chronological order without resort to back-propagation techniques. For a fixed system time, the update of connection weights according to (23) usually occurs at a slower rate with the learning time τ. It is important to note that the weight $w_{ij}^{\tau}(t)$ can take independent (constant) values over the system time ($t = 0, 1, 2, \ldots, T-1$).

Remark 4: Equation (23) does not involve differentiation of a signal function or summation over connection weights, while the BTT algorithm involves both (see (31)-(33) in the next section). It means that a non-differentiable function may be used for a signal function although its derivative appears only formally in the derivation of the learning algorithm, e.g., in (8)-(13). It also means that weight connections may be generated and deallocated dynamically at each node independent of the learning algorithm (23), while the BTT requires a static adjacency information of the network.

A central role in the stochastic sensitivity approach described above is played by ubiquitous noise at each individual unit. In general, such noise is small and the resulting variation of a performance functional may not be noticeably large. However, the Theorem indicates that the iterative modification of connection weights can be performed as a stochastic *reaction* of each connection weight $w_{ij}(t)$ to the ubiquitous noise $\xi_i(t+1)$ and the global performance functional $L[x(T)]$. Since the reaction (23) can occur locally without a synchronous transmission of information backward along network connections (e.g., back-propagation), and may be analogous to the subconscious (i.e., autonomous) activity of neurons, the learning algorithm (23) is called Subconscious Noise Reaction (SNR).

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

4.2. Framework of the algorithm

The framework of the learning algorithm based on the main theorem (23) is as follows:

1. Initialize the connection weights $w_{ij}(t)$ for $t = 0, 1, 2, \ldots, T-1$.
2. **For** learning time $\tau := 1, 2, \ldots, M$ **do**
3. **begin**
4. Clear the update increments $\delta w_{ij}(t)$ for $t = 0, 1, 2, \ldots, T-1$.
5. **For** running time $r := 1, 2, \ldots, R$ **do**
6. **begin**
7. Set an initial state vector $x(0) = x^0$.
8. Generate the Gaussian white noise vectors $\xi(t)$ for $t = 1, 2, \ldots, T$.
9. Calculate the state vectors $x(t)$ for $t = 1, 2, \ldots, T$.
10. Calculate the performance functional $L[x(T)]$.
11. Calculate new update increments based on (23) as
$\delta w_{ij}(t) := \delta w_{ij}(t) - \mu L[x(T)]\xi_i(t+1)S_j(t)$ for $t = 0, 1, 2, \ldots, T-1$.
12. **end;**
13. Update the connection weights as
$w_{ij}(t) := w_{ij}(t) + \delta w_{ij}(t)/R$ for $t = 0, 1, 2, \ldots, T-1$.
14. Adjust the connection weights based on the given value range $[w_{min}, w_{max}]$ as
$w_{ij}(t) := \max\{\min\{w_{ij}(t), w_{max}\}, w_{min}\}$ for $t = 0, 1, 2, \ldots, T-1$.
15. Stop if the given terminal condition is met.
16. **end;**

A set of training patterns $P = \{\{x^0, x^*\}, \ldots\}$ is usually given in advance, and the above algorithm is applied to obtain the optimal connection weights with which the network outputs desired output vectors x^* corresponding to given input vectors x^0.

The weight and state vectors may be implemented in a computer as floating point variables; for example, the connection weights $w_{ij}(t)$ may be implemented as an $n \times n \times T$ floating point array, where n is a number of nodes and T is a final time of the system. The initial value of each $w_{ij}(t)$ is arbitrary; Gaussian white noise[2] is used in the numerical study.

The procedure in learning time, from Steps 4 to 15, is called an *iteration*, which is further decomposed into four phases: initialize (Step 4), running (from Steps 5 to 12), update (Steps 13 and 14), and evaluation (Step 15). To accumulate update increments valid in a stochastic sense, R runs are performed in the running phase. Usually, a large number is chosen for R compared to the size of the training patterns, i.e., $|P|$. In Step 7, a pair of an input vector and a desired output vector, $\{x^0, x^*\} \in P$, is arbitrarily chosen. The performance functional $L[x(T)]$ should be defined to reflect the extent of errors in the actual output vector $x(T)$ compared to the desired output vector. The details of implementation of the SNR framework for three-layer feedforward networks are described in the next section.

5. Numerical Study

In this section, the convergence behavior of the proposed Subconscious Noise Reaction (SNR) algorithm is studied and compared with that of the standard back-propagation-through-time (BTT) algorithm [17]. A linearly separable pattern – logical OR – and two linearly non-separable patterns – logical XOR and symmetry detection – are used to evaluate performances of the SNR and BTT algorithms.

[2] Gaussian white noise with zero mean and unit variance is generated by $\cos(2\pi a)\sqrt{-2\log b}$, where a and b are real numbers uniformly distributed on $(0, 1)$-interval.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

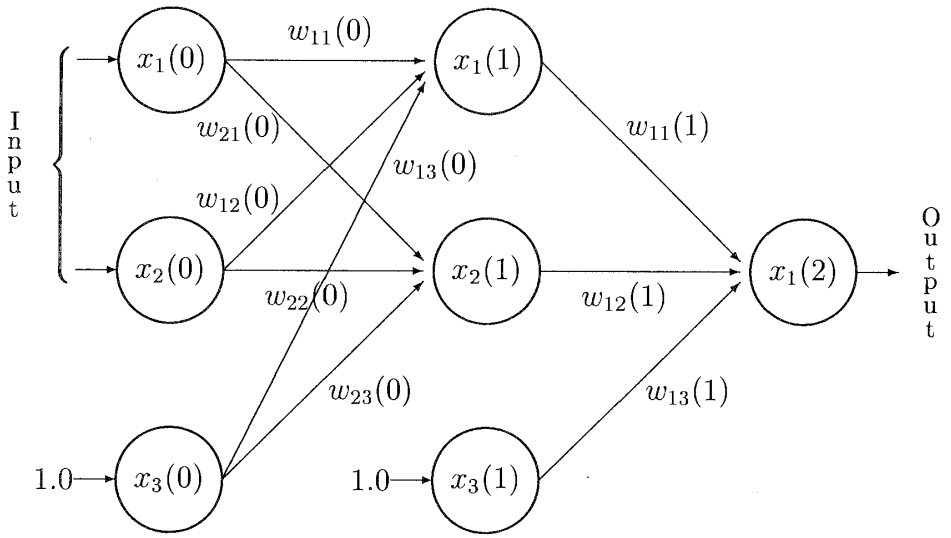

Figure 1: $N(3;3;1)$ layered network

5.1. Network models and learning algorithms

In order to adapt the SNR algorithm to general layered networks, $x_i(t)$ in (1) is interpreted as denoting the internal state at the i-th unit in the t-th network layer. This is the unfolded feedforward network model described in Section 2. Hence, hereafter, t represents an index for the t-th network layer; e.g., $t = T$ denotes the output layer. Similarly, $w_{ij}(t)$ is the weight of the connection between the j-th unit in the t-th layer and the i-th unit in the $(t+1)$-th layer.

Two types of three-layer-networks, $N(3;3;1)$ and $N(7;n;1)$, are used for the present numerical study. The $N(3;3;1)$ layered network (Figure 1) has two input units in the zeroth layer, two hidden units in the first layer, and one output unit in the second layer. In addition, auxiliary (threshold) units in the zeroth and first layers, which always hold the constant input (= 1.0), give threshold values to the units in the first and second layers, respectively. The $N(7;n;1)$ layered network (Figure 2) has six input units in the zeroth layer, $n-1$ hidden units in the first layer, one output unit in the second layer, and one threshold unit both in the zeroth and first layers. The value of n ranges from 2 to 20 in a *simulation*, where a simulation means an execution of the framework described in Subsection 4.2. For both networks, the internal states of units in the input layer are denoted by $x_i(0)$, those of units in the hidden layer by $x_i(1)$, and a single unit in the output (second) layer by $x_1(T)$ $(T = 2)$. Connection weights are denoted by $w_{ij}(t)$ $(t = 0, 1)$. In the following simulations, for the input and threshold units, the obvious relation $S_i(t) = x_i(t)$ is assumed. Similarly, for desired output vectors, $S^* = x^*$ is assumed. The sigmoid function, (2), is used for other units unless otherwise stated.

The performance functional $L[x(T)]$ is now specified to be a summation of the square of the difference (error) between the actual output $S_i(T)$ and the desired output S_i^*:

$$L[x(T)] = \frac{1}{2}\sum_i \{S_i(T) - S_i^*\}^2, \tag{30}$$

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

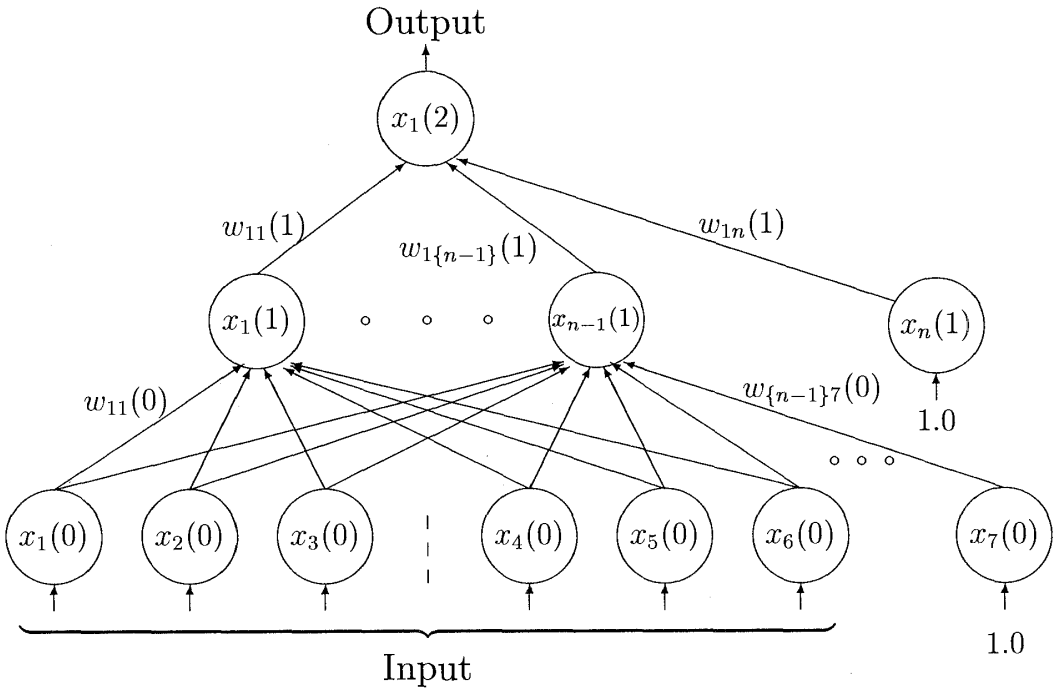

Figure 2: $N(7;n;1)$ layered network

where the summation is taken at all the corresponding output units. Note that, for $N(3;3;1)$ and $N(7;n;1)$, the network has only one output unit. Note also that, in the present numerical study, the gross performance index is defined by $J = \langle L[x(T)]\rangle$, where $\langle\cdot\rangle$ denotes the statistical average in a running phase.

Computation of the error derivatives associated with (30) in the unfolded network can be carried out by using the following BTT algorithm:

$$\begin{aligned}\sigma_j(t) &= S_j'(t)\sum_k w_{kj}^{\tau}(t)\sigma_k(t+1)\\ &= \{1+S_j(t)\}\{1-S_j(t)\}\sum_k w_{kj}^{\tau}(t)\sigma_k(t+1),\end{aligned} \tag{31}$$

with the terminal data

$$\begin{aligned}\sigma_i(T) &= \mu S_i'(T)\{S_i(T)-S_i^*\}\\ &= \mu\{1+S_i(T)\}\{1-S_i(T)\}\{S_i(T)-S_i^*\},\end{aligned} \tag{32}$$

and the weight modification

$$w_{ij}^{\tau+1}(t) = w_{ij}^{\tau}(t) - \sigma_i(t+1)S_j(t), \tag{33}$$

where μ is the learning rate ($\mu > 0$).

Several observations may be made here. In (32), when the actual output $S_i(T)$ is staying close to the desired output S_i^*, $\sigma_i(T)$ behaves essentially like a noise. As a result, (31) is regarded as back-propagating the noise to compute the error derivatives, i.e., $\sigma_i(t+1)$, used in the BTT algorithm (33). Note that the back-propagation equation (31) is essentially equivalent to the adjoint equation (13), though the terminal conditions, (32) and (14), respectively, are different. Further, the BTT algorithm (33) corresponds to the SNR algorithm (23), where the stochastic correlation $\mu L[x(T)]\xi_i(t+1)$ is used instead of $\sigma_i(t+1)$

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

or error sensitivity. Thus, in contrast to the BTT algorithm, the SNR algorithm does not need to solve the back-propagation equations (31) and (32).

For $N(n;n;\ldots;n)$ layered networks, the time complexity (computational cost) of updating a connection weight by means of the BTT algorithm (31)-(33) is $O(n)$, because of the summation involved in (31). In consequence, the time complexity of updating all the connection weights in a layer by means of the BTT algorithm is $O(n^2)$. On the other hand, the complexity of updating a connection weight by means of the SNR algorithm is $O(1)$; i.e., the total time complexity is $O(n)$, which is much smaller than that of the BTT algorithm.

5.2. Simulation results

Both the SNR and BTT algorithms were used to perform numerical simulations for the $N(3;3;1)$ and $N(7;n;1)$ layered networks, and their convergence behaviors were compared. The framework described in Subsection 4.2 was used for both algorithms. For BTT simulations, Step 11 of the framework was modified to calculate the update increments based on (33) as $\delta w_{ij}(t) := \delta w_{ij}(t) - \sigma_i(t+1)S_j(t)$ for $t = 0,1,2,\ldots,T-1$, which means the back-propagation of error derivatives should be performed in this step.

For training patterns (i.e., sets of input and desired output vectors), logical OR and XOR patterns were used for $N(3;3;1)$, and a mirror symmetry (SYM) pattern was used for $N(7;n;1)$. In the OR pattern, only two hidden units (including a threshold unit) are required, while the XOR and SYM patterns require at least three hidden units for neural learning. Note that the SYM pattern is used to detect mirror symmetry in an input vector of six bits; when an input vector is symmetric – $(0,0,0,0,0,0)$, $(1,0,0,0,0,1)$, $(0,1,0,0,1,0)$, $(1,1,0,0,1,1)$, $(0,0,1,1,0,0)$, $(1,0,1,1,0,1)$, $(0,1,1,1,1,0)$, $(1,1,1,1,1,1)$ – the output is 1; otherwise, the output is 0. This symmetry detection is rather a difficult problem since all the neighbors of symmetric vectors within Hamming distance 1 are asymmetric.

In Step 7 of the framework, an input vector was prepared as follows: a desired output value S_1^* was selected randomly from 1 and 0, and an input vector was then selected randomly from vectors corresponding to the selected output. Each component in an input vector took a value of either $+1$ or -1. Once an input vector (x_1^0, x_2^0) or $(x_1^0,\ldots,x_6^0)$ had been set, the network states and the following decision function was calculated:

$$D[S(T)] = \begin{cases} 1 & \text{when } S(T) \text{ is correct,} \\ 0 & \text{when } S(T) \text{ is incorrect,} \end{cases} \tag{34}$$

where $S(T)$ is said to be *correct* when

$$\begin{aligned} +1 \geq S_i(T) \geq 0 \quad &\text{iff} \quad +1 \geq S_i^* \geq 0, \\ -1 \leq S_i(T) < 0 \quad &\text{iff} \quad -1 \leq S_i^* < 0. \end{aligned}$$

Simulations started with each connection weight initialized to a Gaussian white noise (with zero mean and unit variance), and terminated when the following conditions were both met:

$$\begin{aligned} J = \langle L[x(T)] \rangle \quad &\leq 0.01; \quad \text{and,} \\ 1 - \langle D[S(T)] \rangle \quad &\leq 0.01, \end{aligned}$$

where $\langle\cdot\rangle$ denotes the average value in a running phase. The simulation was aborted if the number of iterations exceeded M, i.e., if the simulation did not converge. For the OR and XOR patterns, $M = 1000$, and for the SYM pattern, $M = 5000$. Throughout this numerical study, Gaussian white noise was injected into the hidden and output units, except in the second case of BTT simulation for the OR pattern (Figure 5), where the network was completely noise-free.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

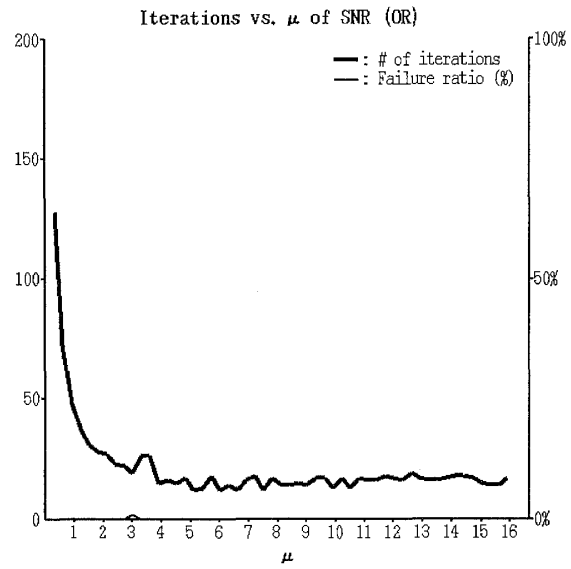

Figure 3: Average convergence time and failure ratio of the SNR algorithm for an OR pattern, plotted against μ

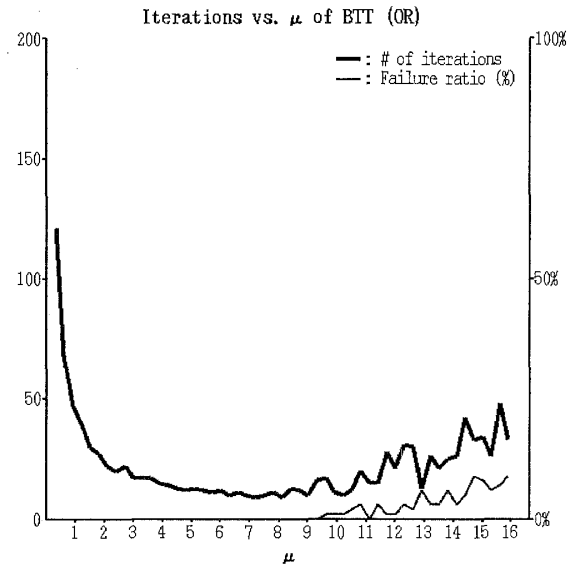

Figure 4: Average convergence time and failure ratio of the BTT algorithm for an OR pattern, plotted against μ

The running phase consists of $R = 100$ runs to compute the network states and $D[S(T)]$, and to accumulate the update increments, i.e., the second terms in the right-hand sides of (23) and (33). After the running phase, each connection weight is modified in the update phase. A value range $[-10, +10]$ was specified for each of the connection weights. In the evaluation phase, the conditions for termination are checked, and if the simulation converges, the convergence time is recorded.

For simulations of the OR and XOR patterns, the learning rate μ varies over a range of values from 0.1 to 16.0, and 100 simulations were conducted for each value of μ. For simulations of the SYM pattern, μ is always set to 5.0; on the other hand, the number of hidden units n (including a threshold unit) varies over a range of values from 2 to 20, and 100 simulations were conducted for each value of n. The average value of the convergence time and the number of failed runs among the 100 simulations are defined as the *average convergence time*, and as the *failure ratio*, respectively.

Comparison of the SNR and BTT algorithms for the OR pattern (Figures 3 and 4) shows that simulations always converge for both algorithms when μ is in a certain range, and that SNR has basically the same performance characteristics as BTT. The average convergence times of both algorithms are almost the same for small μ.

When BTT was used in a noise-free environment, on the other hand, where no noise was injected into the hidden and output units, BTT's performance was very poor. Figure 5 shows the result of using the BTT algorithm for the OR pattern without noise injection. Comparison of Figures 4 and 5 reveals that, to obtain better performance, the BTT algorithm should be used in a stochastic environment just as the SNR algorithm. This is related to the recent findings of the enhanced performance of neural networks with additive noise (e.g., [16]), and constitutes one of motivations for investigating the present stochastic learning algorithm.

Figure 6 shows the result of using the SNR algorithm for the OR pattern while using a

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

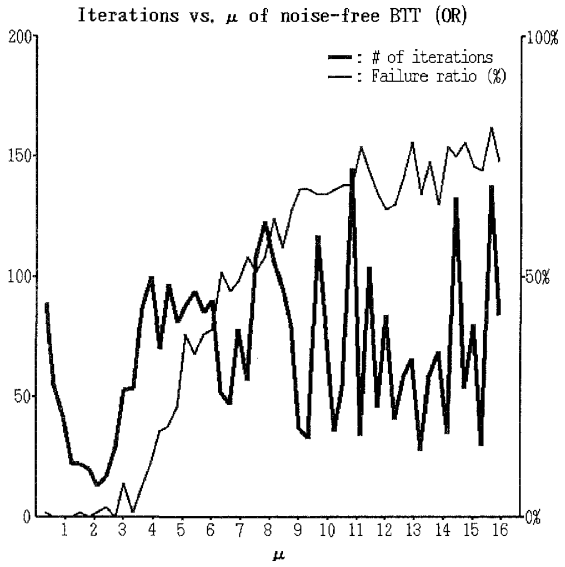

Figure 5: Average convergence time and failure ratio of the BTT algorithm for an OR pattern, plotted against μ. No noise was injected.

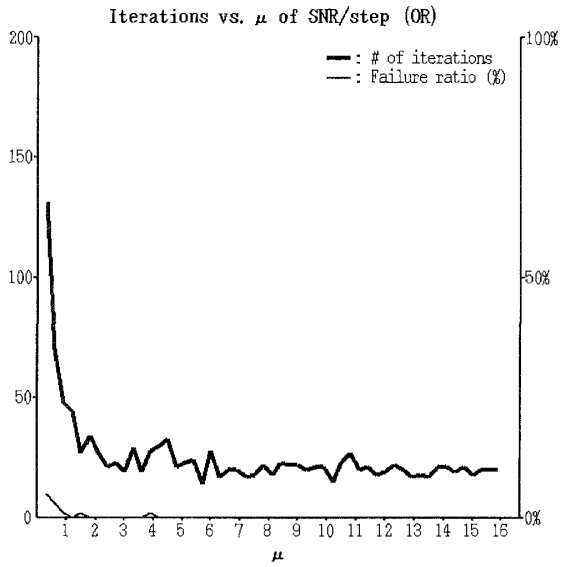

Figure 6: Average convergence time and failure ratio of the SNR algorithm with a step function for an OR pattern, plotted against μ

step function

$$S_i(t) = \mathrm{step}[x_i(t)] = \begin{cases} +1 & \text{when } x_i(t) \geq 0, \\ -1 & \text{when } x_i(t) < 0, \end{cases} \tag{35}$$

for normal units at $t = 1$ and $t = T$ ($T = 2$). Note that the obvious relation $S_i(t) = x_i(t)$ was assumed for units at $t = 0$ and for threshold units. Comparison of Figures 3 and 6 reveals that the SNR can perform neural learning while using a step function as well as with a sigmoid function. Note that the BTT algorithm does not work with a step function because (32) and (33) explicitly differentiate the signal function.

Simulations were conducted in a similar manner for the XOR pattern (Figures 7 and 8). In contrast to the OR pattern, the failure ratios of both the SNR and BTT algorithms depend strongly on μ, and no single value of μ guarantees convergence of the BTT algorithm. Which means determination of a proper learning rate is crucial and tedious for both algorithms. Note that the failure ratio of SNR becomes higher than that in Figure 7 when the value range is not set, and that determination of a proper value range is important for SNR.

The result of the SNR algorithm while using the step function (Figure 9) reveals that SNR can train the neural network for the XOR pattern as well as with a sigmoid function (Figure 7). Note that both the number of iterations and the failure ratio can be decreased by increasing the number of hidden units or by adjusting the noise variance, and that the experimental setup used here may not be optimal for both algorithms.

Simulation results for the SYM pattern (Figures 10 and 11) show that the sensitivities of the SNR and BTT algorithms to the complexity of the network architecture (i.e., the number of hidden units n) are basically the same. No single value of μ guarantees convergence of the BTT algorithm as well as for the XOR pattern. Comparison of the results of the SNR algorithm with the sigmoid and step functions (Figures 10 and 12) reveals that the choice of these signal functions does not greatly affect the convergence behavior and the architecture sensitivity of SNR.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

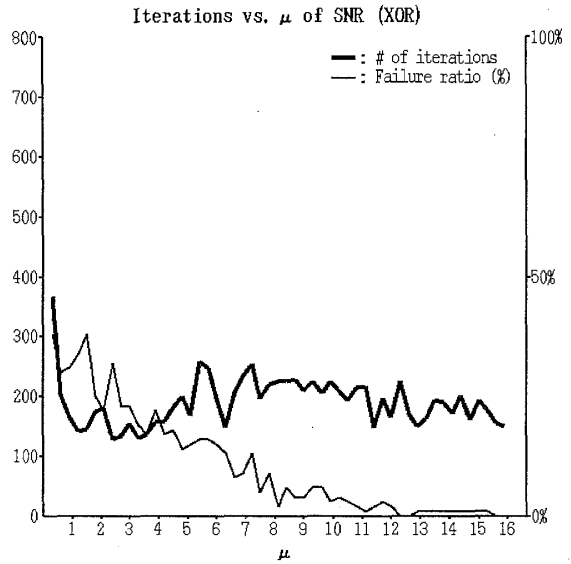

Figure 7: Average convergence time and failure ratio of the SNR algorithm for an XOR pattern, plotted against μ

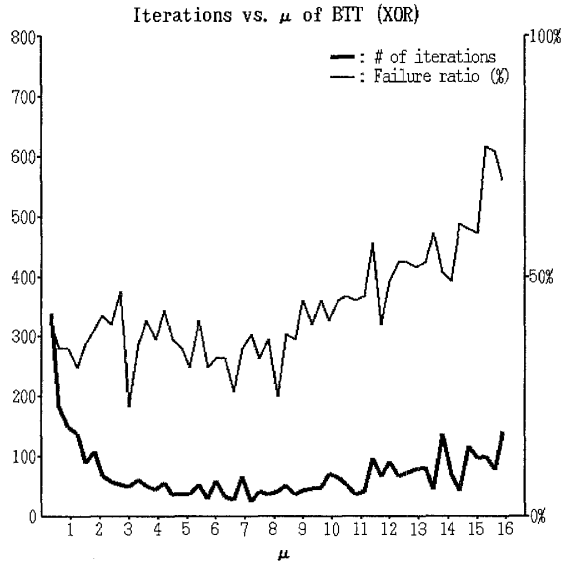

Figure 8: Average convergence time and failure ratio of the BTT algorithm for an XOR pattern, plotted against μ

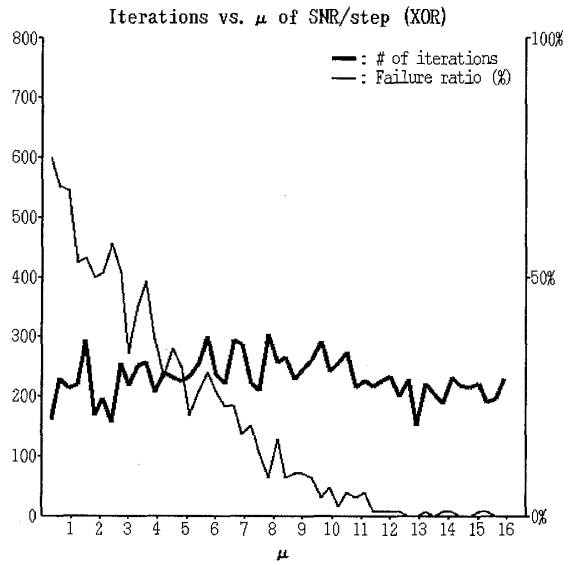

Figure 9: Average convergence time and failure ratio of the SNR algorithm with a step function for an XOR pattern, plotted against μ

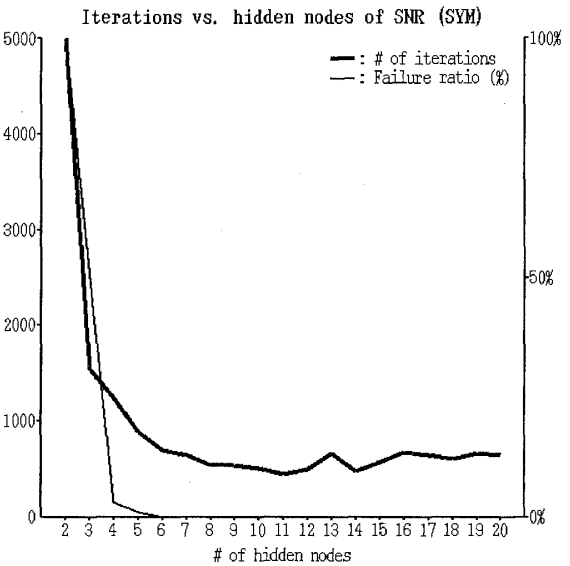

Figure 10: Average convergence time and failure ratio of the SNR algorithm for a mirror symmetry pattern, plotted against the number of hidden units

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

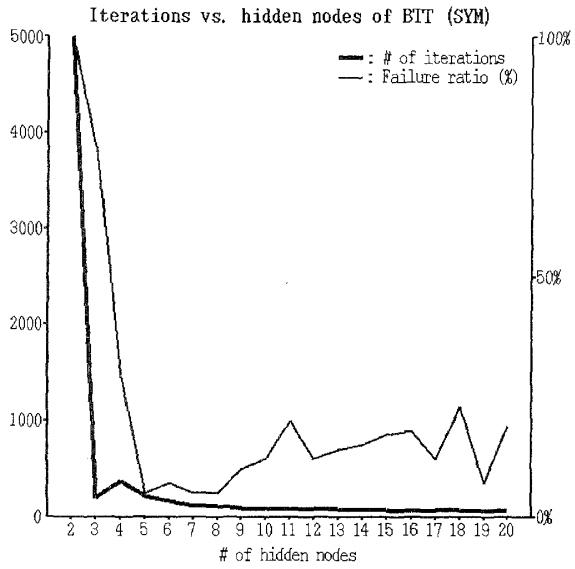

Figure 11: Average convergence time and failure ratio of the BTT algorithm for a mirror symmetry pattern, plotted against the number of hidden units

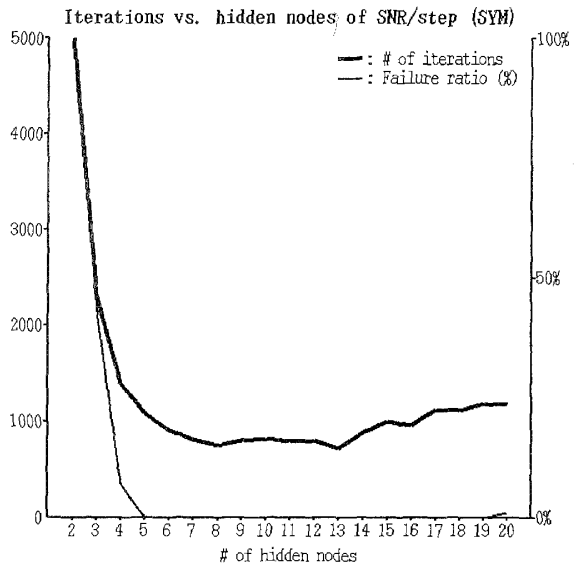

Figure 12: Average convergence time and failure ratio of the SNR algorithm with a step function for a mirror symmetry pattern, plotted against the number of hidden units

These results indicate that the SNR algorithm can be used to train the neural network in a stochastic environment, and that the SNR and BTT algorithms have basically the same capabilities. For all the experiments in this section, the failure ratio of SNR is generally smaller than that of BTT while the average number of iterations required by SNR is larger than that by BTT. As stated earlier, the time complexity of the SNR algorithm for updating a connection weight is smaller than that of the BTT algorithm; however, this advantage may be offset by an increase in the number of iterations needed for convergence. Consequently, the overall performance of the SNR and BTT algorithms may be comparable when suitable setups for the network architecture or the noise variances are realized for both algorithms.

6. Conclusions

A new stochastic learning algorithm, referred to as Subconscious Noise Reaction (SNR), was developed for a class of discrete-time recurrent neural networks with time-dependent connection weights and additive Gaussian white noise. The algorithm iteratively modifies connection weights as a stochastic reaction to the ubiquitous noise and to the global network objective. The present SNR algorithm has an advantage over standard back-propagation techniques in that it alleviates the need to solve lengthy back-propagation equations by providing a comprehensive theoretical framework derived from stochastic sensitivity analysis using the variational approach. Moreover, SNR's computer implementations are much simpler than standard back-propagation techniques; a simple, but non-differentiable step function can be used for the signal function, and SNR's complexity of updating all the connection weights in a layer is much smaller than that required by BTT. For $N(n; n; \ldots; n)$ layered networks, SNR only requires $O(n)$ updates while BTT requires $O(n^2)$. Further, SNR does not use the adjacency information of the network connections in updating their weights. Because of these properties, SNR may be suited to hardware (chip) implementations.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

Acknowledgment

The authors would like to thank anonymous referees for their valuable comments which helped to improve the paper. The work of the first author was supported in part by a Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports, and Culture of Japan.

References

[1] M. A. Cohen and S. Grossberg: Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-13** (1983) 815-826.

[2] F. Crick: The recent excitement about neural networks. *Nature*, **337** (1989) 129-132.

[3] D. K. Dacol and H. Rabitz: Sensitivity analysis of stochastic kinetic models. *Journal of Mathematical Physics*, **25** (1984) 2716-2727.

[4] S. Grossberg: Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, **1** (1988) 17-61.

[5] D. O. Hebb: *The Organization of Behavior* (John Wiley, New York, 1949).

[6] J. Hertz, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation* (Pegasus Books, 1991)

[7] D. H. Hubel and T. N. Wiesel: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, **160** (1962) 106-154.

[8] S. Kirkpatrik, C. D. Gelatt, Jr., and M. P. Vecchi: Optimization by simulated annealing. *Science*, **220** (1983) 671-680.

[9] M. Koda: Sensitivity analysis of stochastic dynamical systems. *International Journal of Systems Science*, **23** (1992) 2187-2195.

[10] M. Koda: Stochastic sensitivity analysis method for neural network learning. *International Journal of Systems Science*, **26** (1995) 703-711.

[11] M. Koda: Neural network learning based on stochastic sensitivity analysis, *IEEE Transactions on Systems, Man, and Cybernetics Part B*, **27** (1997) 132-135.

[12] M. Koda: Stochastic sensitivity analysis and Langevin simulation for neural network learning. *Journal of Reliability Engineering and System Safety*, **57** (1997) 71-78.

[13] M. Koda and J. H. Seinfeld: Sensitivity analysis of distributed parameter systems. *IEEE Transactions on Automatic Control*, **AC-27** (1982) 951-955.

[14] B. Kosko: Structural stability of unsupervised learning in feedback neural networks. *IEEE Transactions on Automatic Control*, **AC-36** (1991) 785-792.

[15] K. Matsuoka: Learning and evolution of neural nets (in Japanese). *Journal of SICE*, **32** (1993) 843-847.

[16] A. F. Murray and P. J. Edwards: Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks*, **NN-5** (1994) 792-802.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams: Learning representation by back-propagation errors. *Nature*, **232** (1986) 533-536.

Hiroyuki Okano
IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato
Kanagawa-ken 242-8502, JAPAN
E-mail: okanoh@jp.ibm.com

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.