Learning time constant of continuous-time neurons with gradient descent

Toshitaka Matsuki and Katsunari Shibata

Oita University, 700 Dannoharu, Oita, Japan
{matsuki,shibata}@oita-u.ac.jp

Abstract. In this paper, we propose a learning method to update the time constant in each continuous-time neuron with gradient descent to generate desired output patterns. Selecting appropriate time constant for each neuron in a continuous-time recurrent neural network is difficult. Hence, the development of adaptive method of the time constant is desired. However, direct update of time constants with gradient descent is significantly unstable. Therefore, to avoid the instability, we propose a learning method applying gradient descent to the logarithm of the time constant. We carried out an oscillator reproducing task in which a learning network is trained to generate the same oscillatory outputs from the teacher network. The training result shows that our proposed method can successfully update the time constants and suggests that leaning of time constants expands the freedom in learning and improve the learning performance.

Keywords: Neural Network, Continuous Time Neuron, Time Constant, Continuous Time Recurrent Neural Network

1 Introduction

The remarkable performance of the deep learning (DL) has attracted much attention in recent years[1]. In the DL, an entire computational function from input to output is acquired in a large-scale neural network (NN) through modification of connection weights, and successfully trained NN can be more flexible and powerful than carefully designed systems. The world we live has not only the extent of space but also the flow of time, therefore, in the future development of the NN for processing sensor input sequences, memorizing or recalling information, generating consistent motor outputs, making decision or thinking, temporal processing will be considerably significant. A recurrent neural network (RNN), in which neurons are recurrently connected and information is maintained over time, is used to learn time series processing. For generating continuous output pattern or producing complex dynamics in the RNN, a continuous-time recurrent neural network (CTRNN) which consists of continuous-time neurons is used [2]. The internal state of the continuous-time neuron is modeled by the linear first-order differential equation and its time scale is determined by its time constant. With a small time constant, the internal state of a neuron strongly decays and

takes the current inputs. Conversely, with a large time constant, internal state of a neuron changes sluggishly holding its previous internal state and taking little current inputs.

A practical task requires an artificial agent to think and/or act properly in various time scales. For example, we suppose that an artificial agent pours juice in a bottle into a glass. In a short time scale, the agent has to change its motor commands finely and quickly depending on the pouring situation, while in a long time scale, it has to transfer its mode among multiple states such as taking the bottle, uncapping, pouring the juice into a glass and handing someone the glass of juice.

Achieving such multiple time scale behavior is difficult for RNN consisting of static neurons because its time scale is equivalent to the step size and so all the neurons have the same time scale. To introduce various time scales into RNN, the continuous-time neurons having different time constant is essential. Tani et al. showed that functional hierarchy can emerge in the CTRNN which has multiple time scales with different and fixed time constants [3][4]. Quite some time ago, M.C.Mozer showed the RNN which has multi-scale temporal structure can effectively learn structures in temporally expanded sequences, and referred to potential and difficulty of direct learning of time constant [5]. We expect that the modification of time constant can assist a network to learn various time scale tasks and the functional hierarchy according to the time scale to emerge among the neurons, and also expansion of the degree of freedom into the time axis increases the leaning performance significantly.

Thus, the development of adaptive method of the time constant is desired. However, direct update of time constants with gradient descent is unstable, because when the time constant is small, the output of the neuron is significantly sensitive, therefore the gradient descent makes the update of the time constant large, and vice versa. Hence the time constant often becomes negative.

In this paper, to avoid the difficulty, we propose a learning method applying gradient descent to the logarithm of the time constant and demonstrate the network can successfully modify the time constant of its neurons to generate desired oscillatory patterns with this method.

2 Method

2.1 Network

In a CTRNN, internal state of a neuron is updated continually by the following differential equation

$$\tau_j \dot{u}_{j,t} = -u_{j,t} + \sum_{i=1}^{N} w_{j,i} x_{i,t} \tag{1}$$

where τ_j, $u_{j,t}$, $x_{j,t}$ are the time constant, the internal state and the output of the j-th neuron at time t respectively, $w_{j,i}$ is the connection weight from the i-th neuron to the j-th neuron. In the computer simulation in this research, we

compute the network behavior with finite difference approximation, and actual update is computed according to

$$u_{j,t} = (1 - \frac{\Delta t}{\tau_j})u_{j,t-\Delta t} + \frac{\Delta t}{\tau_j}\sum_{i=1}^{N} w_{j,i}x_{i,t-\Delta t} \tag{2}$$

where $\Delta t = 0.01$ is the simulation time step. In Eq. (2), the internal state of each neuron is determined not only by the current inputs but also by the decayed internal state of itself. The output of the neuron is calculated according to tangent hyperbolic function as

$$x_{j,t} = \tanh(u_{j,t}) = \frac{e^{u_{j,t}} - e^{-u_{j,t}}}{e^{u_{j,t}} + e^{-u_{j,t}}}. \tag{3}$$

The time constant τ_j decides the time scale of the network dynamics. When the τ_j of a neuron is large, its internal state changes slowly because $\frac{\Delta t}{\tau_j}$ is small and the previous state of the neuron strongly affect the current internal state. Whereas, when the τ_j of a neuron is small, its internal state changes quickly because $\frac{\Delta t}{\tau_j}$ is large and the current inputs strongly affect the current internal state. In this research, we propose the training method for the time constant value of each neuron in CTRNN.

2.2 Training

We use gradient descent to modify not only the connection weights w but also the time constants τ of the network neurons. Here, we assume an easy case that every neuron is given its training signal directly. We define the error function at time t for the neurons as

$$E_t = \sum_{j=1}^{N} \frac{1}{2}(d_{j,t} - x_{j,t})^2 \tag{4}$$

where $d_{j,t}$ and $x_{j,t}$ are the desired output and actual output of j-th neuron at time t respectively.

Then, we can adjust the connection weights using the gradient descent by

$$\Delta w_{ji} = -\eta_w \frac{\partial E_t}{\partial w_{ji}} = -\eta_w \frac{\partial E_t}{\partial x_{j,t}} \frac{dx_{j,t}}{du_{j,t}} \frac{\partial u_{j,t}}{\partial w_{ji}} \tag{5}$$

where η_w is a small positive constant called learning rate determining the step size in the gradient descent search. This equation is defined with negative sign because the error should be decreased. The Eq. (5) can be expanded as

$$\Delta w_{ji} = \eta_w (d_{j,t} - x_{j,t})(1 - x_{j,t}^2)\frac{\Delta t}{\tau_j} x_{i,t-\Delta t} \tag{6}$$

Now, we consider that the time constant of a neuron is trained with the same way as the following equation.

$$\Delta\tau_j = -\eta_T \frac{\partial E_t}{\partial \tau_j} = -\eta_T \frac{\partial E_t}{\partial x_{j,t}} \frac{dx_{j,t}}{du_{j,t}} \frac{\partial u_{j,t}}{\partial \tau_j}. \tag{7}$$

However, actually, the time constant cannot be trained stably with this equation. We explain the reason why the training of time constant is unstable. The change of internal state u_j at time t is

$$\Delta u_{j,t} = u_{j,t} - u_{j,t-\Delta t} = \frac{\Delta t}{\tau_j}\big(-u_{j,t-\Delta t} + \sum_{i=1}^{N} w_{j,i} x_{i,t-\Delta t}\big). \tag{8}$$

This equation indicates that for small τ_j, u_j is sensitive to small variations of τ_j while insensitive for large τ_j because τ_j is in the denominator. Now, $\frac{\partial u_{j,t}}{\partial \tau_j}$ is calculated as

$$\begin{aligned}
\frac{\partial u_{j,t}}{\partial \tau_j} &= \frac{\partial}{\partial \tau_j}\left\{(1 - \frac{\Delta t}{\tau_j})u_{j,t-\Delta t} + \frac{\Delta t}{\tau_j}\sum_{i=1}^{N} w_{j,i} x_{i,t-\Delta t}\right\} \\
&= \frac{\Delta t}{\tau_j^2} u_{j,t-\Delta t} + (1 - \frac{\Delta t}{\tau_j})\frac{\partial u_{j,t-\Delta t}}{\partial \tau_j} \\
&\quad -\frac{\Delta t}{\tau_j^2}\sum_{i=1}^{N} w_{j,i} x_{i,t-\Delta t} + \frac{\Delta t}{\tau_j}\sum_{i=1}^{N} w_{j,i}\frac{\partial x_{i,t-\Delta t}}{\partial \tau_j}.
\end{aligned}$$

Ignoring the fourth term, which indicates the indirect influence of τ_j through other neurons,

$$\frac{\partial u_{j,t}}{\partial \tau_j} = -\frac{\Delta t}{\tau_j^2}(-u_{j,t-\Delta t} + \sum_{i=1}^{N} w_{j,i} x_{i,t-\Delta t}) + (1 - \frac{\Delta t}{\tau_j})\frac{\partial u_{j,t-\Delta t}}{\partial \tau_j} \tag{9}$$

$$= -\frac{1}{\tau_j}(u_{j,t} - u_{j,t-\Delta t}) + (1 - \frac{\Delta t}{\tau_j})\frac{\partial u_{j,t-\Delta t}}{\partial \tau_j}. \tag{10}$$

This equation can be calculated through recursive computation by replacing $\frac{\partial u_{j,t}}{\partial \tau_j}$ with $a_{j,t}$ as

$$a_{j,t} = -\frac{1}{\tau_j}(u_{j,t} - u_{j,t-\Delta t}) + (1 - \frac{\Delta t}{\tau_j})a_{j,t-\Delta t}. \tag{11}$$

Eq. (7) and (9) let us find that $\Delta\tau_j$ is the value of the order of τ_j^{-2}. Therefore, when τ_j is very small, $\Delta\tau_j$ diverges and τ_j often becomes negative. While once τ_j become very large, $\Delta\tau_j$ is too small for τ_j to be back to a small value.

Hence, we need a technique to regulate this sensitivity depending on τ_j and keep it positive. To meet these requirements, we introduce a logarithm in the time constant

$$T_j = \log_e \tau_j.$$

By updating τ_j through T_j, τ_j can be prevented from being negative or stuck a large value. For small τ_j, $\frac{\partial u_j}{\partial T_j}$ and $\Delta\tau_j$ become smaller, and for large τ_j, $\frac{\partial u_j}{\partial T_j}$ and $\Delta\tau_j$ become larger as

$$\frac{\partial u_j}{\partial T_j} = \tau_j \frac{\partial u_j}{\partial \tau_j}$$
$$\Delta\tau_j = \tau_j \Delta T_j$$

Fig.1 shows the mapping from T_j to τ_j. As shown in this figure, when T_j is a large negative value, $\Delta\tau_j$ is small and τ_j never becomes negative but converges to zero, conversely, when T_j is a large positive value, $\Delta\tau_j$ is large.

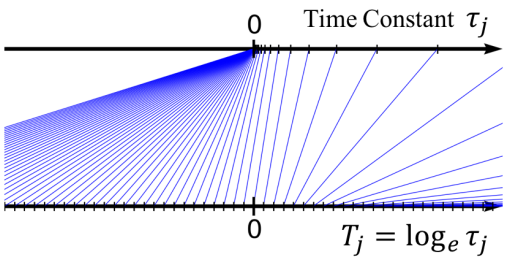


Fig. 1. The mapping from T_j to τ_j.

Based on the gradient descent, T_j is updated by

$$\Delta T_j = -\eta_T \frac{\partial E_t}{\partial T_j} = -\eta_T \frac{\partial E_t}{\partial x_{j,t}} \frac{dx_{j,t}}{du_{j,t}} \frac{\partial u_{j,t}}{\partial \tau_j} \frac{d\tau_j}{dT_j}. \tag{12}$$

For training τ_j, we use the relation between dT_j and $d\tau_j$ as

$$d\tau_j = \tau_j dT_j \tag{13}$$

and we obtain updating equation as

$$\Delta\tau_j = -\eta_T \tau_j^2 \frac{\partial E_t}{\partial x_{j,t}} \frac{dx_{j,t}}{du_{j,t}} \frac{\partial u_{j,t}}{\partial \tau_j}. \tag{14}$$

Now, the equation (14) can be expanded as

$$\Delta\tau_j = -\eta_\tau \tau_j^2 (d_{j,t} - x_{j,t})(1 - x_{j,t}^2) \frac{\partial u_{j,t}}{\partial \tau_j}. \tag{15}$$

The τ_j^2 in equation (15) cancels the effect of $\frac{1}{\tau_j^2}$ in equation (9), and consequently, the exploding or vanishing of $\Delta\tau_j$ can be avoided.

In addition, there is one more problem for learning pattern generation using recurrent connections of an RNN. If the output includes a lot of errors, the

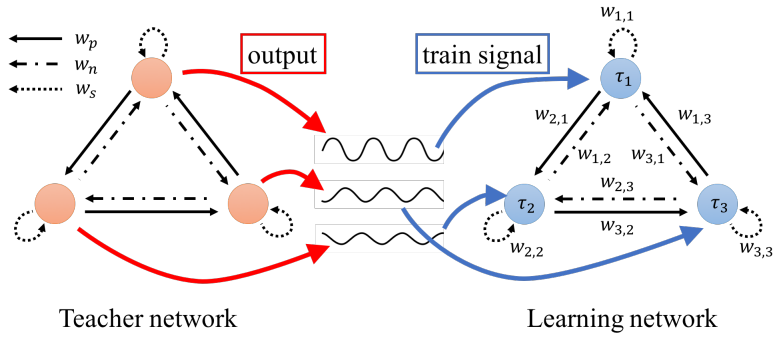


Fig. 2. The network structure.

neurons learn based on the erroneous outputs as feedback inputs for the next time step, and leaning does not progress appropriately. Same problem in learning dynamic pattern generation is treated in some literatures about reservoir computing. Jeager et al. employed teacher forcing that use teacher signal as output feedback during training [6]. Sussillo and L.F.Abbott proposed FORCE leaning that modifies weights so that error in the network output fed back to reservoir is kept small every time step during training [7]. These works avoid the problem by keeping the feedback error zero or small. In this work, to keep the feedback error small during learning, we modify the internal state of the neurons according to the teacher signal at every time step during training with

$$\Delta u_{j,t} = \eta_u (d_{j,t} - x_{j,t}) \frac{du_{j,t}}{dx_{j,t}} \tag{16}$$

where η_u is modification rate, which is a small positive constant that adjusts the modification.

3 Simulation

3.1 Task

To test the proposed method, we applied it to an oscillator learning task as shown in Fig. 2. We employed two CTRNNs: one of them is a "teacher network" and the other is a "learning network", and each of them has $N = 3$ neurons. The teacher network generates oscillatory outputs and the learning network learned using them as training signals. We observed how time constants or connection weights of the learning network were changing during training and whether the learning network can successfully learn. To make the teacher network generate self-sustained oscillatory activities as training signal, we set specific values on its connection weights. There are three kinds of connection weights, we set the weights in anti-clockwise pathways to w_p, in clockwise pathways to w_n and self-feedback pathways to w_s. The initial connection weight w_p, w_s is set a positive

value and w_n is set a negative value. Moreover, the initial internal state of one neuron was set to 1, and others were set to 0. The excitation of a neuron shifts anticlockwise and self-sustained oscillations appear in the teacher network. The learning network modifies its time constant and/or connection weights to approximate the outputs to the training signals.

We examined in three task settings. First, only the time constants of learning network was set to different values from teacher network and trained to generate desired outputs (Task 1). Second, the connection weights and time constants of learning network were set to different values from teacher network and only connection weights were trained (Task 2). Third, the initial parameters of the learning network were same as Task 2 and all the parameters were trained (Task 3). The setting of the tasks are summarized in Table 1. The learning rates and modification rate were set as shown in Table 2 and the initial parameters of the two networks were set as shown in Table 3. For the purpose of preventing time constants from becoming gratuitously large value or less than simulation time step, the range of time constants is limited from 0.01 to 100.

Table 1. The outline of the simulation tasks

	time constant		connection weight	
	initial value	training	initial value	training
Task 1	different	trained	same as teacher network	fixed
Task 2	different	fixed	different	trained
Task 3	different	trained	different	trained

Table 2. The learning rates and modification rate

η_w	η_T	η_u
0.1	0.01	0.01

Table 3. Initial parameters of the networks

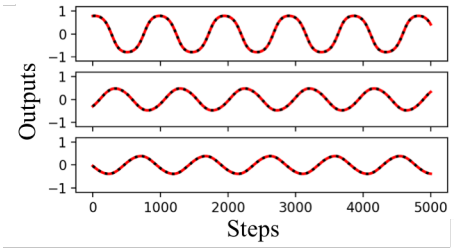
	τ_1	τ_2	τ_3
target network	2.0	5.0	8.0
learning network	0.1	0.1	0.1

		task1	task2	task3
target network	w_s		2.0	
	w_p		3.0	
	w_n		-4.0	
learning network	w_s	2.0	0.0	0.0
	w_p	3.0	0.0	0.0
	w_n	-4.0	0.0	0.0

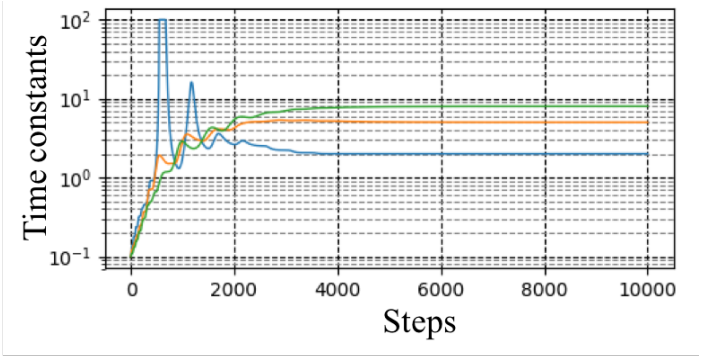
3.2 Results

The learning network was trained with the training signals from the teacher network in training phase and tested without updating its connection weights or time constants in test phase. The number of learning steps is 10000 for Task 1 and 50000 for Task 2, 3. The number of test steps is 5000 for each task. Fig. 3, 4 and 5 show the training results.

Task 1: As shown in Fig. 3, each time constant in the learning network converged to the corresponding value of the teacher network, and the learning network successfully learned to generate the same oscillatory activities as the training signals. This result indicates the time constant of each continuous-time neuron can be trained perfectly on the basis of training signal to generate desired output pattern when the connection weights in each network are the same values.



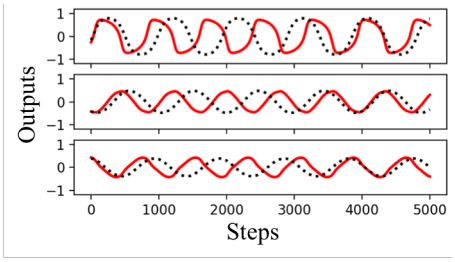
(a) Outputs (test phase)



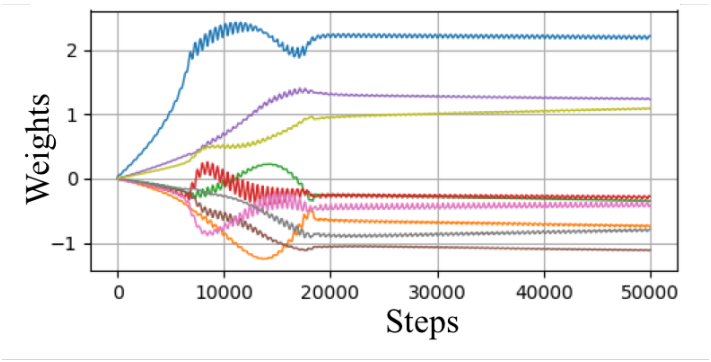
(b) Time constants (training phase)

Fig. 3. The results in Task 1. Figure (a) shows the outputs of the learning network (red line) and the desired output generated by teacher network (dotted line) during test phase. Figure (b) shows the time constants of the learning network during training.

Task 2: As shown in Fig. 4, the learning network failed to learn. Although the learning network modified its connection weights to generate desired pattern with the difference of time constants from the teacher network, the network did not succeed to generate desired output.



(a) Outputs (test phase)

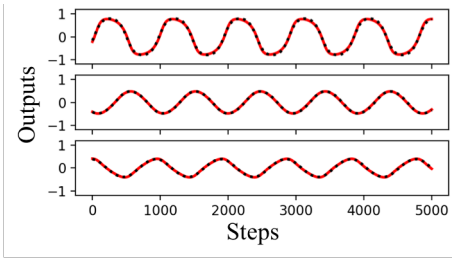


(b) Weights (training phase)

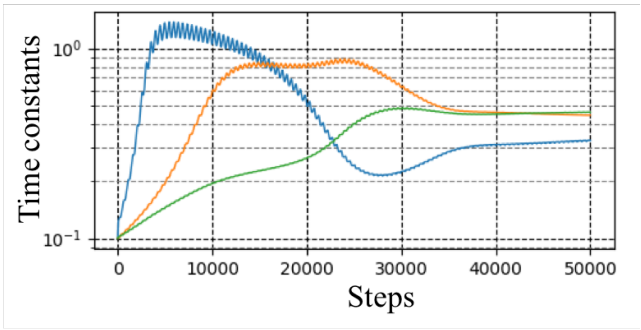
Fig. 4. The results in Task 2. Figure (a) shows the outputs of the learning network (red line) and the desired output generated by teacher network (dotted line) during test phase. Figure (b) shows the connection weights of the learning network during training phase.

Task 3: As shown in Fig. 5, although the time constant and connection weights of the learning network converge to different values from the teacher network, the learning network successfully learned. This result suggests that there are multiple solutions to generate the desired output. Considering that the network failed to learn with the same initial condition in the Task 2, these results show the limitation of weight modification in potential of learning pattern generation without optimizing time scale and the advantage of the time constant modifi-

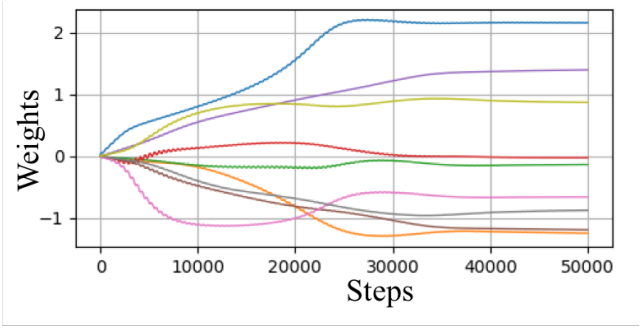
cation in learning. In this task, our proposed method expands learning into the time axis and greatly increases the learning performance.



(a) Outputs (test phase)



(b) Time constants (training phase)



(c) Weights (training phase)

Fig. 5. The results in Task 3. Figure (a) shows the outputs of the learning network (red line) and the desired output generated by teacher network (dotted line) during test phase. Figure (b), (c) show the time constants and the connection weights of the learning network during training phase respectively.

4 Conclusion

This paper proposed a learning method to update the time constant in dynamical neurons when a teacher signal is provided to each neuron. It is demonstrated that in a three-neuron oscillator reproducing task, each neuron in the leaning network can modify the time constant and the network can reproduce the output pattens generated by the teacher network. When we initialized the leaning network with the same connection weights as teacher and fixed, the time constant of each neuron in the learning network converges to the same value as the corresponding one in the teacher network. When the connection weights are modified as well as the time constants, the output can be reproduced, although the weights and time constants are not the same as the teacher network. However, when the time constants are set to different values from the teacher network and fixed, learning failed. This suggests the possibility that leaning of time constants expands the degree of freedom and improve the learning performance drastically.

Our future work includes expanding our proposed method with backpropagation through time for training multi layered CTRNN and challenging more difficult tasks than this study in which the network is required generating complex output patterns or processing time-series inputs to behave properly.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP15K00360, JP18H00543.

References

1. Y.LeCun, Y.Bengio, G.Hinton : Deep learning. Nature 521, 436-444 (2015)
2. K.Doya and Y.Shuji: Adaptive neural oscillator using continuous-time back-propagation learning. Neural Networks 2.5, 375-385 (1989)
3. Y.Yamashita and J.Tani.: Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment. PLoS Computational Biology 4, Vol.11, (2008)
4. J.Namikawa, R.Nishimoto, J.Tani: A neurodynamic account of spontaneous behaviour. PLoS Computational Biology, Vol.7, Issue.10 (2011)
5. M.C.Mozer: Induction of Multiscale Temporal Structure: Advances in Neural Information Processing Systems 4: 275 - 82. (1992)
6. H.Jaeger: Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach. Tech- nical Report GMD Report 159, German National Research Center for Information Technology. (2002)
7. D.Sussillo, L.F.Abbott: Generating coherent patterns of activity from chaotic neural networks. Neuron Article, Vol.63, No.4, pp.544-557(2009)