Reinforcement Learning Control using Self-Organizing Map and Multi-layer Feed-Forward Neural Network

Jae-Kang, Lee*, and Il-Hwan, Kim**

* Department of Control & Instrumentation Engineering, Kangwon National University, Korea
(Tel : +82-33-251-8789; E-mail: margrave@cclab.kangwon.ac.kr)
**Department of Electrical & Computer Engineering, Kangwon National University, Korea
(Tel : +81-33-250-6347; E-mail: ihkim@kangwon.ac.kr)

Abstract: Many control applications using Neural Network need a priori information about the objective system. But it is impossible to get exact information about the objective system in real world. To solve this problem, several control methods were proposed. Reinforcement learning control using neural network is one of them. Basically reinforcement learning control doesn't need a priori information of objective system. This method uses reinforcement signal from interaction of objective system and environment and observable states of objective system as input data. But many methods take too much time to apply to real-world. So we focus on faster learning to apply reinforcement learning control to real-world. Two data types are used for reinforcement learning. One is reinforcement signal data. It has only two fixed scalar values that are assigned for each success and fail state. The other is observable state data. There are infinitive states in real-world system. So the number of observable state data is also infinitive. This requires too much learning time for applying to real-world. So we try to reduce the number of observable states by classification of states with Self-Organizing Map. We also use neural dynamic programming for controller design. An inverted pendulum on the cart system is simulated. Failure signal is used for reinforcement signal. The failure signal occurs when the pendulum angle or cart position deviate from the defined control range. The control objective is to maintain the balanced pole and centered cart. And four states that is, position and velocity of cart, angle and angular velocity of pole are used for state signal. Learning controller is composed of serial connection of Self-Organizing Map and two Multi-layer Feed-Forward Neural Networks.

Keywords: Reinforcement Learning, Neural Network, Self-Organizing Map, Neural Dynamic Programming

1. INTRODUCTION

Reinforcement learning control does not need a priori information about the objective system. Several researches validated the reinforcement learning control method[1],[2],[3],[4].Those reinforcement learning control methods which use neural network could accomplish control task, but it took too much learning time to apply in real-world control task. Learning is accomplished in neural network part in reinforcement learning control which use neural network. It means that improvement of neural network learning time is improvement of entire controller. Reducing learning data is important element in improving learning time of neural network. If we have large number of learning data, neural network needs much learning time, but can learn more exactly. On the contrary, if we have few learning data, neural network can be learned in a short time, but probability of failure becomes higher. So we have to use proper learning data for fast and exact learning of neural network.

In reinforcement learning, two kinds of learning data are used for neural network controller. One is reinforcement signal which is obtained from interaction between control objective and environment. It has only two fixed scalar values that are assigned for each success and fail state. The other is observable state data. There are infinitive states in real-world system. So when we apply these states to neural network, we need to reduce the number of states for faster learning.

In this paper, we used SOM(Self-Organizing Map) to improve learning time without drop of learning success rate. And also we used NDP(neural dynamic programming) as learning algorithm. In this algorithm, system states associated with the positive reinforcement is memorized through a network learning process where in the future, similar states will be more positively associated with a control action leading to a positive reinforcement.

2. SELF-ORGANIZING MAP

SOM is a kind of neural network. The SOM usually consists of an array of units arranged in a grid. Associated with each unit, t, $w^t = [w_1^t, w_2^t, \ldots, w_D^t]$ is a weight vector where D is the dimensionality of the input data. The aim is to find a suitable set of weights for each unit so that the network models the distribution of the input data in the input space (also the weight space).

The learning rule responsible for finding a suitable set of weights is simple: given an input vector, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$, the distance between each unit, *t*, of the SOM and that vector is calculated by:

$$\sum_{d=1}^{D} (x_d - w_d^t)^2 , \tag{1}$$

The unit with the smallest distance is that which most closely represents the current input, and is thus considered the winner for that input. The weights of the winning unit are now updated toward that input:

$$\mathbf{w}^{\text{winner}} = \mathbf{w}^{\text{winner}} + \beta(\mathbf{x} - \mathbf{w}^{\text{winner}}), \tag{2}$$

where β is the learning rate. In addition to the winning unit being updated towards the current input vector, the winning unit's neighbors are also moved in this direction but by an amount that decays with the distance of those neighbors from the winning unit. The neighborhood function which controls this is often normally distributed around the winning unit, but for the purposes of the following experiments a very simple linear neighborhood will be used.

The weights of the map are initialized to random values and then the above process is iterated for each input vector in the data set, effectively resulting in a competition between different regions of the input space for units of the map. Dense regions of the input space will tend to attract more units than sparse ones, with the distribution of units in weight space

ultimately reflecting the distribution of the input data in the input space.

Fig. 1 shows the input data of SOM. This data will be classified by SOM. Fig. 2 shows randomly initialized SOM unit. Fig. 3 shows trained SOM by input data of Fig. 1.

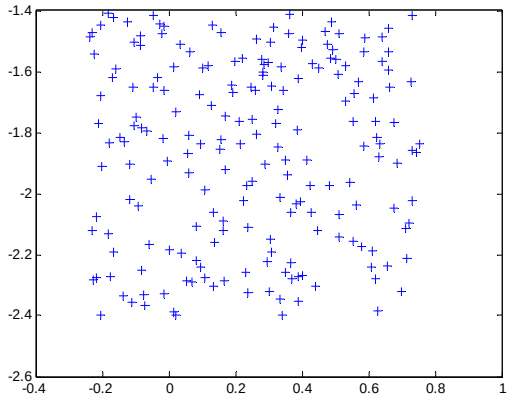

Fig. 1 Input data.

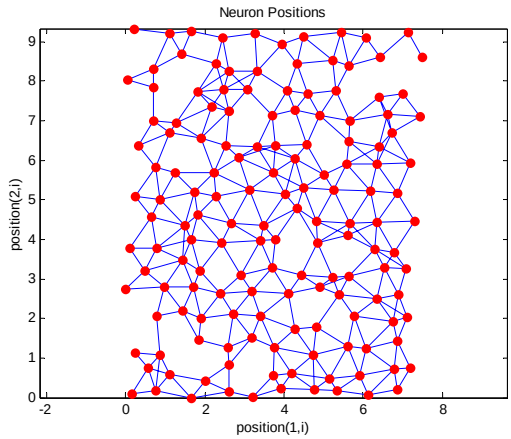

Fig. 2 Initialized SOM units

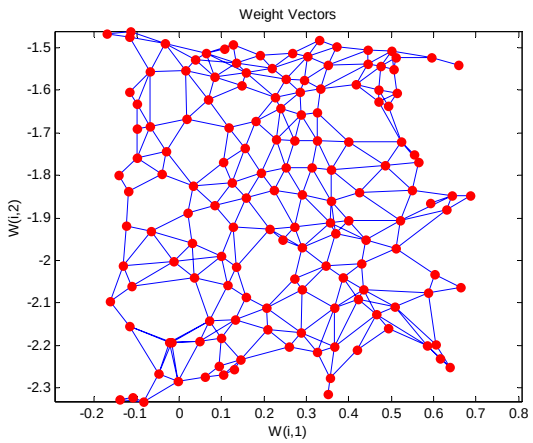

Fig. 3 Trained SOM.

3. LEARNING ALGORITHM

Fig. 4 is a block diagram of controller. It consists of three major part, namely action network, critic network, and SOM.

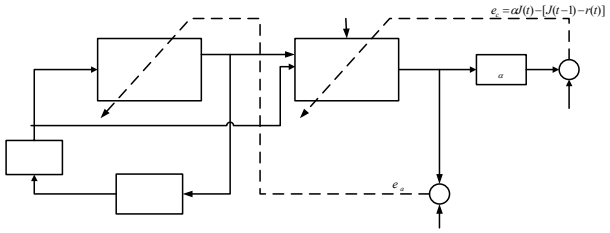

Fig. 4 Block diagram of controller

The reinforcement signal *r(t)* is provided from the external environment and it means "success" or "failure". The controller is "naïve" when it just starts to control, namely the action network and the critic network are both randomly initialized in their weights and parameters. Once a system state is observed, an action will be subsequently produced based on the parameters in the action network. A "better" control value under the specific system state will lead to a more balanced equation of the principle of optimality. This set of system operations will be reinforced through memory or association between states and control output in the action network. Otherwise, the control value will be adjusted through tuning the weights in the action network in order to make the equation of the principle of optimality more balanced.

The output of critic network, the *J* function, approximates *R(t)* at time *t* given by

$$R(t) = r(t+1) + \alpha r(t+2) + \cdots, \tag{3}$$

where *R(t)* is the future accumulative reward-to-go value at time *t*, α is a discount factor of the infinite-horizon problem ($0< \alpha <1$). $r(t+1)$ is the external reinforcement value at time $t+1$.

3.1 The Critic Network

The critic network const of five inputs, hidden layer with six neurons and output layer with one neuron. And learns function *J* which approximate *R(t)*. We define the prediction error for the critic network as

$$e_c(t) = \alpha J(t) - [J(t-1) - r(t)], \tag{4}$$

and the objective function to be minimized in the critic network is

$$E_c(t) = \frac{1}{2} e_c^2(t), \tag{5}$$

The weight update rule for the critic network is a gradient descent rule.

$$w_c(t+1) = w_c(t) + \Delta w_c(t), \tag{6}$$

$$\Delta w_c(t) = l_c(t)\left[-\frac{\partial E_c(t)}{\partial w_c(t)}\right], \tag{7}$$

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \left[-\frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_c(t)} \right], \quad (8)$$

where $l_c(t) > 0$ is the learning rate of the critic network at time t, which usually decreases with time to a small value, and w_c is the weight vector in the critic network.

3.2 The Action Network

The action network const of four inputs, hidden layer with six neurons and output layer with one neuron. The action network learns through error between the desired ultimate objective, denoted by Uc, and the approximate J function from the critic network. The weight updating can be formulated as follows. Let

$$e_a(t) = J(t) - U_c(t), \quad (9)$$

The weights are updated to minimize the following performance error measure:

$$E_a(t) = \frac{1}{2} e_a^2(t), \quad (10)$$

The weight update rule for the action network is a gradient descent rule.

$$w_a(t+1) = w_a(t) + \Delta w_a(t), \quad (11)$$

$$\Delta w_a(t) = l_a(t) \left[-\frac{\partial E_a(t)}{\partial w_a(t)} \right], \quad (12)$$

$$\frac{\partial E_a(t)}{\partial w_a(t)} = \left[-\frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial u(t)} \frac{\partial u(t)}{\partial w_a(t)} \right], \quad (13)$$

where $l_a(t) > 0$ is the learning rate of the action network at time t, which usually decreases with time to a small value, and w_a is the weight vector in the critic network.

4. SIMULATION

The inverted pendulum on a cart system model used in this paper is following Eq. (14), (15).

$$\frac{d^2\theta}{dt^2} = \frac{g \sin\theta + \cos\theta[-F - ml\dot{\theta}^2 \sin\theta + \mu_c \operatorname{sgn}(\dot{x})] - \frac{\mu_p \dot{\theta}}{ml}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}\right)}, \quad (14)$$

$$\frac{d^2x}{dt^2} = \frac{F + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta] - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}, \quad (15)$$

where

g	9.8 m/s^2, the acceleration of gravity;
m_c	1.0 kg, mass of cart;
m	0.1 kg, mass of pole;
l	0.5 m, half of pole length;
μ_c	0.0005, coefficient of friction of cart on track;
μ_p	0.000002, coefficient of friction of pole on cart;
F	±10 N, force applied to cart's center of mass;

We can observe four state variables of system: 1) $x(t)$, position of cart on the track; 2) $\theta(t)$, angle of the pole with respect to the vertical position; 3) $\dot{x}(t)$, velocity of cart; 4) $\dot{\theta}(t)$, angular velocity of pole. In our simulation, the range of track which the cart can move is [-2.4m, +2.4m] in reference to the center position of track. And the controllable range of pole is [-12º +12º]. The pole is considered fallen when the pole is outside of controllable range and/or the cart is bumped to the boundary of track. We used 0.02 sec for time step. If the pole lasted for 6000 steps(2 min.), we considered as success. We classified entire four states with SOM. And the success rate of learning was too low. The reason was too sensitive two velocity states with respective to discretization error. So we only classified position of track and angle of pole with 10 x 15 SOM. Two velocity states are applied directly.

We accomplished two types of experiment. One is neural dynamic programming only controller. The other is neural dynamic programming with SOM controller. Table. 1 is the result of 10 experiments using neural dynamic programming only controller. Table 2 is the result of 10 experiments using neural dynamic programming with SOM controller.

Table 1 Result of NDP only controller

Success learning rate	Trials until success of learning			
	Min.	Max.	Average	Standard deviation
100%	4	58	32.8	36.2

Table 2 Result of NDP with SOM controller

Success learning rate	Trials until success of learning			
	Min.	Max.	Average	Standard deviation
100%	8	48	29.4	21.4

If we consider the real inverted pendulum on a cart system, 4 trials means that we only have to make straight up 4 times until successful learning of controller. We think that it is enough to apply to real control system.

And we can see average learning time of the NDP with SOM controller is faster than that of NDP only controller. Also the standard deviation of learning time of NDP with SOM controller is smaller than NDP only controller.

5. CONCLUSION AND DISCUSSION

In this paper, we proposed a reinforcement learning control method using NDP algorithm and SOM. And an inverted pendulum on a cart system was simulated to validate the method. The result shows that the NDP algorithm is fast enough to apply to real system. And faster and less varying learning time is accomplished.

Using SOM to classify the learning data needs more computation. It means that the more computation time is needed. Because the inverted pendulum on a cart system is relatively simple system, the additional computation by SOM takes large part in computation. But if we apply SOM to more complicate system, additional computation for SOM will take small part. This will induce even faster learning than in simple system.

REFERENCES

[1] Sutton, R. S., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE*

Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, pp. 835-846, 1983.

[2] Anderson C. W, "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control Systems Magazine*, Vol 9, pp. 31-37, 1989.

[3] Anderson C. W, "Strategy Learning with Multilayer Connectionist Representations," *Proceedings of the Fourth International Workshop on Machine Lerning*, pp. 103-114, 1987.

[4] M. Wiering, R. Salustowicz, and J. Schmidhuber, "CMAC Models Learn to Play Soccer," *Proceedings of the 8th International Conference on Artificial Neural Networks*, 1998.

[5] Andrew James Smith, "Applications of the self-organising map to reinforcement learning", *In Neural Networks (Special Issue)*, 15, pp. 1107-1124, 2002.

[6] Jenni Si and Yu-Tsung Wang, "On-Line Learning Controlby Association and Reinforcement," *IEEE Transactions on Neural Network*, Vol. 12, No. 2, 2001.

[7] Santamaria, J. C., Sutton, R. S., and Ram, A., "Experiments with reinforcement learning in problems with continuous state and action spaces,", *Adaptive Behavior*, Vol 6, pp. 163-218, 1998.

[8] Dean F. Hougen, Maria Gini, and James Slagle. "Partitioning Input Space for Reinforcement Learning for Control," *In Proceedings of the IEEE International Conference on Neural Networks*, pp. 755-760, 1997