Dynamical Control by Recurrent Neural Networks through Genetic Algorithm

Toru KUMAGAI* **,
Mitsuo WADA***, Sadayoshi MIKAMI***, and Ryoichi HASHIMOTO*
*National Institute of Bioscience and Human-Technology
**Hokkaido University Graduate School, ***Hokkaido University
1-1 Higashi, Tsukuba 305 Japan, e-mail:kuma@nibh.go.jp

1. Introduction

Dynamic control problems often require a set of control rules. For example, an inverted pendulum system requires two different control rules for swinging up and stabilization of a pendulum. Recurrent neural networks (RNNs) are potential candidates for service as controllers of such complex tasks. RNNs memorize, recall and discriminate time-series information in a parallel way, extracting information from large amounts of real-world data, deciding subsequent behavior, and controlling the actuator to achieve suitable dynamic movement. For instance, we have shown that a recurrent network comprising binary output neuron can acquire a set of time-dependent limit cycles in a small network [3]. However, there is no powerful design method for recurrent neural networks. We compose a genetic algorithm that uses copy operators for recurrent neural network learning. The copy operator copies one part of a gene to another part. We show that the proposed algorithm accelerates learning in comparison with a simple genetic algorithm; also, it can embed two different control rules in a single recurrent network.

2. Conventional Learning Method for Recurrent Neural Networks

Supervised learning methods studied for the RNNs, i.e., back-propagation through time (BPTT) [1] and real-time recurrent learning (RTRL) [2] are not applicable for most problems because they require teaching signals. Hill-climbing methods also suffer from the local minimum problem. Reinforcement learning is sometimes powerful in acquiring a classifier system, yet that method is not suficiently powerful to learn a set of sequences, especially sequences including context-sensitive conditional branches.

The genetic algorithm, a potential learning technique for RNNs, offers advantage that it can search an entire solution space without the local minimum problem. However, its disadvantage is that it consumes an enormous amount of time in network learning. Many researchers have attempted to apply the genetic algorithm to neural network learning [5]. Thereby, they showed that the learning speed of a simple genetic algorithm is extremely inefficient when a network is large. One reason is that the crossover operator often makes a weak descendant because a neural network can assume different structures for a certain function [6]. The genetic algorithm must also repeatedly learn the same functions. A complex network is assumed to comprise many functional blocks, some of which appear repeatedly in a network that is composed of many neurons that are primitive function units. The simple genetic algorithm has no operator to copy functional blocks; therefore, it must re-learn the same function block repeatedly.

3. Genetic Algorithm with Copy Operator

We introduced a genetic algorithm operator, internal copy, responsible for copying the block of loci in each gene [7]. This operator copies a function block that has been acquired by learning and accelerates learning. These copied function blocks are expected to become seeds of a new function block. A set of copied units may also compose other function blocks. We also introduced an interindividual copy operator that copies one part of a gene to another part of another gene. Next, we define the gene encoding and algorithm used in this study (Fig.1). The gene is composed of loci. Each locus contains a threshold value and weight values of connections from neurons and external input. The training algorithm with internal copy operator is as follows:

1) Initialize all genes in the population with a random function.

2) Duplicate all genes. Note that “duplicated” genes are “copied” and “original” genes are “original”.
3) Apply one operator to each copied gene. The applied operator is chosen from a set of operators with equal probability for each gene.
4) Evaluate each gene with evaluation function $f()$ and (1).

$$eval = f()(1 + rnd(k_n)), \quad (1)$$

where $rnd(k_n)$ is an equally random function whose range is $(-k_n, k_n)$. In this study, $k_n = 0.1$.
5) Select half of the genes for the next generation in sequence of performance.
6) Go to 2)

In stage 3) of the above algorithm, we used the following four copy operators: mutation, crossover, internal copy, and interindividual copy.

(a) Mutation: This operator modifies each element in a gene with probability *m* according to the following equation:

$$\Delta\omega = rnd(k_\omega), \quad (2)$$

where $\Delta\omega$ is the quantity of modifications of each parameter, and $rnd(k_\omega)$ is an equally random function whose range is $(-k_\omega, k_\omega)$. In this study, $k_w = 0.5$.
(b) Internal copy: The internal copy operator copies a consecutive block of loci to another portion of the same gene. A source and destination are chosen by random number. The length of a copied locus block is determined as follows.

$$locilength = Rnd(NB) \quad (3)$$

where $Rnd(NB)$ is an equally random integer function whose range is $[1, NB]$.
(c) Interindividual copy: The interindividual copy operator copies several loci to a destination as in internal copy, but a source gene is chosen from another gene with a random function.

Operators are denoted as M, IC and II. We denote a two-point crossover operator as C. The combination of operators is expressed through +, e.g., M+IC means the combination of mutation and internal copy operators.

4. Swinging up and Stabilization of a Pendulum

We apply the proposed algorithm to the control problem of an inverted pendulum. The goal of task is to swing up and stabilize a pendulum at an inverted position. In this task, we ignore the position of the cart. However, solving the problem by learning remains difficult because swinging up and stabilization of a pendulum require two completely different control rules. A controller must realize these two control rules continuously.

4.1 Recurrent Network

We assume that a neural network consists of a time-discrete binary output leaky integrator model, i.e., Caianiello-Nagumo neuron model [3][4] because this network model acquires different types of limit cycles in a simple network. The binary output network must use many neurons to express a

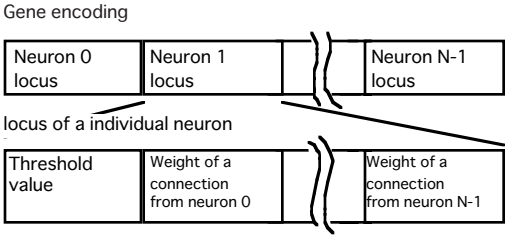

Fig. 1 Gene encoding

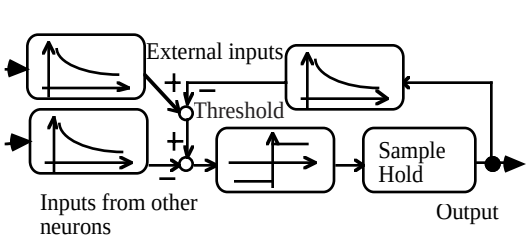

Fig. 2 The neuron model

continuous value, meaning that information is continuously and distributively memorized in many neurons. Neuron output is fed back and the neuron has a refractory period (Fig.2). Each neuron has mutual connections and receives stimuli from an external input. Incoming data are weighted, integrated by the leaky integration way inside the connection path [3], and then summed. The result is compared with a threshold. Each neuron updates its output value simultaneously as follows:

$$x_i(k+1) = \sum_{j=1}^{M} \mathbf{v}_{ij} \sum_{r=0}^{k} T^r u_j(k\text{-}r) - \sum_{j=1}^{N} \mathbf{w}_{ij} \sum_{r=0}^{k} T^r y_j(k\text{-}r) - h_i, \quad y_i(k) = f_s(x(k)), \tag{4}$$

where $f_s(x)$ is the function that outputs 0 if (x<0) or outputs 1 if (x>=0), $y_i(k)$ is the output value of neuron i at time k, $u_j(k)$ is the external input value to neuron j, w_{ij} is the weight of connection from neuron i to neuron j, and v_{ij} is the weight of connection from external input i to neuron j. The threshold value of neuron i is h_i, and T (0<=T<1; in this paper, T=0.5) is the decay parameter. $x_i(k)$ is defined in (4); we call this the neuron state i. N is the number of neurons. M is the number of external inputs.

4.2 Inverted Pendulum System

The following constitute the equation of motion of an inverted pendulum system and parameters which are used here.

$$M\ddot{x} + mL\ddot{\theta}\cos\theta - mL\dot{\theta}^2 \sin\theta + C\dot{x} = u \tag{5}$$

$$mL\ddot{x}\cos\theta + J\ddot{q} + p\dot{\theta} - mgL\sin\theta = 0 \tag{6}$$

$$m : 0.015[\mathrm{kg}] \quad L : 0.12[\mathrm{m}] \quad J : 2.58 \times 10^{-4}[\mathrm{kg \cdot m^2}]$$

$$p : 0.62 \times 10^{-4}[\mathrm{kg \cdot m^2/s}] \quad M : 0.625[\mathrm{kg}] \quad C : 13.6[\mathrm{Ns/m}]$$

The manipulated value u is calculated as follows.

$$net_{out} = \sum_{j=1}^{N} y_j(t) \tag{7}$$

$$u(t) = N \cdot 5/6 \quad (if\ net_{out} > N \cdot 5/6)$$
$$(net_{out}) \quad (if\ N \cdot 5/6 > net_{out} > N/6)$$
$$N/6 \quad (if\ N/6 > net_{out}) \tag{8}$$

Learning conditions are:

NB=4, N=12. (9)

Instead of θ, $\sin(\theta)$ and $\cos(\theta)$ are given to a network as external inputs. We give a network two additonal binary inputs. We assign 1-0 when $|\theta| < 0.3$, 0-1 when $|\theta| \geq 0.3$.

The following are evaluation functions.

$$f_k = \begin{cases} (1 - |\theta|) & \text{if } |\theta| < 0.2 \\ 0 & \text{others} \end{cases} \tag{10}$$

$$f_{sw}(\theta_0) = (\pi - \min(|\theta|)) + \int f_k \mathrm{d}t$$
$$\{\theta, t \mid \theta(0) = \theta_0, 0 \leq t < 10\} \tag{11}$$

The above evaluation function is very simple, but it is slightly complicated by formula expression. In eq. (11), the first term of the right part evaluates swinging up; and the second term evaluates how long a pendulum is in the top position. θ_0 gives the initial value of θ. Through experimentations, population size is determined as 12.

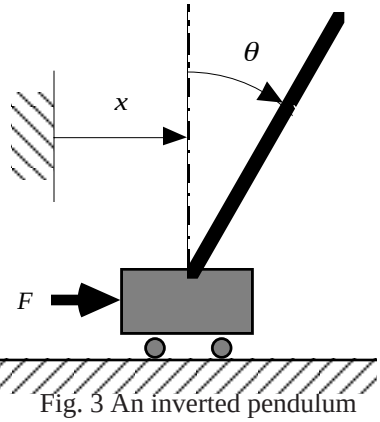

Fig. 3 An inverted pendulum

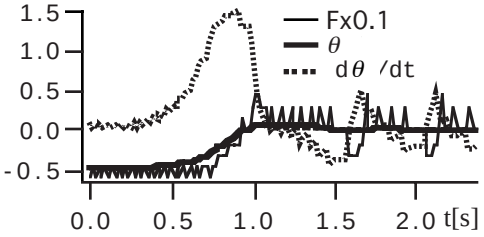

Fig. 4 Stabilization of the pendulum

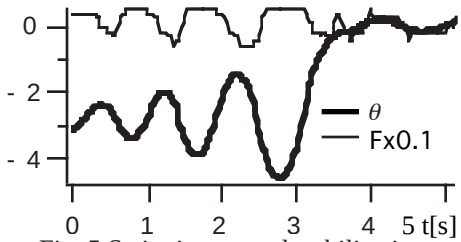

Fig. 5 Swinging up and stabilization

4.3 Results of experiments

(a) Stabilization of an inverted pendulum

First we applied the proposed genetic algorithm to stabilization of a pendulum with the following evaluation.

$$f = f_{sw}(-0.30) + f_{sw}(0.30) \quad (12)$$

Then the network stabilized the pendulum (Fig. 4). Figure 6 (left) shows how many generations are required for stabilization in each set of genetic operators. Clearly, internal copy and interindividual operators accelerate learning.

(b) Swinging up and stabilization of a pendulum

We applied the proposed algorithm to a network which had already learned how to stabilize the pendulum by the above experiments with the following evaluation equation.

$$f = f_{sw}(\pi) + f_{sw}(-0.30) + f_{sw}(0.30) \quad (13)$$

The network acquired the rules to swing up and stabilize a pendulum at top position (Fig. 5). Learning is accelerated by interindividual and internal copy operators again (Fig. 6 (right)).

5. Projected Work

This study introduced two copy operators: one is internal copy and the other is interindividual copy. These operators worked effectively in dynamic control problems. However, the effectiveness of the copy operators, probably depends on gene coding and how gene blocks are chosen and copied. Intuitively, it is possible to increase algorithm capability by regulating the manner of copying; e.g., one of the most general ways is to encode how copying is to be done in the gene.

References [1] D.E.Rumelhart, G.E.Hinton, and R.J.Williams, Learning internal representations by error propagation, Parallel Distributed Processing Vol. 1, MIT Press, pp. 318-362, 1986. [2] K.Doya and S.Yoshizawa, Adaptive neural oscillator using continuous-time back-propagation learning, Neural Network, vol. 2, pp. 375-386, 1989. [3] T.Kumagai, R.Hashimoto, and M.Wada, Learning of limit cycles in discrete-time neural network, Neurocomputing - An international Journal, vol.13, pp.1-10, 1996. [4] S.Gardella, T.Kumagai, R.Hashimoto, and M.Wada, On the dynamics and applications of a discrete time binary neural network with time delay, Journ. Intell. and Fuzzy Sys., Vol. 2, No. 6, pp. 243-250, 1994. [5] V.Petridis, S.Kazarlis, and A.Papaikonomou, A genetic algorithm for training recurrent neural networks, Proc. of the IJCNN'93, pp. 2706-2709, 1993, Nagoya. [6] P.J.Angeline, G.M.Saunders, and J.B.Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. on Neural Networks, Vol. 5, No.1, pp. 54-65, 1994. [7] T.Kumagai et al., Structured Learning in Recurrent Neural Network Using Genetic Algorithm with Internal Copy Operator, ICEC97 (1997)(accepted).

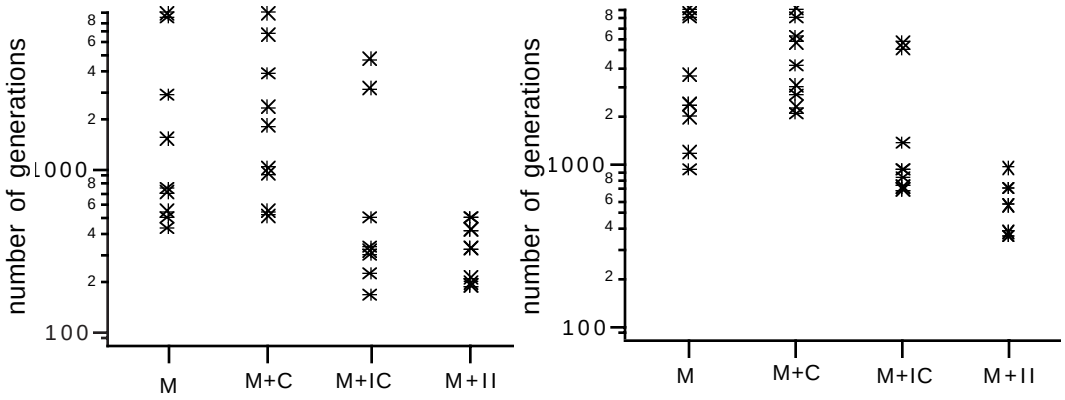

Fig. 6 (Left) The number of generations required to stabilize a pendulum at top position. (Right)The number of generations required to swing up and stabilize a pendulum at top position. Ten experiments were conducted for each set of operators. While M was applied, the network could not stabilize a pendulum in 3 of 10 experiments (not plotted).