Brain computation system that searches for internal procedures by functional parts combination

Takashi Omori, Akitoshi Ogawa
Graduate School of Information Science & Technology
Hokkaido University
omori@complex.eng.hokudai.ac.jp

Abstract: A learning architecture for intelligent system based on brain system nature is proposed. To reduce large amount of learning time that is necessary for conventional methods, we introduce reuse of knowledge that are acquired through past experiences. The architecture enables immediate adaptation to a new task that is similar to well experienced tasks. The effect is evaluated in a series of navigation tasks that becomes difficult stepwise.

keywords: Knowledge Reuse, Immediate adaptation, Neural Network, Genetic Algorithm, Navigation

1. Introduction

An intelligent agent in the real world confronts various tasks and environments that cannot be predicted in advance. The agent is required to solve the tasks efficiently in a short time. For this purpose, the agent needs two specific abilities. One is continuity of a mapping between processing system structures and emerging processing behaviors. The agent should be able to solve multiple tasks that are similar to but different from past learned ones. The other is immediate adaptation, with which the agent can solve a new task quickly. To realize the adaptation ability, it is necessary that the agent re-applies knowledge derived from past experiences to a new task. Because the tasks are unknown in advance, the knowledge for task solving must be acquired autonomously and incrementally from experiences.

Reinforcement Learning (RL) (Sutton & Barto, 1998) and Evolutionary Robotics (ER) (Nolfi & Floreano, 2000) are known as autonomous learning methods based on interaction with the environment. These methods, however, consume much time owing to the need to learn actions by trial and error. To shorten the learning time, reusing the results of past learning is efficient. However, RL and ER requires much time for re-learning new or changed tasks.

On the other side, brain reveals excellent learning ability by reusing past experiences and learning results. Many researchers will agree that brain assembles processing neural circuit for each of different tasks by selectively activating different functional parts in cortical areas. The mechanism of functional parts combination (FPC) looks to be specific computational principle for brain. At least now, however, we don't know much about FPC; what granuity of functional parts are used?, how their combination is searched?, what ability can be realized?, foe example.

In this paper, for seeking for the detailed property of FPC principle, we propose a new neural learning system that reuses knowledge of previously learned tasks to solve a new task. The system can control the topology of a neural network, construct module-like sub-neural networks (SNNs) in a self-organized manner, and form a neural circuit that generates an output action from an input of the new task. The system searches a combination of the sub-networks to form a circuit, in which the connection knowledge within the sub-network and the combination of the sub-networks are reused. The agent in our system reuses the knowledge of past tasks and exhibits fast adaptation to a new task. To verify the effectiveness of the proposed system, we apply it to a basic benchmark of robot navigation and compare its performance against other learning methods.

2. Knowledge-reusing neural learning system

2.1 Formation of input-output circuit SNN combination

Factors that determine the whole processing of the neural network under consideration are the topology of the network, the transfer function of each neuron, the learning rule, and weights of the SNN. Once first three factors are decided, learning according to the input from the environment decides the forth factor, and decides network's fine processing. The four factors are the knowledge for the task solving.

The neural network in this paper consists of three groups of input, intermediate and output layers. Input and output layers are prepared corresponding to multiple sensory inputs and action outputs. Intermediate layers that connect between the input and output layers are also prepared. The connection topology of the intermediate layers can be flexibly changed to form various input-to-output processing circuits. The layers are a set of SNNs that have the ability to learn its

internal connections. The system creates SNNs and makes them learn to acquire various types of knowledge for task solving.

The combination of SNNs sets up the whole neural network and decides the processing of the system. Task solving by the system is equivalent to creating and combining the SNNs to form a whole neural network that generates proper actions for any inputs from the task environment. We propose a following learning system.

A) The system creates SNNs relevant to a task by learning the weights in a self-organizing manner.
B) Searching connections between SNNs, the system forms a transfer function from input to putput.

2.2 Structure Matrix

The calculation process of the neural learning system from input to output is decided by the first three factors. We introduce a structure matrix to search the factors and learn the weights (Fig.1). If the number of an SNN is n, the structure matrix is an $n \times n$ regular matrix. SNNs in the input, intermediate and output layer groups are numbered in sequential order.

Each non-diagonal element in this matrix represents a relation between two SNNs; the row represents the input and the column represents the output. The upper-triangle part, except the diagonal part c_{ij} (i,j = 1,2,…,n, $i<j$) represents the existence of the connection between SNN_i and SNN_j and the learning rule of each connection. The diagonal part represents the transfer function of each SNN. For simplicity, loopy connections and direct connections between input SNNs and between output SNNs are not permitted due to constraining corresponding c_{ij} elements. The transfer functions of input and output layers are fixed by the physical nature of the input and output signal.

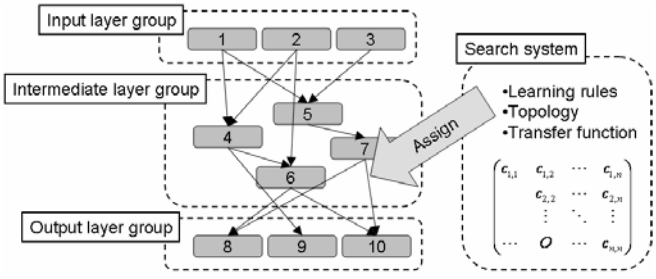

Fig.1 Overview of proposed method. The system assign three factors of knowledge using the structure matrix.

2.3 Searching the Structure Matrix and NN Learning

The system can solve a task by searching a structure matrix and by learning weights for the task. We adopted the genetic algorithm (GA) as the method to search the matrix. In the GA method, the structure matrix corresponds to a gene.

Some learning rules must be prepared for SNNs. The structure matrix assigns a learning rule for each of the SNNs, and each SNN updates its weight from the response of the whole network to data from the task environment.

The GA requires a fitness estimation of the matrix for search. We defined a fitness function corresponding to each gene which represents a structure matrix. The algorithm for task solving and knowledge acquisition for an individual task is summarized as follows.

1. The system creates a set of initial structure matrices or reuses the acquired matrix randomly.
2. Calculate the fitness of each individual.
3. If termination conditions are fulfilled, this algorithm terminates.
4. Crossover by a random pair and mutation on each element of the structure matrix is applied.
5. Go to 2

After the matrix searching and weight learning, we can obtain the best individual that has the highest fitness value. We then change the task to be solved.

The system generates the first generation genes for the new task based on the genetic operations. If the new task is similar to tasks already experienced, then the target structure matrix for the new task can be expected to be similar to the previously acquired ones. The search time for the new task will be shorter and it might even be possible to solve the new task immediately.

3. Simulation Experiments

3.1 Task and environment

We prepared a Khepera simulator that employs real observational data. The Khepera has infrared and visible-light sensors, six on the front and two on back, and two wheels positioned side-by-side. The simulator settings are that the sensory data is noisy, actions are forward, turn left and turn right, the forward speed is 1 cm/s, and the interval time for each action is 0.5 sec. Walls surround the experimental environment.

We prepared three tasks, A, B and C, shown in Fig. 2, 3 and 4 respectively. The starting coordinates $(x, y, \theta) = (5+5r_1, 5+5r_2, 2\pi r_3)$ are common to the tasks. Here, r_i is a uniform random number, $0<r_i<1$. In task A, the coordinates of the light are set to (60,25) with low intensity, and the dot-lined area that surrounds the light is the goal (Fig. 2). In task B, the coordinates of the light are set to (90,25) with high intensity, while the dot-lined area that surrounds the light is the goal

(Fig. 3). In task C, the light setting is the same as that in task B, though the goal is located at the center of the environment. Because the goal of task C is opposite the light in task B, the behavior toward the goal in task B interferes with the learning in task C

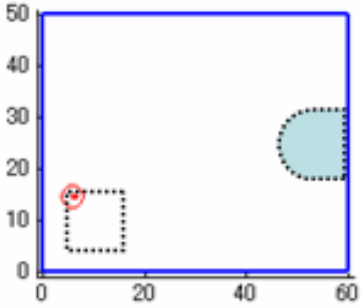

Fig. 2 Task A environment. See text for details. This is a screen of the Khepera simulator.

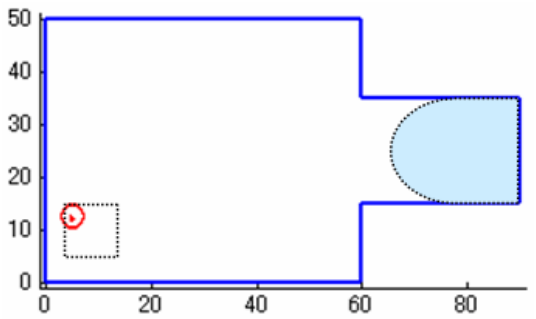

Fig. 3 Task B environment. See text for details.

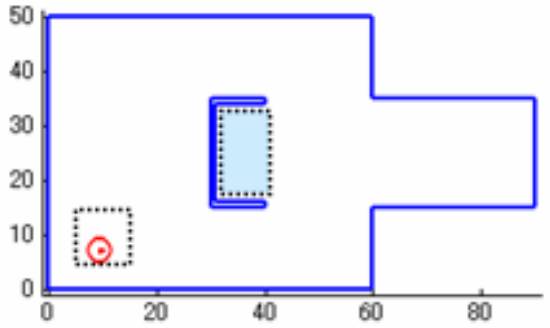

Fig. 4 Task C environment. See text for details.

We tried two task change cases to evaluate system adaptation ability by reusing previously learned knowledge.

Case I The agent acquires knowledge by learning task A, and reuses the knowledge when it encounters task B.

Case II The agent reuses the knowledge learned in case I when it encounters task C.

Because all the knowledge derived from task A can be reused in task B, an agent can adapt to task B if it acquires and properly combines some additional knowledge. In task C, however, some of the knowledge from the task B can be reused, but some interferes the adaptation.

3.2 Implementation of the learning system

We divide the Khepera's front sensors into six groups by direction and type. We prepare six input SNNs for the groups and six intermediate SNNs that contain two neurons. The input SNNs normalize the sensory information. Three output SNNs correspond to the actions: move ahead, turn right, and turn left. We have prepared the following two transfer functions for the neurons (a=1,b=1,c=2).

$$f(x) = \exp(-ax) \qquad (1)$$

$$f(x) = -1/\{1+\exp(-bx+c)\} \qquad (2)$$

The equation $x_i = \Sigma_j | y_j - w_{ij} |$ determines the inputs of the transfer function in the intermediate and the output SNNs, where the i-th SNN receives the j-th SNN's output y_{ij} and the weights of the connection are w_{ij}.

Weights are initialized with positive small numbers. We prepared the following three learning rules for the weights inside the SNN:

$$\Delta w = \eta (x - w) \quad \text{if } x > 0.5 \qquad \text{(i)}$$

$$\Delta w = \eta (x - w) \quad \text{if } x < 0.5 \qquad \text{(ii)}$$

$$\Delta w = \eta (x - w) \qquad \text{(iii)}$$

where η is a learning rate ($\eta = 0.1$). The weights are updated at every step. The action of the agent is decided as a result of the competition between the output SNNs.

The structure matrix is a 15×15 regular matrix, and the system includes 15 SNNs. Each element of the matrix has a value within [0 , 1] and designates the feature of a SNN as follows. If the diagonal element is less than 0.5, the transfer function of the corresponding SNN is (1), otherwise (2). Each element of the upper-triangle part of the matrix assigns the connection between SNNs and the learning rule of each connection; $c < 0.6$ designates no connection, $0.6 \leqq c < 0.7$ designates the learning equation (i), $0.7 \leqq c < 0.8$ designates (ii), $0.8 \leqq c < 0.9$ designates (iii) and $0.9 \leqq c$ designates fixed weights.

3.2.1 Task settings

The fitness of each individual is decided by average number of steps for six trials. The agent updates the weights throughout the trials. A trial ends when the agent reaches the goal, collides with an obstacle, or reaches timeout steps 800.

The parameters of GA are as follows: 20 for the population, 0.1 for the mutation rate, and 2 for the number of elites. One crossover point is used on the row of the structure matrix, and pairs for the crossover are selected randomly. Mutation is applied to every element of the matrix with a probability of 0.1. If the mutation operation removes a connection, the system removes the corresponding SNN weights, whereas if

the mutation operation generates a connection, the system generates a new SNN and initializes the weights randomly. Without mutation, weights are carried over generations.

In case I, task A switches to task B when the 31st generation starts. The system reuses the knowledge derived from task A at the first generation of task B, and the system keeps on searching. Case II is similar to case I. To demonstrate the efficiency of knowledge reuse, we compare case I with a case in which the same agent starts and continues learning in task B for 50 generations.

3.2.2 Results

Fig.5 shows the result of case I ; the transition of average steps by the best individual. The agent adapted to task B at the first generation following the task switching. Fig.6 shows the result of the task B-only case. The superior speed of our method is demonstrated by comparing these two figures. The peaks, indicated by O in figure 5, were caused by collisions with the corner near the goal.

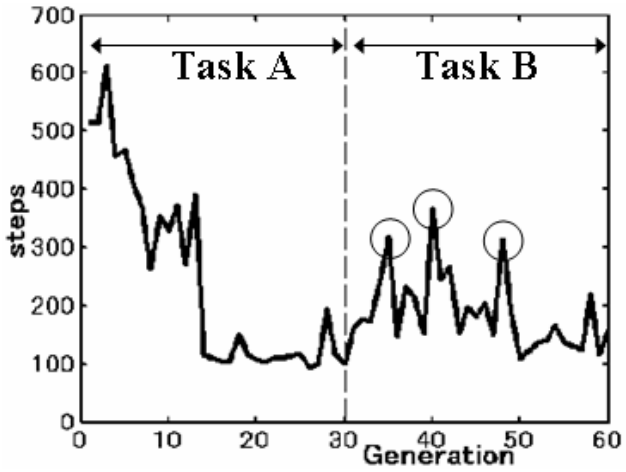

Fig.5 The result in case 1. The navigation agent acquired knowledge of task A in about the 14th generation and adapted to task B in the 1st generation of the search.

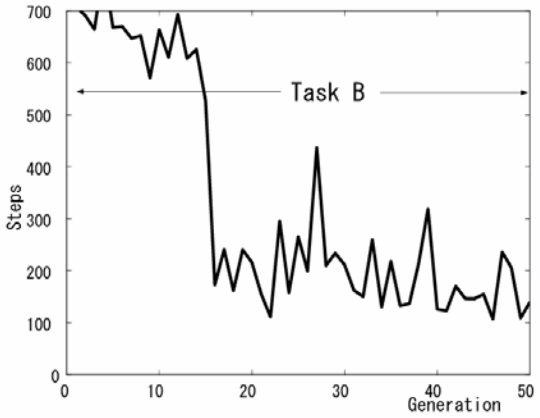

Fig.6 The result of task B, without reusing the knowledge from task A. The agent acquired the knowledge for task B at about the 16th generation.

Next, we investigate the knowledge-reuse state in case I. The structure matrix for task A had 93 effective elements. In task B, 12 of those elements were reused. These elements are related to obstacle detection and action selection, indicating that the agent reused the knowledge derived from task A in task B. In task A, the agent went ahead when no obstacle was detected. If an obstacle was detected, the agent performed "avoiding obstacles" by using a left-hand strategy and reached the goal. Then, in task B, the agent could perform "avoiding obstacles," learned in task A, to get near the goal area without learning; the agent only had to learn the behavior "approaching light" to reach the goal. Thus, the knowledge learned in task A, "avoiding obstacles," reduced the learning time in task B.

When the agent that has the knowledge of task A performs task B without learning, the agent does come near the goal area. It cannot, however, approach the goal because the agent does not have the knowledge "approaching light," since task A does not give the agent "approaching light."

Fig.7 shows the result of case II. The agent adapted at the 19th generation after the change to task C. The learning was slower than in case I because the agent had to acquire the knowledge "avoiding light" and inhibit the knowledge "approach light".

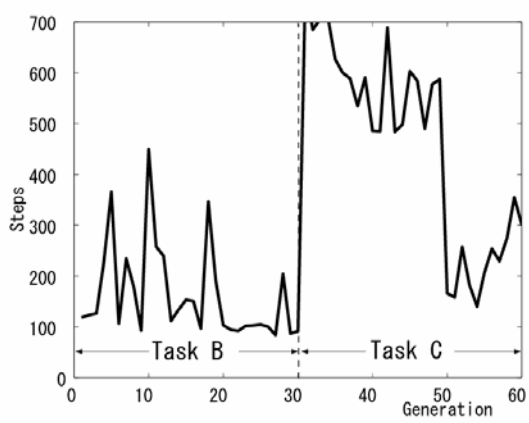

Fig.7 The result of case Ⅱ. The agent adapted at the 19th generation after the change to task C.

3.3 Evolutionary Robotics

We can categorize ER into two methods. One method uses genetic programming to acquire the calculation process directly, whereas the other method optimizes weights or learning rules of a neural network. We adopted the latter method because this method is similar to ours, and many ER researchers have also adopted it.

We prepared a feed-forward neural network that has input

and output layers. The IR sensor and the light sensor signal are provided to the input layers, while the output layer has three neurons corresponding to the actions of move forward, turn left, and turn right. The neuron that is fired by competition between them decides the action of the agent. We used a post-synaptic indirect encoding that does not encode weights directly but encodes learning rules and rates into genes (Floreano, 2000).

Learning rules are as follows:

$$\Delta w = \eta (x - w) \quad \text{if } x > 0.5$$
$$\Delta w = \eta (x - w) \quad \text{if } x < 0.5$$

where η is the learning rate ($0 < \eta < 1.0$) and is compromised in the gene.

Similarly to case I of our method, the agent adapts task A and acquires the gene of the elite. The final knowledge in task A is reused to generate the genes for task B. The fitness function and the condition for trial termination are the same as in our method. The parameters of GA are as follows: 20 for the population, 0.1 for the mutation rate, and 2 for the number of elites. All genes are generated randomly. One crossover point is used, with the crossover point and pair selection decided randomly. Learned weights are reset at every trial termination.

Fig.8 shows the transition of the maximum fitness in case I. The agent adapted task A faster than our method, but it could not adapt to task B. Consequently, we cannot evaluate the performance of case II.

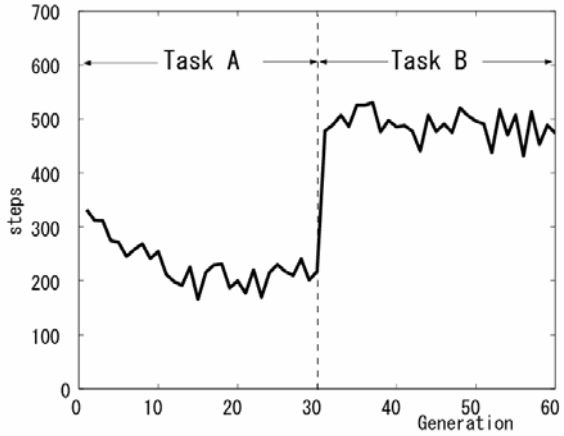

Fig.8 The result of ER in case I. See text for details.

3.4 Reinforcement Learning

We chose the actor-critic (λ) from many RL methods due to its adaptability to a continuous space and learning speed. We used the world coordinates (x,y, θ) for the learning tasks to avoid the perceptual aliasing problem, though the tasks become easier than with our method and that of ER. Actor-critic (λ) is a learning method of TD learning. We used the NRBF (Normalized Radial Basis Function) (Sato, 2000), where the base functions are Gaussian, to learn the state value and the action preference.

The agent learns linear parameters to the output of NRBF because the learning of nonlinear parameters to the output of the state value destabilizes the learning of the state value. The bases are deployed randomly within the task environment in advance.

The knowledge for task solving in this experiment is the NRBF parameters. This knowledge is reused in a new task by taking over the parameters derived from the tasks undertaken before. If additional bases need to be added to new state spaces for new tasks, the learning system adds new bases there at once before the new task starts. When the agent reaches a goal, a reward is given. The condition for trial termination is the same as in our method.

In task A, 200 bases are deployed randomly in the state space (x,y, θ). The agent learns task A for 5,000 trials and acquires the knowledge of the task. This knowledge consists of the parameters of the state value and action preference, and is reused in task B. Before task B begins, 50 bases are added to the additional state space. Task C requires no such additions. The agent thus performs 5,000 trials in tasks B and C respectively. We compare the result with that of our method. In this RL experiment, we evaluate the agent by the number of goals reached for every 100 trials.

Fig.9 shows that about 2,000 trials were necessary for the learning of task B in case I. It corresponds to 17 generations in our method. The result of case II is not shown because we could not increase the goal rate any way in task C.

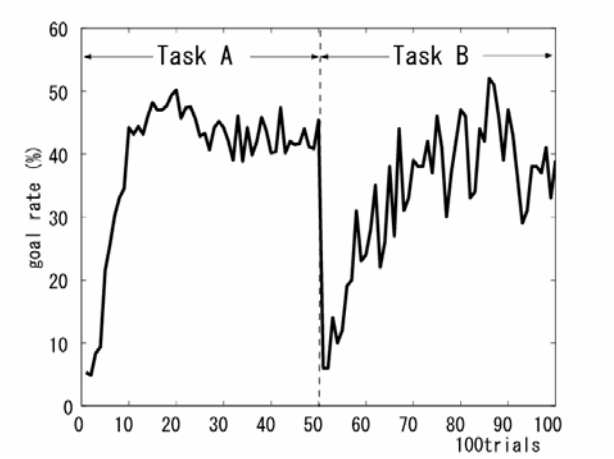

Fig.9 The result of RL in case I. Task A is switched to task B at the 5,000th trial. The goal rate converges at about the 2,000th trials after the switching.

4. Discussion

Our method is able to immediately adapt to the task because it can self-organize the configuration of SNNs and search for combinations of them. Since some SNNs compose a group of SNNs that are used in other tasks also, the building-block gene corresponding to the group reduces the number of possible combinations in a new task. Therefore, the agent only had to acquire the knowledge for task B that was not included in the knowledge for task A. That is why the agent could adapt to task B immediately. In case II, the agent altered some SNNs to inhibit the knowledge "approaching light" in task C. This demonstrates how flexibly our method can recompose SNNs if necessary.

Size and hierarchy of knowledge are two reasons why ER and RL learned a task more slowly or could not learn the task at all in the case of knowledge reuse. The basic ER method learns a neural network; however, ER cannot acquire partial processes, e.g. obstacle detection, and it cannot use the processes hierarchically. Therefore, it is difficult to both add and inhibit knowledge by this method. The knowledge of RL is the parameters of state value or action preference, and the knowledge is restricted to a small size. Therefore, it takes much re-learning time to add or inhibit knowledge. If we reduce the number of parameters, the perceptual aliasing problem occurs, making the learning task even more difficult.

From ER's point of view, the proposed system may be one type of ER method. We, however, consider our system is different from ER because almost no ERs approach the problem from the perspective of knowledge reuse between tasks.

5. Conclusion & Brain Like Computation

In this paper, we proposed a learning system that reuses knowledge from former tasks by controlling the topology of a neural network and constructing module-like sub-neural networks in a self-organized manner. We compared our learning system with ER and RL and evaluated its efficiency.

Intelligent agents in the real world have insufficient time to adapt to different tasks, and such agents should aggressively reuse knowledge acquired in past tasks. For the purpose, brain acquired ability of meta learning that assemble learning circuit corresponding to task requirements. This is what brain learning essentially differs from other learning system, and should be a principle of brain intelligence emergence. Motivation is one of important factors that drive the meta learning system, as is pointed by Gen Matsumoto. But the motivation is just a part of whole system and does not give concrete mechanism for intelligence emergence in brain. However, we know little of brain meta learning system. In the situation, this paper revealed ability of proposed system to be able to realize the meta learning behavior. More practical application is necessary to demonstrate effectiveness of the meta learning system.

REFERENCES

[1] Baird, L., Residual Algorithms: Reinforcement Learning with Function Approximation, Proc. of Twelfth Int. Conf. on Machine Learning, pp. 30-37 , 1995

[2] Bishop, C.M., *Neural Networks for Pattern Recognition*, 1995.

[3] Floreano, D., Urzelai, J., Evolutionary robots with on-line self-organization and behavioral fitness, Neural Networks, 13, 431-443, 2000

[4] Nolfi, S., Floreano, D., *Evolutionary Robotics*, 2000

[5] Ogawa, A., Omori, T., Looking for a suitable strategy for each problem –Multiple-tasks approach to navigation-learning tasks-, Proc. 2nd Int. workshop on Epigenetic Robotics, pp. 125-132, 2002

[6] Ogawa, A., Omori, T., The Acquisition of Space Search Procedure Depending on Agent Structure, Proc. of 1st Int. Symp. on Measurement, Analysis and Modeling of Human Functions, pp. 210-215, 2001

[7] Omori, T., Ogawa, A., Two hypotheses for realization of symbolic processing in the brain, Proc. ICONIP2001, pp. 1114-1119, 2001

[8] Sato, M., Ishii, S., On-line EM Algorithm for the Normalized Gaussian Network, Neural Computation, 12, 407-432, 2000

[9] Toga, A.W., Maziotta, J.C., *Brain Mapping -The systems-*, 2000

[10] Russell, S.J., Norvig, P., *Artificial Intelligence: A Modern Approach*, 1995

[11] Sutton, R.S., Barto, A.G., *Reinforcement Learning: An Introduction*, 1998