Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A Reinforcement Learning Model Using Macro-actions in Multi-task Grid-World Problems

Hiroshi Onda and Seiichi Ozawa
Graduate School of Engineering, Kobe University, Kobe, Japan
Email: human_kindjp@yahoo.co.jp, ozawasei@kobe-u.ac.jp

***Abstract*—A macro-action is a typical series of useful actions that brings high expected rewards to an agent. Murata et al. have proposed an Actor-Critic model which can generate macro-actions automatically based on the information on state values and visiting frequency of states. However, their model has not assumed that generated macro-actions are utilized for leaning different tasks. In this paper, we extend the Murata's model such that generated macro-actions can help an agent learn an optimal policy quickly in multi-task Grid-World (MTGW) maze problems. The proposed model is applied to two MTGW problems, each of which consists of six different maze tasks. From the experimental results, it is concluded that the proposed model could speed up learning if macro-actions are generated in the so-called correlated regions.**

***Index Terms*—reinforcement learning, macro-action, multi-task learning, neural network**

I. Introduction

To enhance the learning efficiency in reinforcement learning problems, several approaches to generating and utilizing macro-actions have been studied [1], [2]. A macro-action is a state-action sequence which brings high rewards to an agent for different but related tasks; that is, it should be

i) a useful state-action sequence leading to high rewards,
ii) a frequently observed typical state-action sequence,
iii) a state-action sequence useful for other tasks.

McGovern et. al [2], [3] empirically demonstrated that the learning is sped up by using macro-actions; however, they have not presented how to generate macro-actions automatically. Murata et. al [4] have proposed an Actor-Critic RAN-LTM (AC-RAN) model in which actor and critic are constructed by neural networks called *Resource Allocating Network with Long-Term Memory* (RAN-LTM). In Murata's AC-RAN, the first two properties of macro-actions are implemented by controlling the temperature in the softmax action selection. The temperature is determined by state values and visiting frequency of states. Since a sequence of states with high temperature are usually aligned along an optimal path in a grid-world problem after sufficient learning, an agent would take deterministic actions in the neighbor of the optimal path. We have claimed that such an action sequence leads to indirect creation of macro-actions. However, the created macro-actions in AC-RAN do not have the third property; thus, they cannot be utilized for learning other tasks.

Konidaris et. al [5] proposed a method to create macro-actions satisfying the third property in reinforcement learning, in which two types of options are separately defined in a *problem space* and a *agent space*. In the problem space, the agent's states satisfies the Markov property, but an optimal policy is learned for a specific task; thus, it is not used in other tasks. On the contrary, the states in an agent space might not satisfy the Markov property but a learned policy could be used in different related tasks. Konidaris et. al showed that the learning speed for correlated tasks is accelerated by using macro-actions whose states are defined in the agent space.

In this paper, we extend Murata's AC-RAN so that macro-actions satisfying all the properties above can be automatically created in grid-world maze problems by introducing Konidaris et. al's idea. In the extended model, a sequence of deterministic state-action pairs satisfying the first two properties are acquired by AC-RAN. Then, the acquired sequence is segmented into partial sequences based on the time variation of the distance information defined in the problem space. The rotation invariant matching is applied between the partial sequences and the macro-action candidates obtained in the past tasks. If there is a partial state-action sequence matched, a macro-action is created from the partial sequence and the matched macro-action candidate.

This paper is organized as follows. Section II gives a brief review on Murata's AC-RAN which can generate macro-actions satisfying the first two properties. In Section III, we propose an extended AC-RAN in which generated macro-actions satisfy all the properties above so that they can be utilized for learning different grid-world maze tasks. In Section IV, the extended AC-RAN is evaluated for two multi-task grid-world maze problems. Finally, the conclusions are given in Section V.

II. Murata's AC-RAN Model

Murata et. al [4] have proposed a reinforcement learning model called *Actor-Critic RAN-LTM* (AC-RAN) in which two RAN-LTMs are respectively used for Actor and Critic (see Fig.1).

Assume that an agent can select an action from a set $\mathcal{A} = \{a_1, \cdots, a_K\}$ in a state $\boldsymbol{x}(t) = \{x_1(t), \cdots, x_I(t)\}^T$ at time t. Here, T means the transpose of a vector or matrix. Based on the softmax strategy, the action preferences $P(\boldsymbol{x}(t), a_k)$ $(k = 1, \cdots, K)$ are calculated by the outputs of Actor RAN-LTM; then the selection probability $\Pr\{a(t)=a_k\}$ of a_k is calculated by Eq. (1).

$$\Pr\{a(t)=a_k\} = \frac{\exp\left(P(\boldsymbol{x}(t), a_k)/\tau(t)\right)}{\sum_{k'=1}^{K} \exp\left(P(\boldsymbol{x}(t), a_{k'})/\tau(t)\right)} \quad (1)$$

where $\tau(t)$ is a temperature parameter to control the randomness of action selection. On the other hand, the output of Critic RAN-LTM corresponds to a state value $V(\boldsymbol{x}(t))$.

A state $\boldsymbol{x}(t)$ is given to Actor and Critic RAN-LTMs as inputs; then, the outputs of hidden units $y_j(\boldsymbol{x}(t))$ are calculated as follows:

$$y_j(\boldsymbol{x}(t)) = \frac{\hat{y}_j(\boldsymbol{x}(t))}{\sum_{j'=1}^{J} \hat{y}_{j'}(\boldsymbol{x}(t))} \quad (j = 1, \cdots, J) \tag{2}$$

$$\hat{y}_j(\boldsymbol{x}(t)) = \exp\left(-\frac{\|\boldsymbol{x}(t) - \boldsymbol{c}_j\|^2}{\sigma_j^2}\right) \tag{3}$$

where J is the number of hidden units; $\boldsymbol{c}_j$ and σ_j are the center and width of the jth hidden unit, respectively. The outputs $z_k(\boldsymbol{x}(t))$ of RAN-LTM are given by

$$z_k(\boldsymbol{x}(t)) = \sum_{j=1}^{J} w_{kj} y_j(\boldsymbol{x}(t)) + \xi_k \quad (k = 1, \cdots, K) \tag{4}$$

where K is the number of outputs; w_{kj} and ξ_k are the connection weight and bias, respectively.

The outputs $z_k(\boldsymbol{x}(t))$ of Actor RAN-LTM are associated with the action preference $P(\boldsymbol{x}(t), a_k)$, and an action is selected based on Eq. (1). Then, the following Temporal Difference (TD) error is calculated using the reward $r(t+1)$:

$$e(t) = r(t+1) + \gamma V(\boldsymbol{x}(t+1)) - V(\boldsymbol{x}(t)) \tag{5}$$

where γ is a discount factor.

In Actor and Critic RAN-LTMs, not only the TD error but also the errors for the input-output relations in *memory items*, which are retrieved from the long-term memory (LTM) in Fig. 1, are minimized. These memory items are generated from a pair of representative inputs and outputs in the learning process and they are kept in LTM (see [6] for the details). A memory item in Actor is represented by a pair of a state and the action preference $(\boldsymbol{x}^{(l)}, P^{(l)}(a_k))$ $(l = 1, \cdots, N_i)$, On the other hand, A memory item in Critic is represented by a triplet $(\boldsymbol{x}^{(l)}, V^{(l)}, F^{(l)})$ $(l = 1, \cdots, N_{i'})$ where $F^{(l)}$ is the visiting frequency of a state $\boldsymbol{x}^{(l)}$. Here, N_i and $N_{i'}$ are the numbers of memory items in Actor and Critic RAN-LTMs, respectively.

In AC-RAN, the probability of an action is controlled by the temperature parameter $\tau(t)$ which is given by

$$\tau^{(l)} = \frac{\mu}{\varepsilon^{(l)}} \tag{6}$$

where μ is a positive constant. $\varepsilon^{(l)}$ in Eq. (6) stands for the state usefulness which is evaluated from the state value $V^{(l)}$ and the visiting frequency $F^{(l)}$ of the lth memory item as follows:

$$\varepsilon^{(l)} = \lambda \frac{V^{(l)} - V_{\min}}{V_{\max} - V_{\min}} + (1 - \lambda) \frac{F^{(l)} - F_{\min}}{F_{\max} - F_{\min}} \tag{7}$$

where $V_{\max}$ and $F_{\max}$ are the maximum values of $V^{(l)}$ and $F^{(l)}$ among all memory items; $V_{\min}$ and $F_{\min}$ are the minimum values; λ is a constant between 0 and 1. As seen from Eqs. (6) and (7), if $\varepsilon^{(l)}$ for $\boldsymbol{x}^{(l)}$ is large (i.e., $\tau^{(l)}$ is small), an action at a state around $\boldsymbol{x}^{(l)}$ is deterministically selected.

In a grid-world problem, the states with large $\varepsilon^{(l)}$ are generally aligned along an optimal path to reach a goal. Since a sequence of deterministic actions works as a macro-action, we can say that AC-RAN creates and executes macro-actions in an indirect manner [4].

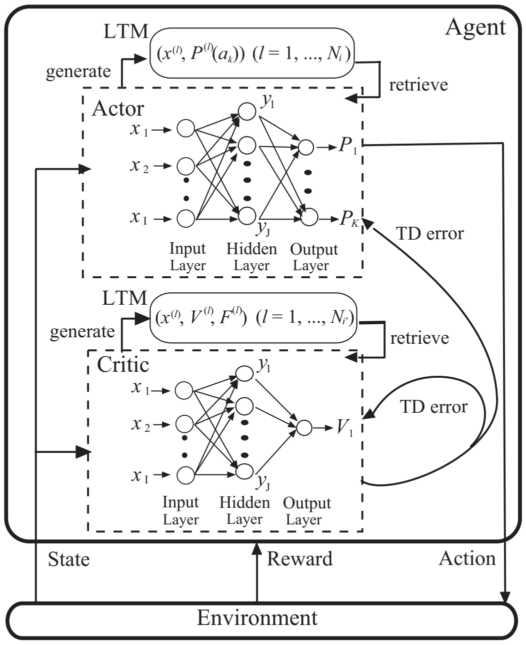


Fig. 1. Murata's Actor-Critic RAN-LTM (AC-RAN) Model.

III. Extended AC-RAN

Since macro-actions created in AC-RAN are available only for a specific maze task, they are not satisfied with the third property in Section I. Thus, we extend AC-RAN such that generated macro-actions could be utilized for the learning of different maze tasks.

A. Multi-Task Grid-World Maze Problem

Here, we consider a multi-task grid-world (MTGW) maze problem in which multiple maze problems are given sequentially to an agent. The purpose of each grid-world maze problem is to reach a goal as fast as possible. We assume that these mazes in an MTGW problem shares geometrical similarity in the obstacle locations. Therefore, if macro-actions are created in the regions with geometrical similarity called *correlated regions*, it is expected that the learning of the extended AC-RAN would be boosted in future tasks by utilizing the macro-actions.

We assume that an agent can observe two types of information: *location* $\boldsymbol{x} = \{x_1, x_2\}^T$ and *distances to the obstacles in eight directions* $\boldsymbol{s} = \{s_1, s_2, \cdots, s_8\}^T$ (see Fig. 2). The first information $\boldsymbol{x}$ is used for learning Actor and Critic RAN-LTMs as an input while the second information $\boldsymbol{s}$ is not used in the learning; instead, it is used for defining macro-actions.

Figure 2 illustrates the distance information in two correlated regions. Although the two distance vectors in Figs. 2 (a) and (b) are different, the distance information should be regarded as identical because the two vectors can be completely matched by rotation.

B. Definitions

Before describing the learning algorithm of the extended AC-RAN, let us define several terms, variables, and functions. An *episode* is defined as a period needed for an agent to move from start to goal, and let L_e be the length of an episode. The sequences of the distance information and actions in an episode are respectively defined as $\boldsymbol{s}(1:L_e)=\{\boldsymbol{s}(1),\cdots,\boldsymbol{s}(L_e)\}$ and $a(1:L_e)=\{a(1),\cdots,a(L_e)\}$. These two sequences can be written in a sequence of state-actions $\boldsymbol{Z}(1:L_e)=\{\boldsymbol{s}(1:L_e),a(1:L_e)\}$. Similarly, for a period $[t_1,t_2]$, the partial sequences of states, actions, and state-actions are denoted as $\boldsymbol{s}(t_1:t_2)=\{\boldsymbol{s}(t_1),\cdots,\boldsymbol{s}(t_2)\}$, $a(t_1:t_2)=\{a(t_1),\cdots,a(t_2)\}$, and $\boldsymbol{Z}(t_1:t_2)=\{\boldsymbol{s}(t_1:t_2),a(t_1:t_2)\}$, respectively.

A macro-action, a partial state-action sequence $\boldsymbol{Z}(t_1:t_2)$ satisfying all the macro-action properties mentioned in Section I, is denoted as $\boldsymbol{M}^i(1:L_m^i)=\{\boldsymbol{s}^i(1:L_m^i),a^i(1:L_m^i)\}$ where L_m^i is the length of the ith macro-action. On the other hand, a macro-action candidate satisfying the first two properties in Section I is denoted as $\hat{\boldsymbol{M}}^i(1:\hat{L}_m^i)=\{\hat{\boldsymbol{s}}^i(1:\hat{L}_m^i),\hat{a}^i(1:\hat{L}_m^i)\}$ where $\hat{L}_m^i$ is the length of the ith macro-action candidate.

To measure the similarity between two partial state-action sequences, let us define the following similarity function R:

$$\begin{aligned} R\;&(\boldsymbol{Z}(t_1:t_2),\mathrm{rot}(\hat{\boldsymbol{M}}^i(t_3:t_4),\theta)) \\ &= \frac{1}{L}\sum_{l=0}^{L-1}\delta(a(t_1+l),\mathrm{rot}(\hat{a}^i(t_3+l),\theta)) \\ &\quad\times\frac{\boldsymbol{s}(t_1+l)^T\mathrm{rot}(\hat{\boldsymbol{s}}^i(t_3+l),\theta)}{\|\boldsymbol{s}(t_1+l)\|\|\hat{\boldsymbol{s}}^i(t_3+l)\|} \end{aligned} \tag{8}$$

where t_1 and t_2 are the start and end time of the matched state-action sequence; t_3 and t_4 are the start and end time of the matched macro-action candidate; $L=\min\{t_2-t_1+1,t_4-t_3+1\}$ and $\delta(a_1,a_2)$ is a function which gives 1 for $a_1=a_2$ and 0 for $a_1\neq a_2$; $\mathrm{rot}(*,\theta)$ is the function that gives a rotation θ to a sequence.

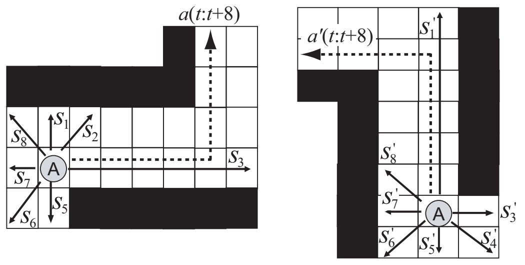


Fig. 2. An example of distance information and action sequences matched by rotation in two different mazes.

C. Learning Algorithm

The proposed learning algorithm mainly consists of the four procedures:

1) segmentation of state-action sequences,
2) matching state-action sequences,
3) creation of macro-actions,
4) selection and execution of macro-actions.

The details of the above procedures are described below.

1) Segmentation of State-action Sequences: Assume that Actor and Critic RAN-LTMs are trained over a sufficient number of episodes. After the learning, an episode is started and the agent's actions with the largest selection probability are selected deterministically (i.e., greedy action selection) until the agent reaches the goal. Let us define the deterministic state-action sequence by $\boldsymbol{Z}(1:L_e)=\{\boldsymbol{s}(1:L_e),a(1:L_e)\}$.

$\boldsymbol{Z}(1:L_e)$ is segmented into partial state-action sequences. The segmented points on $\boldsymbol{Z}(1:L_e)$ are determined based on the following $\Delta\boldsymbol{s}(t)$:

$$\Delta\boldsymbol{s}(t)=\|\boldsymbol{s}(t)-\boldsymbol{s}(t-1)\|. \tag{9}$$

$\Delta\boldsymbol{s}(t)$ is calculated at every state on $\boldsymbol{Z}(1:L_e)$; then, we search for the largest N_s-1 of $\Delta\boldsymbol{s}(t)$ points where N_s is the number of partial sequences to be segmented. Assume that the segmented points are found at time $t_1,\cdots,t_{N_s-1}$. Then, N_s partial state-action sequences $\boldsymbol{Z}(1:t_1)$, $\boldsymbol{Z}(t_1:t_2)$, $\cdots$, $\boldsymbol{Z}(t_{N_s-1}:L_e)$ are obtained and they are kept as macro-action candidates.

2) Matching Similar State-action Sequences: Assume that there are $\hat{N}_m$ macro-action candidates obtained in the past tasks. Let us denote them as $\hat{\boldsymbol{M}}^i(1:\hat{L}_m^i)=\{\hat{\boldsymbol{s}}^i(1:\hat{L}_m^i),\hat{a}^i(1:\hat{L}_m^i)\}$ $(i=1,\cdots,\hat{N}_m)$. To find macro-action satisfying the third property in Section I, we perform the matching between $\hat{\boldsymbol{M}}^i(1:\hat{L}_m^i)$ and the partial state-action sequences $\boldsymbol{Z}(t_{n-1}:t_n)$ $(n=1,\cdots,N_s)$ obtained in the current task. Let t'_n and t'_i be the matching points on $\boldsymbol{Z}(t_{n-1}:t_n)$ and $\hat{\boldsymbol{M}}^i(1:\hat{L}_m^i)$, respectively. And let l and θ be the length of a matched sequence and the rotation angle. Then, the best matching is given by finding the following values:

$$\begin{aligned} (l^*,t_n'^*,t_i'^*,\theta^*) \;&=\; \underset{l,t'_n,t'_i,\theta}{\operatorname{argmax}} R(\boldsymbol{Z}(t'_n:t'_n+l), \\ &\qquad \mathrm{rot}(\hat{\boldsymbol{M}}^i(t'_i:t'_i+l),\theta)). \end{aligned} \tag{10}$$

For $\boldsymbol{Z}(t_{n-1}:t_n)$ and $\hat{\boldsymbol{M}}^i(1:\hat{L}_m^i)$, the similarity $\mathcal{R}(n,i)$ is defined as follows:

$$\begin{aligned} \mathcal{R}(n,i) \;&=\; R(\boldsymbol{Z}(t_n'^*:t_n'^*+l^*), \\ &\qquad \mathrm{rot}(\hat{\boldsymbol{M}}^i(t_i'^*:t_i'^*+l^*),\theta^*)). \end{aligned} \tag{11}$$

3) Generating Macro-actions: Assume that several state-action sequences are matched and the similarity is measure by $\mathcal{R}(n,i)$ in Eq. (11). A natural thought is that a macro-action should be generated from the matched sequences with large similarity. Therefore, if the similarity of the nth partial state-action sequence $\boldsymbol{Z}(t_{n-1} : t_n)$ and the ith macro-action candidate $\hat{\boldsymbol{M}}^i(1 : \hat{L}^i_m)$ is larger than a threshold η, a macro-action is generated from the matched state-action sequences $\boldsymbol{Z}(t'^*_n : t'^*_n + l^*)$ and $\hat{\boldsymbol{M}}^i(t'^*_i : t'^*_i + l^*)$. However, the actions of $\boldsymbol{Z}(t'^*_n : t'^*_n + l^*)$ and $\hat{\boldsymbol{M}}^i(t'^*_i : t'^*_i + l^*)$ are not completely consistent with each other because these state-action sequences are obtained in different maze tasks. Here, let us adopt the actions in the matched macro-action candidate as the actions in a generated macro-action. Then, a macro-action is given by

$$\begin{aligned} \boldsymbol{M}^{N_m+1} \quad & (1 : l^*) = \{\boldsymbol{s}^{N_m+1}(1 : l^*), a^{N_m+1}(1 : l^*)\} \\ \overset{\text{def}}{=} \quad & \left\{ \frac{\boldsymbol{s}(t'^*_n : t'^*_n + l^*) + \text{rot}(\hat{\boldsymbol{s}}^i(t'^*_i : t'^*_i + l^*), \theta^*)}{2}, \right. \\ & \left. \text{rot}(\hat{a}^i(t'^*_i : t'^*_i + l^*), \theta^*) \right\}. \end{aligned} \tag{12}$$

After creating a macro-action, the matched sequences $\boldsymbol{Z}(t'^*_n : t'^*_n + l^*)$ and $\hat{\boldsymbol{M}}^i(t'^*_i : t'^*_i + l^*)$ are deleted and the other partial sequences. Since the creation of macro-actions is performed after the learning of the current task, these created macro-actions are utilized in the next task.

4) Selection and Execution of Macro-actions: When an action is selected at a state $\boldsymbol{s}(t)$, an agent check if there is a macro-action whose starting state is matched with the current state $\boldsymbol{s}(t)$ at time t; that is, the following condition is checked for every macro-action $\boldsymbol{M}^i(1 : L^i_m) = \{\boldsymbol{s}^i(1 : L^i_m), a^i(1 : L^i_m)\}$ $(i = 1, \cdots, N_m)$:

$$\boldsymbol{s}(t)^T \text{rot}(\boldsymbol{s}^{i^*}(1), \theta^*) \geq \eta \tag{13}$$

where

$$(i^*, \theta^*) = \underset{i,\theta}{\text{argmax}} \{\boldsymbol{s}(t)^T \text{rot}(\boldsymbol{s}^i(1), \theta)\}. \tag{14}$$

Here, θ is the rotation angle and η is a threshold.

If there exists a macro-action $\boldsymbol{M}^{i^*}(1 : L^{i^*}_m)$ satisfying Eq. (13), the action $\text{rot}(a^{i^*}(1), \theta^*)$ is selected at time t. The same procedure is repeated until the current state $\boldsymbol{s}(t + \tau)$ is not satisfied with the following condition:

$$\boldsymbol{s}(t + \tau)^T \text{rot}(\boldsymbol{s}^{i^*}(\tau + 1), \theta^*) \geq \eta \tag{15}$$

where $1 \leq \tau < L^{i^*}_m$.

While a macro-action is selected, Critic RAN-LTM is trained based on the TD error in Eq. (5). On the other hand, Actor RAN-LTM is not learned while executing the macro-action. The learning of Actor RAN-LTM is conducted based on the following accumulated rewards when a macro-action is terminated:

$$r_M(t + \tau') = r(t + 1) + \gamma r(t + 2) + \cdots + \gamma^{\tau' - 1} r(t + \tau') \tag{16}$$

where $1 \leq \tau' < L^{i^*}$. The following TD error is calculated and the Macro-Q learning [3] is applied to Actor RAN-LTM:

$$e_M(t + \tau') = r_M(t + \tau') + \gamma^{\tau'} V(\boldsymbol{x}(t + \tau')) - V(\boldsymbol{x}(t)). \tag{17}$$

The learning algorithm of AC-RAN is summarized in Algorithm 1.

Algorithm 1 Learning Algorithm of Extended AC-RAN

```
Input:
  • Set of actions $\mathcal{A} = \{a_1, \cdots, a_K\}$.
  • Sequence of maze tasks $\mathcal{T}^1, \mathcal{T}^2, \cdots$.
  • Actor and Critic RAN-LTM.
Initialize the task index: $j = 1$
loop
  Start the maze task $\mathcal{T}^j$.
  Initialize time: $t = 1$.
  Initialize location information: $\boldsymbol{x}(1) = \boldsymbol{x}_s$.
  repeat
    Observe the location information $\boldsymbol{x}(t)$ and the distance information $\boldsymbol{s}(t)$.
    Select a macro-action satisfying Eq. (13).
    if There exists a matched macro-action then
      Execute the macro-action.
      Calculate TD error by Eq. (17).
      Learn Actor RAN-LTM.
      $t \leftarrow t + \tau'$
    else
      Input $\boldsymbol{x}(t)$ to Actor and Critic RAN-LTM, and calculate the output $\boldsymbol{z}(\boldsymbol{x}(t))$ by Eq. (4).
      Set the action preference $P(\boldsymbol{x}(t), a_k)$ and the state value $V(\boldsymbol{x}(t))$ to the outputs $\boldsymbol{z}(\boldsymbol{x}(t))$ of Actor and Critic RAN-LTM, respectively.
      Retrieve the closest memory item in Critic RAN-LTM $(\boldsymbol{x}^{(l)}, \boldsymbol{z}^{(l)}, F^{(l)})$ to the input $\boldsymbol{x}(t)$.
      Increment the visiting frequency: $F^{(l)} \leftarrow F^{(l)} + 1$
      Update $\varepsilon^{(l)}$ based on Eq. (7).
      Calculate the temperature $\tau(t)$ based on Eq. (6).
      Select an action $a(t)$ according to the softmax strategy in Eq. (1).
      Observe the reward $r(t+1)$ and the next state $\boldsymbol{x}(t+1)$.
      Calculate TD error for Critic RAN-LTM by Eq. (5).
      Learn Actor and Critic RAN-LTM.
      $t \leftarrow t + 1$
    end if
  until $\boldsymbol{x}(t) = \boldsymbol{x}_g$
  Reset the agent's location to $\boldsymbol{x}_s$.
  Select actions based on the preference of Actor RAN-LTM until the agent reaches the goal, and obtain a sequence of state-actions $\boldsymbol{Z}(1 : L_e) = \{\boldsymbol{s}(1 : L_e), a(1 : L_e)\}$.
  Perform the segmentation of $\boldsymbol{Z}(1 : L_e)$ (see III-C1).
  Create macro-actions (see III-C3).
  $j \leftarrow j + 1$
end loop
Output: Actor and Critic RAN-LTM.
```

IV. Evaluation

A. Experimental Setup

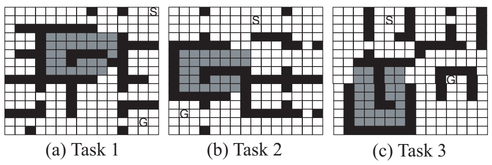
Figures 3 and 4 illustrate two multi-task Grid-World (MTGW) maze problems. An MTGW problem consists of six

15×15 grid-world mazes, each of which have one correlated region (shaded area). Here, ‘S’ and ‘G’ correspond to the start and the goal, respectively. We define five sequences of the six maze tasks for each MTGW problem:

Sequence 1: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$
Sequence 2: $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 6$
Sequence 3: $3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 6$
Sequence 4: $4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6$
Sequence 5: $5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6$

The last task (task 6) is a so-called *deceptive problem* in which an agent would be stacked at a dead end if a generated macro-action is used. In general, the execution of a macro-action corresponds to providing a bias in the action selection. This bias could lead to a positive effect to the learning speed; however, it is well known that such a bias could also lead to a negative effect (i.e., negative bias) [7]. Therefore, we estimate this negative effect by comparing the learning performance of Murata’s AC-RAN and the extended AC-RAN in task 6.

An agent learns each task with 400 episodes and a reward $+1$ is given to the agent only when it reaches the goal. If an agent cannot reach the goal within 1,000 steps, the episode is terminated and no reward is given. The number of steps to goal is averaged over 50 trials, and the average episodes to converge at a semi-optimal path are also investigated to evaluate the effectiveness of macro-actions.



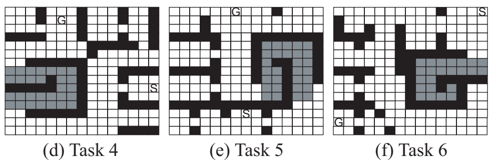


Fig. 3. MTGW Problem 1

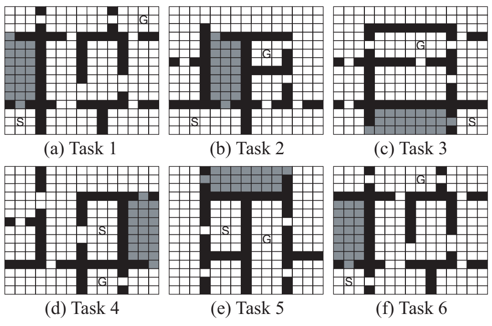


Fig. 4. MTGW Problem 2

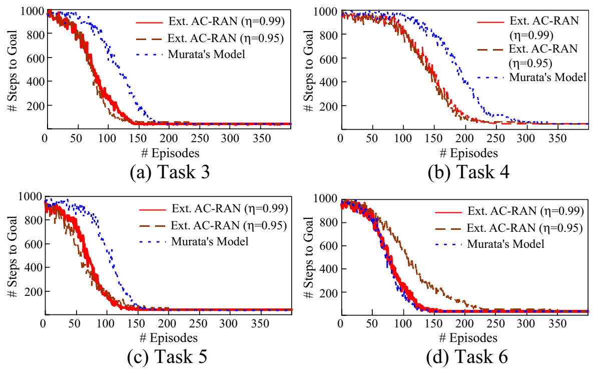


Fig. 5. Time evolutions of steps to reach the goal in MTGW problem 1.

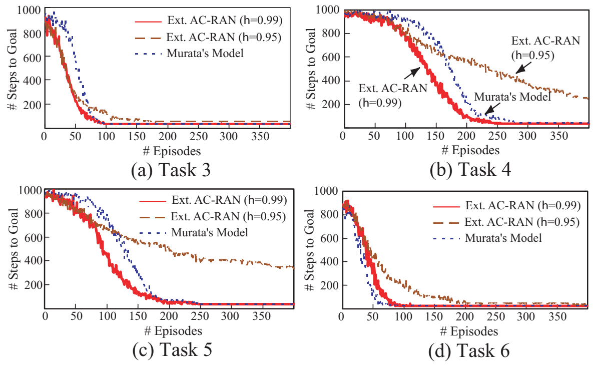


Fig. 6. Time evolutions of steps to reach the goal in MTGW problem 2.

B. Results

Figures 5 and 6 show the time evolutions of the average steps when the matching thresholds η in Eqs. (13) and (15) are set as follows: $\eta = 0.99$ and 0.95. Tables I (a)(b) show the average number of episodes needed for the convergence at a semi-optimal path. The reason why the results of the first two tasks are not shown is that macro-actions are created only after the learning of the second task. Here, the number of segmentation points N_s is set to 2 in all experiments.

As seen from Fig.5, for the MTGW problem 1, the extended AC-RAN can converge at a semi-optimal policy faster than the Murata’s AC-RAN. We also confirm this result in Table I (a). The results imply that macro-actions are generated in correlated regions properly and they are utilized for learning subsequent tasks. Therefore, we can say that macro-actions generated in the extended AC-RAN work as a *positive bias* that allows an agent to learn different tasks faster. Even for the deceptive task (task 6), the convergence property of the extended AC-RAN is almost the same as that of Murata’s AC-RAN when the threshold η is equal to 0.99. However, the convergence of AC-RAN becomes slow when $\eta = 0.95$. In general, if η is small, state-action sequences are easily matched with macro-action candidates; then, many redundant

TABLE I
THE NUMBER OF LEARNING EPISODES TO CONVERGE ON A SEMI-OPTIMAL PATH FROM START TO GOAL. "N/C" MEANS "NOT CONVERGED".

(a) MTGW Problem 1

Task	Ext. AC-RAN ($\eta = 0.99$)	Ext. AC-RAN ($\eta = 0.95$)	Murata's AC-RAN
3	143	219	191
4	246	280	318
5	156	203	171
6	166	341	148

(b) MTGW Problem 2

Task	Ext. AC-RAN ($\eta = 0.99$)	Ext. AC-RAN ($\eta = 0.95$)	Murata's AC-RAN
3	100	N/C	107
4	258	N/C	N/C
5	265	N/C	259
6	95	N/C	84

macro-actions can be created although each maze has only one correlated region. If these redundant macro-actions are frequently selected outside the correlated region, it is obvious that the learning convergence in AC-RAN would be degraded.

As seen from Fig. 6 and Table I (b), for the MTGW problem 2, the extended AC-RAN can converge on a semi-optimal path faster than Murata's AC-RAN when $\eta = 0.99$. However, the learning of AC-RAN fails to converge in all cases when $\eta = 0.95$. As seen from Fig. 4, the correlated regions in the MTGW problem 2 are similar to other sub-regions. Thus, it is considered that created macro-actions would be frequently executed outside the correlated region if η is small. In addition, many macro-actions tend to be created and this also leads to executing macro-actions outside the correlated region. From the above reasons, the convergence property of AC-RAN is seriously degraded when $\eta = 0.5$.

V. CONCLUSIONS

In this paper, we propose a reinforcement learning model called the extended Actor-Critic RAN-LTM (AC-RAN) which can autonomously generate and execute macro-actions in multi-task Grid-World (MTGW) maze problems. We extend Murata's AC-RAN such that created macro-actions can help an agent learn an optimal policy quickly.

In the extended AC-RAN, two types of state information are defined. The one is the agent location in a maze and it is used for learning which direction an agent should move next (i.e., action policy). The other state information is defined as a distance vector whose elements correspond to the distance to the obstacles in eight directions. This information is fairly independent among different tasks; therefore, it is used for generating and executing macro-actions. After learning Murata's AC-RAN, a sequence of state-action pairs is acquired by executing an episode based on the learned action policy. Then, several sub-sequences of state-action pairs are segmented from the sequence, and a rotation-invariant matching is performed between the segmented sub-sequence and macro-action candidates obtained in the previous tasks. If a sub-sequence is matched, it is created as a macro-action; then, it can be utilized for learning other tasks.

In the experiments, the extended AC-RAN is applied to two MTGW problems, each of which consists of six different maze tasks. From the experimental result, we conclude that the extended model can speed up learning if the matching threshold is properly set.

ACKNOWLEDGMENT

The authors would like to thank Professor Shigeo Abe for his helpful comments and discussions. This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C) 205002205.

REFERENCES

[1] G. A. Iba, "A Heuristic Approach to the Discovery of Macro-operators," *Machine Learning*, Vol.3, pp. 285-317 (1989)
[2] A. McGovern and A. G. Barto, "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density," *Proc. Int. Conf. on Machine Learning*, pp. 361-368 (2001)
[3] A. McGovern and R. S. Sutton, "Roles of Macro-Actions in Accelerating Reinforcement Learning: An Empirical Analysis," *Univ. Massachusetts*, UM-CS-1998-070, pp. 70-98 (1998)
[4] M. Murata and S. Ozawa, "A Memory-Based Reinforcement Learning Model Utilizing Macro-Actions," *Proc. Int. Conf. on Adaptive & Natural Comp. Algorithms*, pp. 78-81 (2005)
[5] G. Konidaris and A. G. Barto, "Building Portable Options: Skill Transfer in Reinforcement Learning," *Proc. Int. J. Conf. on Artificial Intelligence*, pp. 895-900 (2007)
[6] N. Shiraga, S. Ozawa, and S. Abe, "A Reinforcement Learning Algorithm for Neural Networks with Incremental Learning Ability," *Proc. Int. Conf. on Neural Information Processing 2002*, Vol. 5, pp. 2566-2570 (2002)
[7] S. Thrun and L. Pratt, *Learning to learn*, Kluwer Academic Pub. (1998)