# A Memory-Based Reinforcement Learning Model Utilizing Macro-Actions

Makoto Murata and Seiichi Ozawa
Graduate School of Science and Technology, Kobe University, Japan
E-mail: ozawasei@kobe-u.ac.jp

**Abstract**

One of the difficulties in reinforcement learning (RL) is that an optimal policy is acquired through enormous trials. As a solution to reduce waste explorations in learning, recently the exploitation of macro-actions has been focused. In this paper, we propose a memory-based reinforcement learning model in which macro-actions are generated and exploited effectively. Through the experiments for two standard tasks, we confirmed that our proposed method could decrease waste explorations especially in the early training stage. This property contributes to enhancing training efficiency in RL tasks.

## 1 Introduction

Recently, in order to avoid waste explorations in reinforcement learning (RL) tasks, several researchers have studied learning algorithms that allow agents to find a series of useful actions called *macro-actions* [1, 2]. For example, when we stand in front of a door we have never seen, first we recognize the knob, reach out our hand for it, and then try to push and pull the door to open. We often attempt to apply some acquired macro-actions to a new task, instead of searching for all combinations of possible actions. Clearly, even when we search for appropriate actions, we exploit some knowledge on actions learned from the past experiences. This fact indicates that we should introduce a mechanism of generating and exploiting macro-actions into reinforcement learning.

On the other hand, we have proposed an RL agent model based on function approximation, in which a value function is approximated by Resource Allocating Network with Long-Term Memory (RAN-LTM) [4]. In RAN-LTM, several representative input-output pairs (called *memory items*) on the approximated value function are stored in long-term memory. Although this learning scheme can be categorized into memory-based learning, the differences from the conventional approaches lie in the number of memory items and their embedded information. In RAN-LTM, memory items are restricted to the minimum number that can suppress the interference caused by incremental learning, and the memory items can store not only the information on an approximated value function but also some other information (e.g., complexity of approximated functions).

In this paper, we propose a new version of RAN-LTM which has a mechanism of generating and exploiting macro-actions to enhance the learning efficiency in RL tasks. To realize this mechanism, new information on the usefulness of state-action pairs as a component of a macro-action is added to the memory items. This information is accumulated in all of the retrieved memory items during an episode. The degree of usefulness is decided depending on how much a series of retrieved memory items contribute to receiving high reward; if this usefulness is high, the corresponding state-action is regarded as a component of a macro-action. Thus, this information is exploited to control the randomness of action selection; that is, when a memory item with high usefulness is retrieved, the probability function of action selection is changed such that a greedy action is taken.

This paper is organized as follows. In Section 2, we present an Actor-Critic model in which function approximators are implemented by RAN-LTM. Section 3 describes how to introduce a mechanism of generating and exploiting macro-actions into RAN-LTM. In Section 4, several experiments are carried out in the two standard tasks. Finally, we state conclusions in Section 5.

## 2 Actor-Critic Model Using RAN-LTM

Actor-Critic model is one of the most frequently used RL models [3]. In Actor, an action is selected based on a preference value $P_k(\boldsymbol{x}(t), a(t))$, while Critic is used to estimate a state value $V(\boldsymbol{x}(t))$. After selecting an action, Critic evaluates a new state $\boldsymbol{x}(t+1)$ based on TD error $e(t)$ shown below:

$$e(t) \leftarrow r(t+1) + \gamma V(\boldsymbol{x}(t+1)) - V(\boldsymbol{x}(t)) \qquad (1)$$

where $\gamma$ is a discount factor. The preference value for a selected action is updated as follows:

$$P_{k'}(\boldsymbol{x}(t), a(t)) \leftarrow P_{k'}(\boldsymbol{x}(t), a(t)) + \beta e(t) \qquad (2)$$

where $k'$ is the index of the action selected in Actor and $\beta$ is a positive constant.

To approximate a value function $V(\boldsymbol{x}(t))$ and a preference $P_k(\boldsymbol{x}(t), a(t))$, RAN-LTM is adopted as a function

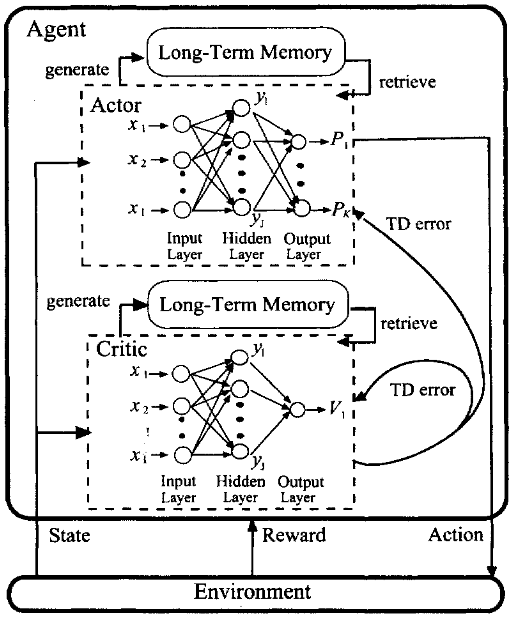

**Fig. 1.** Structure of Actor-Critic model using RAN-LTM.

approximator in both Actor and Critic. The structure of the Actor-Critic model is shown in Fig. 1. The outputs of the two RAN-LTMs correspond to $V(\boldsymbol{x}(t))$ in Critic and $P_k(\boldsymbol{x}(t), a(t))$ in Actor.

The inputs of RAN-LTM in Actor and Critic at time $t$ is denoted as $\boldsymbol{x}(t) = \{x_1(t), \cdots, x_I(t)\}'$ and they correspond to the agent's states $\boldsymbol{s}(t) = \{s_1(t), \cdots, s_I(t)\}'$. Here $I$ is the number of input units (states). As shown in Eq. (3), the hidden outputs $\hat{\boldsymbol{y}}(t) = \{\hat{y}_1(t), \cdots, \hat{y}_J(t)\}'$ are given by

$$\hat{y}_j(t) = \exp\left(-\frac{\| \boldsymbol{x}(t) - \boldsymbol{c}_j \|^2}{{\sigma_j}^2}\right) \quad (j = 1, \cdots, J) \qquad (3)$$

where $\boldsymbol{c}_j$ and $\sigma_j^2$ are a center vector and a width parameter of the $j$th radial basis function; $J$ is the number of hidden units. The hidden outputs $\hat{\boldsymbol{y}}(t) = \{\hat{y}_1(t), \cdots, \hat{y}_J(t)\}'$ are normalized by the sum of hidden outputs. Then, the network outputs $\boldsymbol{z}(t)$ are obtained from

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

where $y_j$, $w_{kj}$ and $\gamma_k$ are a normalized hidden output, a connection weight from the $j$th hidden unit to the $k$th output unit and a bias of the $k$th output unit, respectively; $K$ is the number of output units. As stated before, the outputs of Critic correspond to state values $V(\boldsymbol{x}(t))$, while the outputs of Actor correspond to the preference of actions $P_k(\boldsymbol{x}(t), a(t))$. In the softmax strategy, the agent actions are selected based on the following probability:

$$Pr\{a(t) = a\} = \frac{e^{P(\boldsymbol{x}(t), a)/\tau(t)}}{\sum_{b=1}^{K} e^{P(\boldsymbol{x}(t), a_b)/\tau(t)}} \qquad (5)$$

where $\tau(t)$ is a temperature to control the randomness of agent actions. Note that $K$ is equal to the number of actions to be selected by agents.

In reinforcement learning, a training sample is given incrementally. In such a situation, neural networks can easily forget the knowledge acquired before. To suppress unlearning, we utilize memory items that are extracted from the approximated function. When learning a new training sample, some memory items are simultaneously trained with the new sample in RAN-LTM to keep input-output relations as much as possible. The readers who want to know the detail training procedures in RAN-LTM can refer to [4].

## 3 Generating and Exploiting Macro-Actions

In this section, we introduce a mechanism of generating and exploiting macro-actions into Actor-Critic model described in the previous section. As stated in Section 1, we define macro-actions as a deterministic series of useful actions that lead to large rewards with high possibility. To generate such macro-actions, an agent has to retain all the state-action pairs leading to high rewards in some way. However, if the agent's states are continuous, it is obvious that all the possible pairs cannot be held in memory. Considering that memory items in RAN-LTM are distributed over the state space, we easily come upon to add extra information on successful experiences to these memory items.

Let us introduce a new index $L_m$ in the $m$th memory item in Actor. When an agent receives a reward $r_T$ at time $T$, $L_m$ of all memory items retrieved during the current episode are updated as follows:

$$L_m^{NEW} \leftarrow \eta L_m^{OLD} + \eta^{T - t_m - 1}(r_T - \bar{r}) \qquad (6)$$

where $\bar{r}$, $t_m$ and $\eta$ are the average reward, the time to recall the $m$th memory item, and a decay constant, respectively. This update is based on the profit sharing method. Note that $L_m$ of the $m$th memory item becomes large when an action taken in the corresponding state leads to a high reward to the end and an agent frequently encounters the state. This suggests that a state with large $L_m$ should be considered as the component of a macro-action. On the other hand, if an agent encounters a state with large $L_m$, the action with the highest preference should be taken deterministically.

Based on this idea, we propose the following algorithm of selecting agent's actions which gives a mechanism of generating and exploiting macro-actions for the agent model shown in Fig. 1. Let us denote this agent model RAN-LTM-MA. Note that RAN-LTM-MA does not generate and exploit macro-actions explicitly; macro-actions are generated as a result of controlling the temperature $\tau(t)$ in the action selector that is determined by $L_m$ of the nearest memory item.

**[Algorithm]**

1. Recall a memory item $\tilde{M}_m : (\tilde{x}_m, L_m, t_m)$ whose $\tilde{x}_m$ has the nearest distance to the state $x_t$.
2. If the distance between $x_t$ and $\tilde{x}_m$ is smaller than a threshold, set $t_m = t$.
3. Update $\tau(t)$ as follows: $\tau(t) \leftarrow \tau(t)/L_m$. Select action $a_{t+1}$ according to the probability in Eq. (5). If the agent receives no reward, go back to Step 1. Otherwise, go to Step 4.
4. Update $L_m$ for all retrieved memory items $\tilde{M}_m$ based on Eq. (6).
5. Reset $t$, then go back to Step 1.

## 4 Experiments

To see the usefulness of introducing macro-actions in RAN-LTM, RAN-LTM-MA is applied to the following two standard problems: Mountain-Car Task and Grid-World Task [3].

### 4.1 Mountain-Car Task

Mountain-Car Task is a task in which a car driver (agent) learns an efficient policy to reach a goal located on a hill. In this task, the goal of a car agent is to successfully drive up a steep slope and to reach the goal as soon as possible. There are three actions to be selected: *full throttle to the goal*, *zero throttle*, *full throttle in the opposite direction*. These actions are represented by the following values: $a(t) = \langle +1, 0, -1 \rangle$. A car agent is positioned at random when starting an episode. The position $u(t)$ and velocity $\dot{u}(t)$ are updated based on the following equations:

$$\begin{aligned} u(t+1) &= B[u + \dot{u}(t)] \\ \dot{u}(t+1) &= B[\dot{u}(t) + 0.07a(t) - 0.0025\cos(3u(t))] \end{aligned}$$

where $B$ is a function to restrict the working area, $\{u \mid -1.2 \le u \le 0.5\}$, $\{\dot{u} \mid -0.07 \le \dot{u} \le 0.07\}$. $u(t)$ and $\dot{u}(t)$ are given as inputs of Actor and Critic, and the value of -1 is given at every step as rewards to all actions taken.

The training is carried out through 200 episodes, which are divided into the 4 learning stages: 1-50, 51-100, 101-150 and 151-200. Since we expect that the introduction of macro-actions contribute to enhancing the training speed, the average steps needed to reach the goal is examined at the above 4 stages. For comparative purposes, the average steps are evaluated for the

**Table 1.** The average steps at the 4 learning stages.

| Episodes | RAN | RAN-LTM | RAN-LTM-MA |
|---|---|---|---|
| 1-50 | 980 | 814 | 490 |
| 51-100 | 586 | 451 | 206 |
| 101-150 | 480 | 324 | 210 |
| 151-200 | 327 | 245 | 258 |

Actor-Critic models in which Resource Allocating Network (RAN) [5] and RAN-LTM are adopted as function approximators. The results are shown in Table 1.

As seen from Table 1, at the early stage (1-100 episodes), the average steps in RAN-LTM-MA are greatly reduced as compared with those in the other two models. This result suggests that RAN-LTM-MA can find a proper policy with less experiences by generating and exploiting macro-actions. However, at the last stage (151-200 episodes), the differences in the performances of RAN-LTM and RAN-LTM-MA are not clear. This shows that there are no significant differences in the accuracy as a function approximator; thus one can say that introducing macro-actions contribute to avoiding waste exploration in the early stage of learning.

### 4.2 Grid-World Task

In this task, the purpose of an agent is to find the shortest path from a start point to a goal in a $10 \times 10$ grid-world. Here we adopt 6 grid-world tasks as shown in Fig. 2. In these tasks, an agent can move in 4 directions (east, south, west, north). At the beginning of an episode, an agent is placed at each start point. The reward (+1) is only given to the agent when reaching the corresponding goal.

The training is carried out through 100 episodes, and in each episode the agent's policy is evaluated by the number of steps to the goal. To remove the dependency on random seeds, the training is repeated 30 times and the average steps are evaluated.

Figure 3 shows the time courses of the average steps taken by an agent and the action probability in Task 1. This action probability is obtained by averaging the probability of selecting optimal actions on the three typical shortest paths. If this value is high, one can say that the agent takes optimal actions deterministically. As seen from Fig. 3, the average steps of RAN-LTM-MA rapidly decrease from the 20th episode; at the same time, the action probability rapidly increases. This result suggests that some useful macro-actions are generated on the optimal paths in RAN-LTM-MA and the exploitation of these macro-actions leads the decrease of the average steps. To verify this, we examine the actions selected with 99% and 90% probability. The result is shown in Fig. 4 where black and white arrows correspond to the

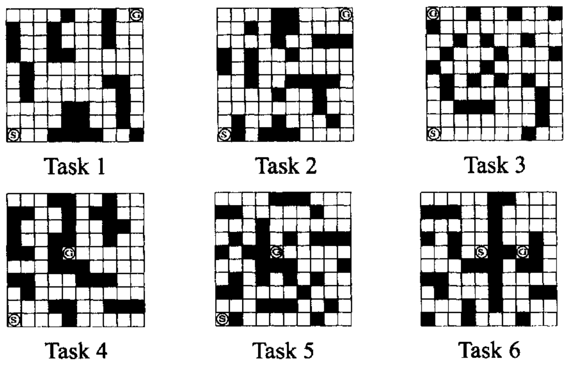

**Fig. 2.** Evaluated grid-world tasks.

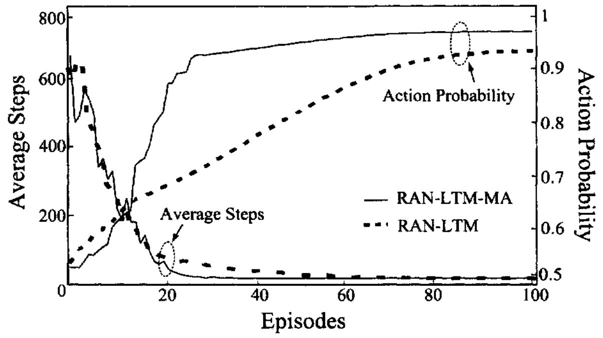

**Fig. 3.** Time courses of average steps and action probability in Task 1.

directions of selected actions with 99% and 90%, respectively. In Fig. 4, we can see that some series of deterministic actions are generated in RAN-LTM-MA after the 20th episode.

Table 2 shows the number of needed episodes to get a proper policy in the 6 grid-world tasks. Here we define that an agent acquire a proper policy when the steps to a goal is less than the following value: (*steps of the shortest path*) × 1.2. From the results in Table 2, we can conclude that RAN-LTM-MA can get a proper policy promptly in all tasks.

## 5 Conclusions

In this paper, we proposed a new agent model in which macro-actions were generated and exploited effectively. Through the experiments, we verified that our proposed model could decrease waste explorations especially at the early stage of training. Further experiments are needed to ensure whether such macro-actions really contributes to enhancing the training efficiency in any RL tasks.

**Acknowledgment**

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C) 16500130, and by the Okawa Foundation for Information and Telecommunications.

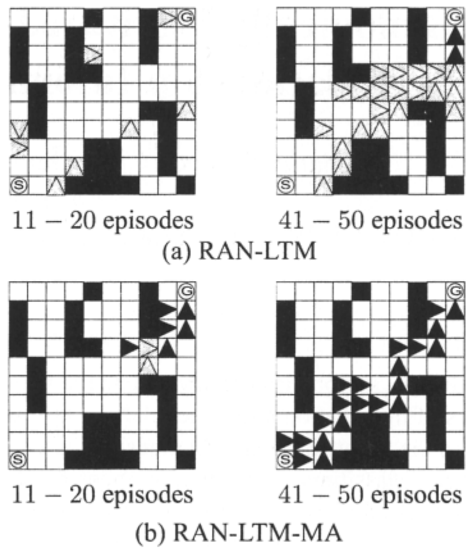

**Fig. 4.** Selected actions with 99% and 90% in Task 1. Black and white arrows correspond to the directions of selected actions with 99% and 90%, respectively.

**Table 2.** The number of needed episodes to get a proper policy in the 6 grid-world tasks.

| Task | RAN | RAN-LTM | RAN-LTM-MA |
|---|---|---|---|
| 1 | 64 | 64 | 33 |
| 2 | 66 | 65 | 33 |
| 3 | 70 | 70 | 41 |
| 4 | 59 | 59 | 26 |
| 5 | 54 | 54 | 27 |
| 6 | 86 | 86 | 39 |

## References

[1] Mcgovern, A., Sutton, R. S., Fagg, A. H. (1997) Roles of macro-actions in accelerating reinforcement learning. Proceedings of the 1997 Grace Celebration of Women in Computing: 13-18

[2] Precup, D., Sutton, R. S. and Singh, S. P. (1997) Planning with closed-loop macro actions. In Working Notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems: 70-76

[3] Sutton, R. S. and Barto, A. G. (1998) Reinforcement learning - An introduction. The MIT Press

[4] Ozawa, S. and Abe, S. (2004) A memory-based reinforcement learning algorithm to prevent unlearning in neural networks. In Neural Information Processing: Research and Development: Rajapakse, J. C. and Wang, L., Eds., Springer-Verlag: 238-255

[5] Platt, J. (1991) A resource allocating network for function interpolation. Neural Computations, 3: 213-225