A REINFORCEMENT LEARNING ALGORITHM FOR NEURAL NETWORKS WITH INCREMENTAL LEARNING ABILITY

Naoto Shiraga, Seiichi Ozawa, and Shigeo Abe

Graduate School of Science and Technology, Kobe University, Kobe, Japan
E-mail: shiraga@chevrolet.eedept.kobe-u.ac.jp, {ozawa,abe}@eedept.kobe-u.ac.jp

ABSTRACT

When neural networks are used for approximating action-values of Reinforcement Learning (RL) agents, the "interference" caused by incremental learning can be serious. To solve this problem, in this paper, a neural network model with incremental learning ability was applied to RL problems. In this model, correctly acquired input-output relations are stored into long-term memory, and the memorized data are effectively recalled in order to suppress the interference. In order to evaluate the incremental learning ability, the proposed model was applied to two problems: Extended Random-Walk Task and Extended Mountain-Car Task. In these tasks, the working space of agents is extended as the learning proceeds. In the simulations, we certified that the proposed model could acquire proper action-values as compared with the following three approaches to the approximation of action-value functions: tile coding, a conventional neural network model and the previously proposed neural network model.

1. INTRODUCTION

In Reinforcement Learning (RL) tasks, an agent is learning what to do through some experiences involving trial and errors[1]. RL agents adapt itself to the environment so as to maximize the total amount of rewards (positive returns) it receives over long run. To maximize the returns, RL agents estimates action-value functions, which are defined for the relations between state-action pairs and their estimates of returns that the agents will earn in the future.

In a simple RL task with several states and actions, it is possible to memorize all action-values into a look-up table. In many cases of practical interest, however, there are far more states than could possibly be entries in such a table. In this case, it is effective to acquire action-value functions using several function approximation methods. One of these is linear method[1], in which action-values are approximated by linear functions of feature vectors that roughly code agent's states. In general, this method tends to need large memories especially when the state space has large dimensions and/or wide area. To solve this problem, it is useful to use continuous approximation functions like neural networks[1]. However it is well known that the learning of neural networks becomes difficult when the distribution of given training data is temporally varied and training data are incrementally given[2]. In such a situation, input-output relations acquired in the past are easy to be collapsed by the learning of new training data.

This disruption in neural networks is called "interference" that is caused by the excessive adaptation of connection weights for new training data. The interference can also arise in RL problems because the rewards for agent actions are incrementally given by the environments. In this context, it is important to learn action-value functions such that the interference is suppressed as much as possible.

There have been proposed several approaches to suppressing the interference in supervised learning[2, 3, 4, 5]. We have proposed an incremental learning model for supervised learning tasks[2, 5], in which Long-Term Memory (LTM) was introduced into Resource Allocating Network (RAN)[6]. For the notational convenience, this model is denoted as RAN-LTM in the followings.

In RAN-LTM, storage data in LTM (noted as "LTM data") are produced from inputs and outputs of networks whose relationships are accurately approximated. In supervised learning, the exact errors between network outputs and their target values are given. In RL problem, however, the agents is generally given only Temporal Difference (TD) errors calculated from currently estimated action-values and immediate rewards instead of exact errors[7]. Therefore, LTM data generated in the past do not always hold proper action-values. In this context, the procedures of updating LTM data should be introduced into RAN-LTM. In this paper, we propose a new version of RAN-LTM whose LTM data are updated.

2. APPROXIMATION OF ACTION-VALUE FUNCTION USING NEURAL NETWORKS

Figure 1 shows the architecture of RAN-LTM that consists of two modules: Resource Allocating Network (RAN)[6]

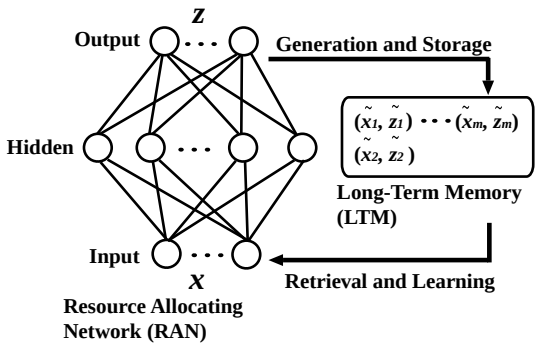

Figure 1: RAN-LTM model.

and Long Term Memory (LTM). To approximate action-value functions, RAN with normalized radial basis functions is adopted here, in which the outputs of hidden units are normalized. When the training gets started in RAN, the number of hidden units is set to one initially; hence, RAN possesses simple approximation ability at first. As the trials proceed, the approximation ability of RAN is developed by allocating additional hidden units.

The inputs of RAN at time t, $\boldsymbol{x}(t) = \{x_1(t), \cdots, x_I(t)\}'$, are the agent's states $\boldsymbol{s}(t) = \{s_1(t), \cdots, s_I(t)\}'$. Here, I is the number of input units of RAN. We associate network outputs $\boldsymbol{z}(t)=\{z_1(t), \cdots, z_K(t)\}'$ with action-values $Q_t(\boldsymbol{x}(t), a_k)$ that are utilized for selecting agent's action $a(t)$; i.e. $z_k(t)=Q_t(\boldsymbol{x}(t), a_k)$. Hence, the number of output units, K, is equal to the number of agent's actions, A. The outputs $\boldsymbol{z}(t)$ are given as follows:

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

$$y_j(t) = \frac{y'_j(t)}{\sum_{k=1}^{J} y'_k(t)} \quad (j = 1, \cdots, J) \quad (2)$$

$$y'_k(t) = \exp(-\frac{\| \boldsymbol{x}(t) - \boldsymbol{c}_j \|^2}{{\sigma_j}^2}). \quad (3)$$

Here, w_{kj} and γ_k are a connection weight from the jth hidden unit to the kth output unit and a bias of the kth output unit, respectively. $y_j(t)$, J and σ_j^2 are respectively the jth output of hidden unit, the number of hidden units and the variance of the jth radial basis function. As shown in Eq.(2), the outputs of hidden units are normalized by the sum of all hidden outputs. The agent's action is selected based on the outputs of RAN. The TD error e_k for the kth output is defined as follows:

$$e_k(t) = \begin{cases} r(t) + \gamma \max_a Q_t(\boldsymbol{x}(t+1), a) \\ \qquad -Q_t(\boldsymbol{x}(t), a_k)) & (k = k') \\ 0 & (\text{otherwise}). \end{cases} \quad (4)$$

Here, k' is the index of action $a(t)$ selected by agents at time t, and $r(t)$ is an immediate reward given by the environment. Based on the TD error $e_{k'}(t)$ and the hidden outputs, the following learning procedures are conducted:

(1) If $e_{k'}(t) > \varepsilon$ and $\|\boldsymbol{x}(t) - \boldsymbol{c}^*\| > \delta(t)$, a hidden unit is added to RAN (i.e. $J \leftarrow J + 1$). Here, $\boldsymbol{c}^*$ is the nearest center vector to $\boldsymbol{x}(t)$. Then, the network parameters for the added hidden unit are respectively set to the following values:

$$c_{Ji} = x_i(t) \quad (i = 1, \cdots, I) \quad (5)$$

$$w_{kJ} = E_k(t) \quad (k = 1, \cdots, K) \quad (6)$$

$$\sigma_J = \kappa \|\boldsymbol{x}(t) - \boldsymbol{c}^*\|, \quad (7)$$

where κ is a positive constant. $\delta(t)$ decreases with time t as follows:

$$\delta(t) = \max[\delta_{max} \exp(\frac{t}{\tau}), \delta_{min}] > 0. \quad (8)$$

Here τ is a decay constant.

(2) Otherwise, in order to reduce TD errors, the network parameters are modified as follows:

$$w_{kj}^{NEW} = w_{kj}^{OLD} + \alpha e_k(t) y_j \quad (9)$$

$$c_{ji}^{NEW} = c_{ji}^{OLD} + 2\frac{\alpha}{\sigma_j}(x_i(t) - c_{ji}) y_j \sum_k e_k(t) w_{kj} \quad (10)$$

$$\gamma_k^{NEW} = \gamma_k^{OLD} + \alpha e_k(t), \quad (11)$$

where α is a positive learning ratio.

3. INCREMENTAL LEARNING ALGORITHM

In general, normalized hidden outputs of RAN do not have large values in the regions where hidden units are densely allocated[8]. Therefore, this type of RAN can suppress the interference itself to some extent. To suppress the interference more completely, neural network models with incremental learning ability like RAN-LTM should be adopted. However, in RL problems, only TD errors, which do not give accurate errors for true action-values, are given by the environments. Hence, LTM data acquired formerly in RAN-LTM do not always hold proper action-values of agents. In this context, it is considered that the update of LTM data

1) Learning episodes are repeated P_L times.
2) Evaluation episodes are repeated P_E times.
3) When the average steps in the evaluation episodes decrease to a certain value, LTM data are produced and/or updated based on the procedures shown in Fig.5.
4) If the number of repetition is equal to N_e, the learning is terminated. Otherwise, go back to 1.

Figure 2: Learning algorithm of RAN-LTM.

1) Set an agent's initial state $\boldsymbol{s}(0)$ to inputs $\boldsymbol{x}(0)$.
2) Based on Eq. (1), all action-values $\boldsymbol{Q}_t(\boldsymbol{x}(t), a_k)$ are calculated, and then a proper action a_k is selected based on the action-values.
3) The RAN-LTM agent takes an action a_k, and then the agent's state changes to $s(t+1)$. A reward $r(t)$ is given to the agent. .
4) TD errors are calculated based on Eq. (4). Depending on hidden outputs, some LTM data are retrieved based on the procedures shown in Fig.4. Then, the errors between outputs and retrieved LTM data are calculated.
5) Based on the calculated errors, the center vectors, connection weights, and biases of RAN are modified.
6) If $s(t+1)$ is a terminal state, then the learning episode is over. Otherwise, $t \leftarrow t+1$ and go to Step 2.

Figure 3: Procedures of learning episode.

1) Obtain all indices j of hidden units whose outputs y_j are larger than θ_r, and define a set of these indices as $\mathcal{I}_2$.
2) If $\mathcal{I}_2 \neq \phi$, then go to 3). Otherwise, no LTM datum is retrieved and the procedure is terminated.
3) For all hidden units belonging to $\mathcal{I}_2$, obtain LTM datum $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{z}}_j)$ that has the nearest distance to center vectors $\boldsymbol{c}_j$.
4) All LTM data that satisfy the condition $\|c_j - \tilde{\boldsymbol{x}}_j\| < \xi$ are retrieved.

Figure 4: Procedures for retrieving LTM data.

should be conducted as the learning of RL agents proceeds. To determine when they should be updated, the appropriateness of agent's actions is estimated through some evaluation episodes stated below.

The procedures of producing, updating and retrieving LTM data are described in Figs. 2-5. As seen in Fig. 2, the proposed algorithm has two types of episodes: learning episode and evaluation episode. In the learning episodes, a RL agent learns its action-value function from experiences based on the procedures stated in Sect. 2. In the evaluation episodes, the learning of agents are stopped, and then the appropriateness of agent's actions is evaluated based on the preliminary defined criterion (e.g. the number of steps to reach the goal). These two types of episodes are done by turns and repeated N_e times.

4. SIMULATION EXPERIMENTS

To examine the performance of the proposed RL agents, we apply it to two problems: *Extended Random-Walk Task* and *Extended Mountain-Car Task*. These problems are extended from the original problems[1] in order to evaluate the incremental learning ability. For comparative purposes, we adopt Tile Coding (TC)[1], RAN and RAN-LTM to approximate action-value functions. TC is one of the linear methods in which action-values are approximated by linear functions of feature vectors that roughly code agent's states. In practical

1) If all outputs of hidden units y_j $(j = 1, \cdots, J)$ for an input $\boldsymbol{x}(t)$ are less than a threshold value θ_c, this procedure is terminated. Otherwise, go to Step 2).
2) Obtain all indices j of hidden units whose outputs y_j are larger than θ_c, and define a set $\mathcal{I}_1$ of these indices. The value r_j is updated for all $j \in \mathcal{I}_1$ as follows:

$$r_j \leftarrow r_j + 2\exp(-\rho E(t)) - 1.$$

Here, $E(t)$ is TD error and ρ is a positive constant.
3) If $r_j > \beta$ for $j \in \mathcal{I}_1$, then go to Step 4). Otherwise, this procedure is terminated. Here, β is a positive constant.
4) Initialize r_j. If no LTM datum for the jth hidden unit has been produced yet, then go to Step 5). Otherwise, go to Step 6).
5) The jth center vector $\boldsymbol{c}_j$ is given to RAN as its input $\tilde{\boldsymbol{x}}_M$, and the output $\tilde{\boldsymbol{z}}_M$ is calculated. $(\tilde{\boldsymbol{x}}_M, \tilde{\boldsymbol{z}}_M)$ is stored into LTM as the Mth LTM datum. The number of LTM data M increases by one (i.e. $M \leftarrow M+1$), and then the procedure is terminated.
6) $\tilde{\boldsymbol{x}}_j$ is given to RAN as its input, and the output $\tilde{\boldsymbol{z}}_j$ is obtained. If $\tilde{\boldsymbol{z}}_j$ largely differs from the previously stored LTM data, the nearest LTM datum is updated; that is, the LTM is replaced with $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{z}}_j)$.

Figure 5: Procedures for producing and updating LTM data.

problems, enormous number of receptive fields (tiles) for features could be generated for accurate approximation of action-value functions. Moreover, we evaluate two types of RAN-LTMs: one is the model that cannot update LTM data (noted as RAN-LTM1) and the other is the model that can update them (noted as RAN-LTM2).

The objective of RL agents in the above two problems is to reach a goal as soon as possible. Therefore, we can use the average number of steps to reach a goal as a criterion for the appropriateness of agent's actions. If the average number of steps decreases to a certain value, one can say that RAN has more accurate approximation for action-value function; hence the procedures of producing and updating LTM data should be carried out in this case.

4.1. Extended Random-Walk Task

In the original Random-Walk Task, there are five states 1-5 and each episode starts in the state 3 (see Fig.6). The agent can move either left or right at each step with equal probability. On the other hand, in the Extended Random-Walk Task, five more states 6-10 are added to the original task. The learning of these states is conducted after the learning of states 1-5.

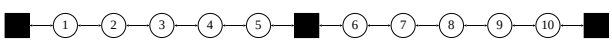

Figure 6: Extended Random-Walk Task.

Table 1: Theoretical state-values in Random-Walk Task.

region 1	1	2	3	4	5
region 2	10	9	8	7	6
state-value	1/6	2/6	3/6	4/6	5/6

In Fig.6, terminal states are denoted by black squares. The agent should learn action-value functions to select right actions for all state s. In the Extended Random-Walk Task, each episode starts from the state '3' or '8'. When an episode is terminated at the central state '5', a reward $(+1)$ is given to the agent; otherwise, the rewards are zero. There are two actions taken by an agent; "move inside" and "move outside". These actions are represented by the following two values: $+1$ or -1. For the notational convenience, the left hand side of the region including states $1 \sim 5$ is called 'region 1', the right hand side of the region including states $6 \sim 10$ is called 'region 2'. The number of input units I is 1, and state number s is set to the input of a neural network.

In the Random-Walk Task, we can calculate the true action-values $\boldsymbol{Q}(\boldsymbol{s}, a)$ from the theoretical state-values as seen in Table 1. Therefore, we can evaluate the true errors between estimated action-values $\boldsymbol{z}$ and theoretical ones.

4.1.1. Results and Discussions

In the learning episodes, an agent selects an action with equal probability because RAN and RAN-LTM agents approximate action-value function. In the evaluation episodes, the action of an agent is determined based on $\boldsymbol{Q}(\boldsymbol{x}(t), a_k)$. The numbers of the learning and evaluation episodes are respectively set to the following values: $P_L = 10$ and $P_E = 10$. And the number of repeated cycle, N_e, is set to 500.

Table 2 illustrates the experimental results. As seen in Table 2, the approximation performances of RAN agents and TC agents are lower than those of RAN-LTMs. Moreover, RAN-LTM2 has smaller errors as compared with RAN-LTM1 in which LTM data are never updated.

Next, we estimate the interference; that is, how much action-values for region 1 are changed after the learning of region 2. Since the states 1-5 are never presented to agents during the learning of region 2, the action-values should degrade seriously if the agents cannot suppress the interference caused by the incremental learning of region 2. Table 3 illustrates the amount of the interference. As seen from this result, little interference arises in TC agents. This is because TC agents learn action-values separately for every small fragments of state space. That is, the changes of action-values within a single tile do not influence on the other tiles. On the other hand, RAN-LTM2 can also suppress the interference effectively as compared with other neural network models. These results suggest that the interference does not arise so much in TC agents but their approximation ability is poor. Moreover, one can say that LTM data should be updated in RAN-LTM as the learning proceeds.

Table 2: Error between theoretical and estimated action values for region 1 and region 2.

	TC	RAN	RAN-LTM1	RAN-LTM2
region 1	0.1448	0.1940	0.1300	0.1108
region 2	0.1398	0.1378	0.1154	0.1083

Table 3: The amount of the interference cased in region 1 after the learning of region 2.

TC	RAN	RAN-LTM1	RAN-LTM2
0.0028	0.0828	0.0630	0.0391

4.2. Extended Mountain-Car Task

Extended Mountain-Car Task is a problem in which a car driver (agent) learns an efficient policy to reach a goal located on the hill between two basins shown in Fig.7. In the original Mountain-Car Task, only the left basin (B_1) is used for learning. Here, the right basin (B_2) is also used as an additional learning domain.

In this problem, when a car agent reaches the left most and right most points, its velocity is reset to zero. The goal of the car agents is to drive up the steep incline successfully and to reach a goal state at the top of the hill as soon as possible. The reward in this problem is -1 at all time steps until the car agent reaches the goal. There are three actions to be selected; "full throttle to goal" and "zero throttle" and "full throttle to opposite side of goal". These actions are coded by the following values: $a(t) = \{+1, 0, -1\}$. A car agent is initially positioned in either of two basins at the beginning of an episode. The position $u(t)$ and velocity $\dot{u}(t)$ are updated by the following dynamics:

$$u(t+1) = B[u + \dot{u}(t)] \quad (12)$$

$$\dot{u}(t+1) = B[\dot{u}(t) + 0.001a(t) - 0.0025\cos(3u(t))]. \quad (13)$$

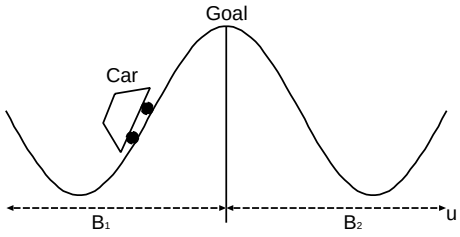

Figure 7: The landscape of the working area in Extended Mountain-Car Task.

Table 4: The average number of steps needed in (a) B_1 after the learning of B_1, (b) B_2 after the learning of B_2, and (c) B_1 after the learning of B_2.

	TC	RAN	RAN-LTM1	RAN-LTM2
(a)	881	326	291	281
(b)	875	424	345	296
(c)	881	2178	284	268

Table 5: The average number of stalls in B_1 after the learning of B_2.

TC	RAN	RAN-LTM	RAN-LTM2
4.9	14	2.6	2.3

B_1 and B_2 are given as follows: B_1:$\{u \mid -1.2 \leq u < 0.5\}$ and B_2:$\{u \mid 0.5 \leq u < 2.2\}$. The goal is located at 0.5. $B[\cdot]$ in Eqs. (12) (13) is used for bounding the areas of B_1 and B_2. The number of input units I is 3. Here, the inputs of a neural network are the position $u(t)$, velocity $\dot{u}(t)$ and previous action $a(t-1)$.

4.2.1. Results and Discussions

Table 4 illustrates the experimental results. In this experiment, P_L, P_E and N_e are set to 10, respectively. As seen in Table 4, TC agents need the most steps to reach the goal, while RAN-LTM2 agents do not need so many steps. That is, TC agents have poor approximation ability as compared with neural agents. Since TC agents learn action-values separately for every tiles, the interference does not occur. However, the continuity of action-values for neighbor tiles is not generally taken into consideration. It seems that this causes the poor generalization ability in TC agents. On the other hand, as seen in Table 4, the proposed agents can acquire proper action-values after incremental learning.

Moreover, we examine the average number of the car's stalls in B_1. If many stalls are occurred, one can say that the agents lost a proper policy in the neighborhood of the goal. Table 5 illustrates the results. As seen in Table 5, the RAN-LTM2 agents experience the smallest number of stalls in B_1. These results also justify that the proposed agents outperforms the other agents in the incremental learning ability.

5. CONCLUSIONS

In this paper, we proposed a new version of Resource Allocating Network with Long Term Memory (RAN-LTM), in which LTM data were properly updated as the learning proceeds. To evaluate the incremental learning ability, the proposed model was applied to the following two tasks: Extended Random-Walk Task and Extended Mountain-Car Task. In these tasks, the working space of agents is extended as the learning proceeds. That is to say, the learning was conducted only for one region at first, and then the learning for different regions was carried out. From the simulation results, we certified that the proposed agents could learn more accurate action-values than TC agents, RAN agents and the previous version of RAN-LTM agents. Moreover, it was verified that the proposed agents could suppress the interference effectively as compared with other agents.

ACKNOWLEDGEMENT

This research has been supported by the Kayamori Foundation of Informational Science Advancement.

6. REFERENCES

[1] R. S. Sutton and A. G. Barto: *Reinforcement Learning - An Introduction*, The MIT Press (1998).

[2] S. Ozawa, T. Tamaoki, and N. Baba: "Incremental Learning for Neural Networks with Long-Term Memory" (in Japanese), *Proc. of the 27th Intelligent System Symposium*, 173/178 (2000).

[3] M. Kotani, K. Akazawa, S. Ozawa, and H. Matsumoto: "Detection of Leakage Sound by using Modular Neural Networks", *Proc. of Sixteenth Congress of the Int. Measurement Confederation*, **IX**, 347/351 (2000).

[4] K. Yamauchi, N. Yamaguchi, and N. Ishii: "Incremental Learning Methods with Retrieving of Interfered Patterns", *IEEE Trans. on Neural Networks*, **10**, 6, 1351/1365 (1999).

[5] M. Kobayashi, A. Zamani, S. Ozawa and S. Abe: "Reducing Computations in Incremental Learning for Feedforward Neural Network with Long-Term Memory", *Proc. Int. Joint Conf. on Neural Networks*, 1989/1994 (2001).

[6] J. Platt: "A Resource Allocating Network for Function Interpolation", *Neural Computation*, **3**, 213/225 (1991).

[7] N. Shiraga, S. Ozawa, and S. Abe: "Learning Action-Value Functions Using Neural Networks with Incremental Learning Ability", *Proc. The Fifth Int. Conf. on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies*, **I**, 22/26 (2001).

[8] J. Morimoto and K. Doya: "Learning Dynamic Motor Sequence in High-dimensional State Space by Reinforcement Learning: Learning to Stand Up" (in Japanese), *Trans. The Institute of Electronics, Information and Communication Engineers* D-II, **J82-D-II**, 11, 2118/2131 (1999).