Learning Action-Value Functions Using Neural Networks with Incremental Learning Ability

Naoto SHIRAGA Seiichi OZAWA Shigeo ABE

Graduate School of Science and Technology, Kobe University, Kobe 657-8501, JAPAN
shiraga@chevrolet.eedept.kobe-u.ac.jp {ozawa, abe}@eedept.kobe-u.ac.jp

Abstract. When the distribution of given training data is biased and temporally varied, it is well known that the learning of neural networks becomes difficult in general. In Reinforcement Learning (RL) problems, such situations often arise. In this paper, an incremental learning system, which has been devised for supervised learning, is implemented as an RL agent that can acquire an action-value function properly even in the above difficult situations. The proposed RL agent is applied to an extended mountain-car task in which learning domains are temporally expanded. Through computer simulations, we demonstrate that the proposed agent can acquire a right policy in this task.

1 Introduction

In Reinforcement Learning (RL) problems, an RL agent adapts itself to the environment so as to maximize the total amount of reward it receives over the long run [1]. Through the interaction with the environment, the agent learns an appropriate policy that maps a state to a right action. In general, instead of learning the policy directly, action-values are introduced into the agent and they are modified based on the error between the estimated reward and the immediate reward called *Temporal Difference* (*TD*) *error*. The action-values are defined for each state-action pair and they give the estimation of the total reward that the agent will earn in the future. In tasks with small and finite state sets, it is possible to form these action-values with a lookup table. In many cases of practical interest, however, there are far more states than could possibly be entries in this table. In this case, a continuous function is used for approximating action-values.

Neural networks have been often adopted to approximate action-value functions. However, it is well known that the learning of neural networks becomes difficult when the distribution of given training data is biased and temporally varied. In such a situation, the input-output relations acquired in the past are easily to be collapsed by the learning of newly given data. This disruption in neural networks is called *interference* that is caused by excessive adaptation of connection weights for new data. This interference can also arise in RL problems because similar trials might not be conducted in a short period.

We have proposed an incremental learning system [2], in which Long-Term Memory (LTM) was introduced into Resource Allocating Network (RAN) [3]. For the notational convenience, this system is denoted as *RAN-LTM*. In RAN-LTM, storage data in LTM (denoted as *LTM data*) are produced from inputs and outputs of RAN whose relationships are accurately approximated. When the learning is carried out, some LTM data are retrieved and they are simultaneously trained to suppress the interference. Although RAN-LTM has been devised for supervised learning, it can be also applied to RL problems by introducing TD errors.

In Section 2, RAN-LTM is implemented as an RL agent (called *RAN-LTM agent*) that can acquire proper action-value functions even when the learning domains are temporally expanded. In Section 3, the RAN-LTM agent is applied to an extended mountain-car task, in which a car driver (RL agent) has to learn a right policy on the throttle operation to reach a goal as soon as possible. The agent's ability in approximation of action-value functions is evaluated through computer simulations. Section 4 presents conclusions of this work.

2 RL Agent with Incremental Learning Ability

The proposed RL agent consists of two parts: *Resource Allocating Network* (*RAN*) [3] and *Long-Term Memory* (*LTM*). RAN is used for approximating an action-value function. The inputs of RAN at time t, x_{ti}, $i = 1, \cdots, I$, are composed of the current agent's states $s_{ti'}$, $i' = 1, \cdots, S$, and the previously selected actions $a_{t-1,i''}$, $i'' = 1, \cdots, A$: i.e. $(x_{t1}, \cdots, x_{tI}) = (s_{t1}, \cdots, s_{tS}, a_{t-1,1}, \cdots, a_{t-1,A})$. Here, S and A are the numbers of agent's states and actions, respectively. The outputs z_{tk}, $k = 1, \cdots, K$, correspond to the action-values, which are utilized for determining agent's actions at time t. Note that the number of input units of RAN, I, is equal to $S + A$ and the number of output units, K, is equal to A. In the followings, the inputs and outputs of RAN are denoted as vectors: i.e. $\boldsymbol{x}_t = \{x_{t1}, \cdots, x_{tI}\}'$ and $\boldsymbol{z}_t = \{z_{t1}, \cdots, z_{tK}\}'$.

While the agent takes actions to achieve a goal, useful pairs of inputs and the associated action-values (called *LTM data*) are automatically produced and accumulated inside the agent. The agent can retrieve these LTM data and learn them in order to sustain its proper action-value function. In learning action-values, the production and retrieval of LTM data are simultaneously carried out inside the agent. The details of these procedures are shown below.

2.1 Resource Allocating Network

RAN is an extended model of Radial Basis Function networks [4]. At the beginning, the number of hidden units is set to one; hence, RAN possesses simple approximation ability at first. As the agent experiences many trials, the approximation ability of RAN is developed by allocating additional hidden units.

The outputs of RAN are calculated as follows:

$$y_{tj} = \exp(-\frac{\| \boldsymbol{x}_t - \boldsymbol{c}_j \|^2}{{\sigma_j}^2}) \quad (j = 1, \cdots, J) \tag{1}$$

$$z_{tk} = \sum_{j=1}^{J} w_{kj} y_{tj} + \theta_k \quad (k = 1, \cdots, K), \tag{2}$$

where y_{tj} is the output of the jth hidden unit at time t, J is the number of hidden units, $\boldsymbol{c}_j = \{c_{j1}, \cdots, c_{jI}\}'$ is a center vector of the jth hidden unit, σ_j^2 corresponds to variance of the jth radial basis function, w_{kj} is a connection weight from the jth hidden unit to the kth output unit and θ_k is a bias of the kth output unit.

Assume that the kth output for $\boldsymbol{x}_t$ corresponds to an action-value of the kth action, $Q(\boldsymbol{x}_t, a_{tk})$: i.e. $z_{tk} = Q(\boldsymbol{x}_t, a_{tk})$. Then, the agent's action is selected based on the following probability function in Stochastic Action Selector (SAS):

$$\mathrm{Prob}(a_{tk}) = \frac{\exp(Q(\boldsymbol{x}_t, a_{tk})/T)}{\sum_{l=1}^{K} \exp(Q(\boldsymbol{x}_t, a_{tl})/T)} = \frac{\exp(z_{tk}/T)}{\sum_{l=1}^{K} \exp(z_{tl}/T)}, \tag{3}$$

where T is a parameter that controls the randomness of action selection. At the beginning, T is set to a large value, and then it is gradually reduced. Now we assume that the k'th action is selected at time t in SAS, and then the agent's states are changed to $s_{t+1,1}, \cdots, s_{t+1,S}$ and an immediate reward r_{t+1} is given. The following TD error E_{tk} for the kth output is defined:

$$E_{tk} = \begin{cases} r_{t+1} + \gamma \max_l \tilde{z}_{t+1,l} - z_{tk} & (k = k') \\ 0 & (\text{otherwise}), \end{cases} \tag{4}$$

where $\tilde{z}_{t+1,l}$ is the lth output calculated by using the next state $\boldsymbol{x}_{t+1}$ and the current network connections. Based on the value of $E_{tk'}$, either of the following procedures is conducted:

1) If $E_{tk'} > \varepsilon$ and $\|\boldsymbol{x}_t - \boldsymbol{c}^*\| > \delta$, a hidden unit is added to RAN (i.e. $J \leftarrow J + 1$). Here, $\boldsymbol{c}^*$ is the nearest center vector to $\boldsymbol{x}_t$. ε and δ are positive constants. The network parameters for the Jth hidden unit are set to as follows: $\boldsymbol{c}_J = \boldsymbol{x}_t$ and $w_{kJ} = E_{tk'}$.
2) Otherwise, the network connections are modified as follows:

$$w_{kj}^{NEW} = w_{kj}^{OLD} + \alpha E_{tk} y_{tj} \tag{5}$$

$$c_{ji}^{NEW} = c_{ji}^{OLD} + 2\frac{\alpha}{\sigma_j}(x_{ti} - c_{ji}) y_{tj} \sum_k E_{tk} \sigma_{kj}, \tag{6}$$

where α is a positive learning ratio.

2.2 Long Term Memory

Each LTM datum is composed of an agent's state and the associated action-value. They are respectively used as an input and its target in the learning of RAN. Therefore, LTM data should be produced from inputs and outputs of networks whose relationships are accurately approximated. In reinforcement learning, however, the approximation accuracy of an action-value function cannot be measured by only TD errors. Hence, the average number of actions taken to achieve a goal is also considered as a criterion for producing LTM data. The average number of actions is estimated occasionally during the learning of the agent; and if it is small enough, the following procedure is carried out to produce LTM data:

1) If all outputs of hidden units y_{tj}, $j = 1, \cdots, J$, for $\boldsymbol{x}_t$ are less than a threshold value θ_c, then go to Step 5. Otherwise, go to Step 2.
2) Obtain all indices j of hidden units whose outputs y_{tj} are larger than θ_c, and we define a set $\mathcal{I}_1$ of these indices. The variable q_j is updated for all $j \in \mathcal{I}_1$ as follows: $q_j \leftarrow q_j + 2\exp(\rho E_{tk'}) - 1$, where k' means the index of the selected action. $E_{tk'}$ is the TD error for the k'th output $z_{tk'}$ and ρ is a positive constant.
3) If $q_j > \beta$ for $j \in \mathcal{I}_1$, then q_j is initialized and go to Step 4. Otherwise, go to Step 5. Here, β is a positive constant.
4) If no LTM datum for the jth hidden unit had been produced yet, then 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. The input-output pair $(\tilde{\boldsymbol{x}}_M, \tilde{\boldsymbol{z}}_M)$ is stored into LTM. The number of LTM data, M, increases by one (i.e. $M \leftarrow M + 1$).
5) $t \leftarrow t + 1$ and go to Step 1.

LTM data are retrieved and utilized for learning the RL agent. In order to learn as efficiently as possible, the suitable number of LTM data should be recalled. Here, based on the activation of hidden units, LTM data to be retrieved are determined as follows:

1) Obtain all indices j of hidden units whose outputs y_{tj} 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 Step 3. Otherwise, no LTM datum is retrieved. Go to Step 5.
3) Obtain LTM data nearest to the center vectors $\boldsymbol{c}_j$ for all $j \in \mathcal{I}_2$.
4) Recall these LTM data and the output errors are calculated for each LTM datum. Based on the errors, connection weights of RAN are modified.
5) $t \leftarrow t + 1$ and go to Step 1.

3 Simulations

3.1 Extended Mountain-Car Task

To examine the performance of the proposed RL agent, we apply it to an extended version of *Mountain-Car Task*, in which a car driver (RL agent) learns an efficient policy to reach a goal located on the hill between two basins shown in Fig. 1. In the original Mountain-Car Task [1], only the left basin (B_1) is adopted in learning. Here, the right basin (B_2) is also adopted as an additional learning domain. Suppose that a stationary car is positioned in either of two basins at the beginning of an episode. The goal of the RL agent is to successfully drive up the steep incline and to reach a goal state at the top of the hill as soon as possible. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. Therefore, the RL agent has to learn a policy that it first moves away from the goal and up the opposite slope. Then, by applying full throttle the car can build up enough inertia to carry it up the steep slope.

The state of the RL agent is the car's position and velocity. Three actions a_t are available to the agent in each state: full throttle forward ($+1$), full throttle reverse (-1), and zero throttle (0). The reward is -1 for all state transitions except the transition to the goal state, in which case a zero reward is returned. The car moves according to a simplified physics. Its position x_t and velocity $\dot{x}_t$ are updated by the following dynamics:

$$x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}], \quad \dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)],$$

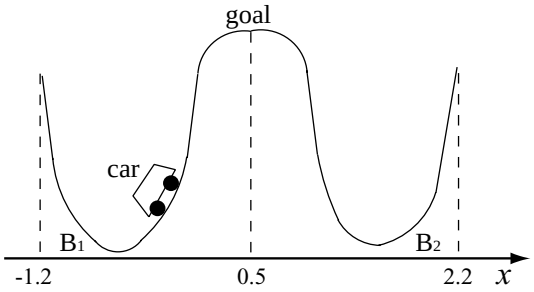

Figure 1: An extended mountain-car task.

Table 1: The average number of steps obtained through 100 episodes and the total time to learn B_1 and B_2. In the column of B_1, 'before' and 'after' respectively correspond to the results of average steps evaluated for B_1 before and after the learning for B_2 converges.

	B_1		B_2	Learning Time (sec.)
	before	after		
RAN agent	811	1595	679	110
RAN-LTM agent	790	541	922	250

where the bound operation enforces $-1.2 \leq x_{t+1} \leq 2.2$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. When x_{t+1} reaches the left and right bound (i.e. $x_{t+1} = -1.2$ and 2.2), $\dot{x}_{t+1}$ is reset to zero. In each episode, the agent starts from a random position with velocity uniformly chosen from the above range. When the agent reaches the goal, the episode is terminated.

3.2 Results

Here, we shall apply the proposed RL agent to the extended mountain-car task in which the learning of the action-value function is conducted only for B_1 at first, and then the learning for B_2 is carried out. That is to say, at first, the RL agent repeatedly experiences episodes in which initial positions are randomly chosen only from B_1, and the learning is continued until the TD error becomes small enough. After the learning for B_1 converges, the RL agent experiences new episodes in which initial positions are randomly chosen only from B_2. Therefore, the episodes experienced in the former learning never occur in the latter learning. After the learning for B_2 is completed, we evaluate the interference by examining the average number of steps needed until the RL agent reaches the goal. Here, the number of steps is equivalent to the number of agent's actions taken in an episode.

Table 1 shows the average number of steps obtained through 100 episodes and the total time to learn B_1 and B_2. In each episode, the RL agent starts from a randomly chosen position in B_1 or B_2. To show the usefulness of the proposed agent (denoted as *RAN-LTM agent*), the performance of the agent using the original RAN (denoted as *RAN agent*) is also examined. The parameters of RAN and Long-Term Memory (LTM) are given as follows:

[RAN] $\varepsilon = 1.0,\ \kappa = 1.0,\ \delta_{max} = \delta_{min} = 0.5,\ \tau = 50,\ \alpha = 0.00001,\ \sigma_j = 0.21,\ \gamma = 0.99$
[LTM] $\theta_c = 0.9,\ \theta_r = 10^{-2},\ \rho = 1,\ \beta = 0.01,\ \eta = 0.01.$

The average number of steps are evaluated for B_1 before and after the learning for B_2 is carried out. As we can see in Table 1, the number of needed steps in B_1 excessively increases in the RAN agent after the learning for B_2; on the other hand, it decreases in the RAN-LTM agent. Considering that there is not so much difference between the RAN agent and the RAN-LTM agent in terms of needed steps before the learning for B_2, this result suggests that the RAN-LTM agent can sustain proper action-values acquired in the past even if lately presented episodes are quite different from those presented in the past. This result is also certified from Fig. 2, in which the variations of action-values are shown for different positions x in B_1. These variations are given by differences between action-values acquired before and after the learning of B_2. As seen in Fig. 2, due to the interference, the action-values of the RAN agent

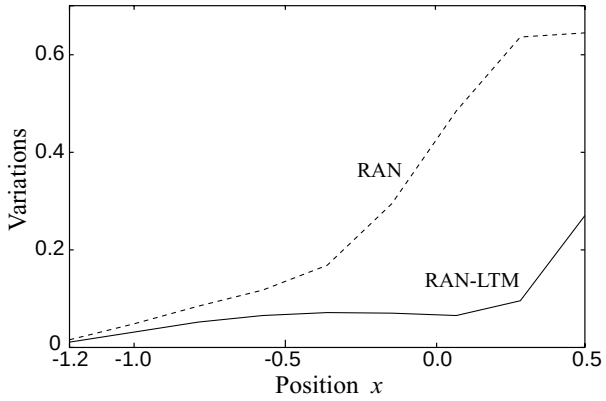

Figure 2: The variations of action-values in B_1.

are varied seriously especially around the boundary between B_1 and B_2 (i.e. $x = 0.5$). On the other hand, the variations are fairly suppressed in the RAN-LTM agent; this suggests that a proper policy for B_1 is maintained.

However, it seems that the larger number of steps in B_2 is needed for the RAN-LTM agent. Since the learning for B_2 was carried out with the fixed number of iterations, it is supposed that the TD error in the RAN-LTM agent did not become small enough within the repetitions, and then the larger number of steps was needed to reach the goal. Furthermore, the learning time of the RAN-LTM agent is two times longer than that of the RAN agent. This is because not only immediate reward but also retrieved LTM data should be learned in the RAN-LTM agent. Therefore, we should devise a more efficient retrieval mechanism for LTM data in order to speed up learning.

4 Conclusions

In this paper, Resource Allocating Network with Long-Term Memory (RAN-LTM) was extended to reinforcement learning. In RAN-LTM, significant pairs of inputs and the associated action-values are automatically stored into Long-Term Memory (LTM). These LTM data are retrieved and utilized for the learning such that the agent holds good approximation accuracy of action-value functions. We evaluated the performance of the RAN-LTM agent in an extended mountain-car task, in which a car driver (RL agent) should learn an efficient policy to reach a goal located on the hill. The learning was conducted only for one basin at first, and then the learning for the other basin was carried out. From the simulation results, we certified that the proposed RAN-LTM agent could suppress the interference and learn a proper policy as compared with the RAN agent.

Acknowledgement

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

References

[1] R. S. Sutton and A. G. Barto: *Reinforcement learning – An introduction*, The MIT Press (1998).

[2] M. Kobayashi, A. Zamani, S. Ozawa, and S. Abe: "Reducing computations in incremental learning for feedforward neural network with long-term memory", *Proc. 2001 Int. Joint Conf. on Neural Networks* (in press).

[3] J. Platt: "A resource allocating network for function interpolation", *Neural Computation*, **3**, 213/225 (1991).

[4] T. Poggio and F. Girosi: "Networks for approximation and learning", *Proc. IEEE Trans. on Neural Networks*, **78**, 9, 1481/1497 (1990).