

Influence of the Space Segmentation and its Adaptive Automation for Reinforcement Learning

Akira Notsu, Yuki Komori, Katsuhiro Honda and Hidetomo Ichihashi

Osaka Prefecture University

1-1 Gakuencho, Nakaku, Sakai, Osaka 599-8531, JAPAN

Email: {notsu, honda, ichi}@cs.osakafu-u.ac.jp

Abstract—We performed a single pendulum simulation and observed the influence of the situation space segmentation pattern in reinforcement learning processes in order to propose a new adaptive automation for situation space segmentation. Usually, in real-world reinforcement learning processes, infinite states and actions and the uncertainty of the optimum solution make the learning process more difficult than the finite Markov decision process. In a numerical experiment, a single pendulum simulation is performed in order to demonstrate the influence and adaptability of the proposed method.

Index Terms—reinforcement learning, space segmentation.

I. INTRODUCTION

Reinforcement learning is a framework for a strategy to learn through trial and error according to the reward provided by the environment[1]. In reinforcement learning, if the purpose of study is set, the agent automatically determines the best way to achieve the goal.

Reinforcement learning is chiefly applied to problems in the finite Markov decision process with a finite state and a finite action. However, recently, research on methods of expressing a continuous state and a continuous action, which is more similar to the real-world situation, has progressed. Research on motion control and autonomous control of the robot has been extensive. For example, studies have examined a control problem in order to enable the robot to walk. However, in the real world, this problem is difficult, as compared with the problem in the Markov decision process, because there are an infinite number of states and actions, and it is unclear as to whether an optimum solution exists.

Q-learning[2], [3] and SARSA are typical reinforced study methods that target discrete actions. Solving a problem that has a continuous state and a continuous action requires discrimination of the continuous state and continuous action.

In the present thesis, we investigate the method of this discretization, how to divide in order to find the optimum solution faster, and the technique by which an agent can acquire the best approach to automatically divide states, and the headstand stabilization problem of the pendulum was considered.

II. REINFORCEMENT LEARNING

Reinforcement learning assumes that agents can visit only a finite number of states, and that when visiting a state, they will collect a numerical reward (which can be interpreted as a punishment in the case of a negative reward). Each state

has a different value associated with it. Given an initial state, subsequent states are reached according to actions. The value of a given state is defined as the average reward that can be collected by selecting actions available in that state.

A. Q-learning

Q-learning is a well-known reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking given action a in given state s and following a fixed policy thereafter. One advantage of Q-learning is that Q-learning can compare the expected utility Q of available actions without requiring a model of the environment (usually, the agent selects the action with the maximum expected utility).

Expected utility Q is updated iteratively. For each state s from state set S , and for each action a from action set A , we update Q-value Q that depends on current state s of the agent, action a selected by the agent, the next state s' of the agent after taking action a , and reward r received by the agent after taking the action. An update is calculated by the following expression:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{\hat{a} \in A(\hat{s})} Q(\hat{s}, \hat{a}) - Q(s, a) \right], \quad (1)$$

where α is the learning rate and γ is the discount factor. Once Q-learning has finished, the optimal policy and the optimal value function have been found without continuously updating the policy during learning.

Q-learning algorithm:

```

procedure Q-learning
begin
  initialize Q-table  $Q$ ,  $\forall s \in S, \forall a \in A$ ;
  set initial state  $s_0$  and goal state  $s_n$ ;
  for cycle := 1 to MAXCYCLE do  $s := s_0$ 
    while  $s \neq s_n$  do
       $a := \text{ActionSelect}(Q, s)$ ;
       $r := \text{GetReward}(s, a)$ ;
       $s' := \text{GotoNextState}(s, a)$ ;
       $Q := \text{UpdateQ}(Q, s, a, \text{trace})$ ;
       $s := s'$ ;
    end
  end
end

```

B. State and action space segmentation

A high-performance agent usually requires appropriately segmented state set S and action set A due to the state-space explosion of an agent. Generally, a genetic algorithm is used to search for appropriate segmentation in such individual double-loop learning. In other words, the particle swarm is a classification system in which we consider the minimization of (3) under the following condition:

$$\sum_{i=1}^c u_{ki} = 1, \quad (2)$$

where c is the number of clusters, and u_{ki} is called a membership function.

$$L = \sum_{i=1}^c \sum_{k=1}^N u_{ki} D(x_k, v_i), \quad (3)$$

where N is the amount of data, c is the number of clusters, and $D(x_k, v_i)$ denotes the distance between data vector x_k and cluster center v_i , which is the difference between expected utility V and observed utility V' in the learning process.

C. Softmax Action Selection

In the present simulation, we use softmax action selection rules as an action selection policy. Softmax action selection rules rank and weight according to Q-value Q . The most common softmax method uses a Gibbs or Boltzmann distribution. This method chooses an action on the play with probability.

$$P(s, a) = \frac{\exp^{Q(s, a)/\tau}}{\sum_{\hat{a} \in A} \exp^{Q(s, \hat{a})/\tau}} \quad (4)$$

where τ is the temperature parameter. High temperatures cause (almost) all of the actions to be equiprobable, and low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. If $\tau \rightarrow \inf$, then softmax action selection becomes the same as greedy action selection.

III. SINGLE PENDULUM SIMULATION

We simulated the single pendulum standing task (fig. 1). This physical calculation is performed by Runge-Kutta 4th order method.

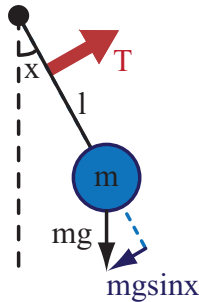


Fig. 1. Single pendulum

The following motion equation is simulated in increments of 0.05 seconds[4]:

$$m\ddot{x} = -mgl \sin x - k\dot{x} + T \quad (5)$$

where the parameters and variables are defined as follows:

- $m = 1(kg)$
- $l = 1(m)$
- $k = 0$
- $g = 9.8(m/s^2)$
- $-\pi < \theta \leq \pi$
- $-3\pi \leq \dot{\theta} \leq 3\pi$
- $-20 \leq T \leq 20$
- 1 iteration = 20 seconds
- 1 learning cycle = 0.2 seconds
- $\theta = \pi, \dot{\theta} = 0$, in the initial state
- $r_t = \frac{|x|}{\pi} - 0.5 \frac{|T|}{20}$
- $\alpha = 0.25$
- $\gamma = 0.9$
- $\tau = 1$ at the first learning cycle, $\tau = 0.001$ at the last learning cycle (linearly decreasing)

A. Segmentation pattern in the state

First, we generated some patterns by dividing the state space, and a simulation was conducted.

```

procedure Q-learning with segmentation pattern
begin
  initialize Q-table  $Q, \forall s \in S, \forall a \in A$ ;
  set initial state  $s_0$ ;
  select dividing methods;
  for cycle := 1 to MAXCYCLE do  $s := s_0$ 
    for time := 0 : 0.05 : 20
       $s' := \text{GetNextState}(s, a)$ ;
      if mod(time, 0.2) == 0;
         $s'_r := s'$ ;
         $r := \text{GetReward}(s'_r, a)$ ;
         $Q := \text{UpdateQ}(Q, s_r, s'_r, a)$ ;
         $a := \text{ActionSelect}(Q, s'_r)$ ;
         $s_r := s'_r$ ;
      end
       $s = s'$ ;
    end
  end
end

```

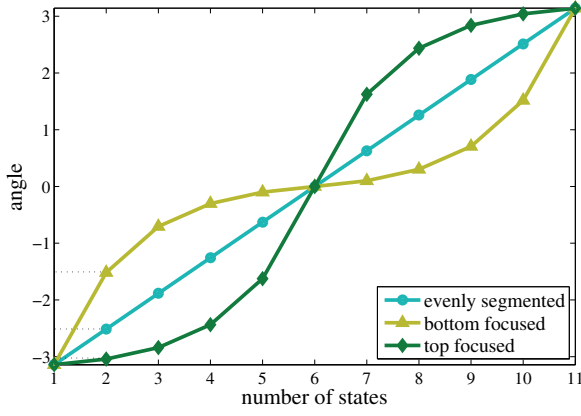


Fig. 2. dividing methods

Fig. 2 show the rate of the space segmentation. Vertical axes show the rate of the angle width (width of each segment) and horizontal axes show the angle of each segment.

For example, for “evenly segmented,” the space is evenly segmented to 10 parts. for “bottom focused,” the space is segmented larger around the bottom, and smaller around the top. In other words, the bottom is segmented to many parts and the agent can find difference of angle even small difference compared with the top. for “top focused,” on the contrary, the top is segmented to many parts.

Other parameters and variables are defined as follows:

- 1000 episodes $\times$ 30
- θ and $\hat{\theta}$ are segmented into 10 segments
- T is segmented into six segments
- $\hat{\theta}$ and T are segmented evenly

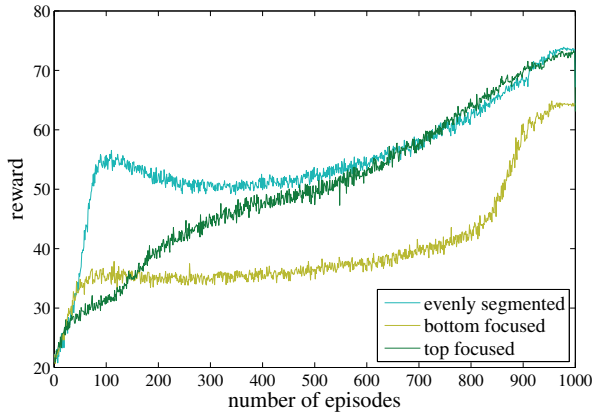


Fig. 3. comparison of the three methods

The results indicate that “evenly segmented” and “top focused” eventually provided good rewards, whereas “bottom focused” provided smaller rewards. In addition, the “bottom focused” requires more iterations than the first method in order to provide good rewards.

B. Segmentation and integration method

We investigated an automatic segmentation and integration method through simulation.

```

procedure  $Q$ -learning with segmentation and integration
begin
  initialize  $Q$ -table  $Q$ ,  $\forall s \in S$ ,  $\forall a \in A$ ;
  set initial state  $s_0$ ;
  for cycle := 1 to MAXCYCLE do  $s := s_0$ 
    for time := 0 : 0.05 : 20
       $s' := \text{GetNextState}(s, a)$ ;
      if mod(time, 0.2) == 0;
         $s'_r := s'$ ;
         $r := \text{GetReward}(s'_r, a)$ ;
         $Q := \text{UpdateQ}(Q, s_r, s'_r, a)$ ;
         $a := \text{ActionSelect}(Q, s'_r)$ ;
         $s_r := s'_r$ ;
      end
       $s = s'$ ;
    end
    if mod(cycle, FREQUENCY) == 0;
       $(S, Q) := \text{Combine}(S, Q)$ ;
       $(S, Q) := \text{Divide}(S, Q)$ ;
    end
  end
end

```

In this method, the space is initially evenly segmented, segmentation and integration is performed at regular cycles by the $\text{Combine}(S, Q)$ and $\text{Divide}(S, Q)$ functions.

The neighboring states that an average of the visited frequency is the least are integrated into a single state that has an average expected utility of the states by the $\text{Combine}(S, Q)$ function. And the state that the visited frequency is the most is equally segmented into two states that have the same expected utility by the $\text{Divide}(S, Q)$ function. Other expected utilities are undisturbed.

Other parameters and variables are defined as follows:

- 1000 episodes $\times$ 30
- θ and $\hat{\theta}$ are segmented into 10 segments
- T is segmented into six segments
- θ is segmented evenly, in the initial state
- $\hat{\theta}$ and T are segmented evenly

First of all, to decide how often states are combined and divided, each various episodes was combined and divided. These average rewards from 951 episodes to 1000 episodes were shown in TABLE I.

As a result, “every 200 episodes” had the best reward. So after this, we chose that the states were combined and divided every 200 episodes.

The proposed method eventually provides the best reward and the reward was increasing especially every 200 episodes, which indicates that the proposed method can be used to determine an appropriate segmentation method automatically over time.

Next, how to divide the states after the learning of “proposed method” was checked and the states were shown in fig.

TABLE I

HOW OFTEN COMBINE AND DIVIDE AND THE AVERAGE REWARD

how often combine and divide	the average reward
every 50 episodes	73.7279
every 100 episodes	77.3704
every 200 episodes	78.3863
every 300 episodes	76.4497
every 400 episodes	75.4669
every 500 episodes	74.6121
not combine and divide	73.2671

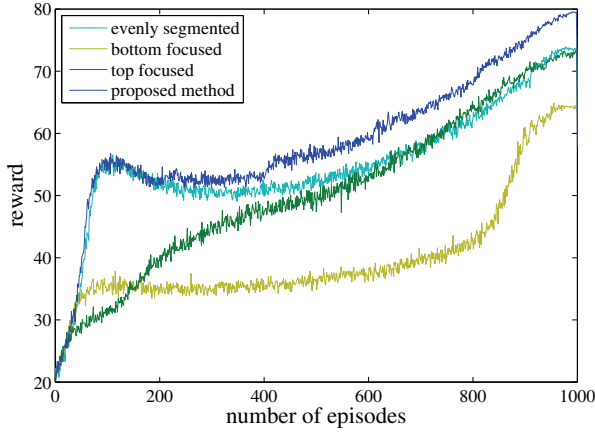


Fig. 4. comparison of the four methods

5. The result was got by average of 100 times.

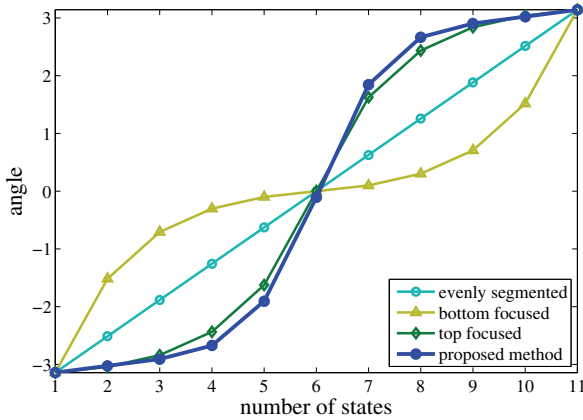


Fig. 5. final dividing methods

Following results were observed in fig. 5, “proposed method” had feature which was emphasized the feature of “top focused.” In short, the top was segmented to many parts and the bottom was segmented to few parts.

Finally, the experiments with the finally segmented states of the proposed method were done from the beginning. In the experiments, the states were not combined and divided. The results were following fig. 6, 7.

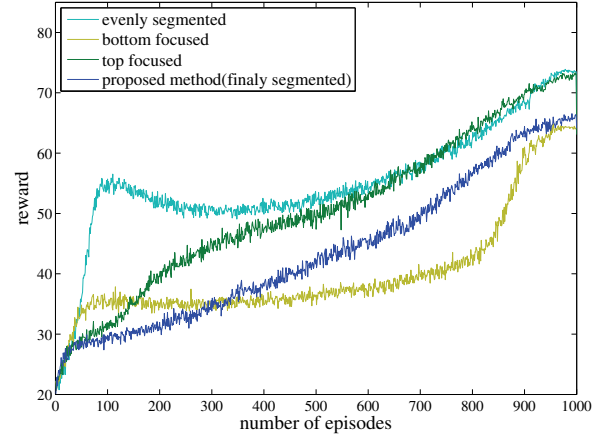


Fig. 6. results in 1000 episodes

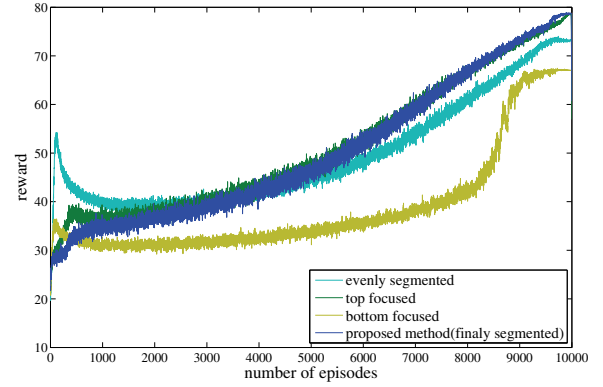


Fig. 7. results in 10000 episodes

Figures show that “1000 episodes” was not enough and “10000 episodes” was enough to get good rewards. This means the dividing way needs much time than the proposed method.

IV. CONCLUSION

We simulated a single pendulum simulation and observed the influence of the space segmentation and its adaptive automation for reinforcement learning. In order to divide state space, three methods were proposed, and “evenly segmented” and “top focused” got good rewards in the end episode.

The proposed method eventually provides the best reward and the reward was increasing especially every 200 episodes, which indicates that the proposed method can be used to determine an appropriate segmentation method automatically over time.

But, when the experiments with the state which were gotten by simulation were done from the beginning, the experiments needed much episodes than the proposed method to get reward at the same level. This shows that an appropriate division method is different depending on the advancement condition of study.

In the future, it is needed that a technique for integrating non-adjacent states, and expressed state space by using a function and particles, etc. In addition, more complicated experiments are also needed. For example, double pendulum simulation.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Education, Culture, Science and Technology, Japan under Grant-in-Aid for Scientific Research No. 21700242 and funded by the Ministry of Internal Affairs and Communications, Strategic Information and Communications R&D Promotion Programme.

REFERENCES

- [1] R. S. Sutton, Learning to Predict by Method of Temporal Differences, Machine Learning, vol. 3, 1, pp. 9-44, 1988.
- [2] T. Jaakkola, M. Jordan, and S. P. Singh, On the Convergence of Stochastic Iterative Dynamic Programming Algorithms, Neural Computation, vol. 6, pp. 341-362, 1992.
- [3] C. J. C. H. Watkins, and P. Dayan, Technical Note: Q-Learning, Machine Learning, vol. 8, pp. 56-68, 1992.
- [4] Y. Kashimura, A. Ueno, S. Tatsumi, A Continuous Action Space Representation by Particle Filter for Reinforcement Learning, JSAI2008, pp. 118-121, 2008
- [5] J. H. Holland, and J. Reitman, Cognitive Systems Based on Adaptive Algorithms, in D. A. Waterman and F. Hayes-Roth (Eds.), Pattern Directed Inference Systems, Academic Press, 1978.
- [6] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard, Induction, The MIT Press, 1986.