Introduction of Majority Vote of Neighborhood Conditions for Sneak Form Reinforcement Learning

Yuki Tezukaa, Akira Notsua, Katsuhiro Hondaa

aOsaka Prefecture University, 1-1 Gakuen-cho, Nakaku, Sakai, Osaka 599-8531, Japan

Abstract

Chain Form Reinforcement Learning (CFRL) was proposed for a reinforcement learning agent using low memory. In this paper, we introduce Sneak Form Reinforcement Learning (SFRL). SFRL is the method where we improve CFRL in terms of Contextual Learning. If a sequence of state-action pairs has a shortest path, a SFRL agent cuts and saves the path. To improve the performance of SFRL, we introduce Majority Vote of Neighborhood Conditions for SFRL and call this method MVNC. Majority Vote of Neighborhood Conditions is the rule which agent in an unknown state selects an action not at random but with circumjacent information. Our methods were made a comparison to Q-Learning and CFRL in several easy simulations. We examined performance and discussed the best usage environment.

Keywords: Reinforcement learning; State-Action set categorization; Majority Vote;

1. Introduction

Reinforcement learning [1] is applied to problems in the finite Markov decision process with a finite state and action. Recently, the research on methods of expressing a continuous state and action, which is more similar to the real-world situation, has progressed. However, there are an infinite number of states and actions in the real world. Therefore, the learning agent needs a lot of memory to learn optimum solutions in the Markov decision process.

Q-learning [2, 3] and SARSA [4] are typical reinforcement learning methods that target discrete actions. Solving a problem that has a continuous state and action requires discrimination of the continuous state and action. In the Q-learning, $Q(s,a)$ is a continuous value and shows how good an action is. We finally need only the essential sequence of actions to arrive in a goal. However, we might evenly store useless information. To reduce useless information, Chain Form Reinforcement Learning(CFRL)[7] was proposed. The aim of CFRL is to make memory small. In the CFRL, several expected utilities are categorized into “GOOD” in the reinforcement learning process. Additionally, the alignment sequence of expected utilities is rearranged as their priorities represented by the sequences themselves in the process. As a result, this can implicitly show whether an action-state would be better than previous action or not and how long the agent gets a reward.
In the paper, we developed CFRL in terms of Contextual Learning and call this method Sneak Form Reinforcement Learning (SFRL). The aim of SFRL is to make memory smaller and reduce learning time. If a sequence of state-action pairs has a shortest path, a SFRL agent cuts and saves the path. End of an episode, the memory table has only the most appropriate path. We accomplish our aim through these processes.

However, performance of SFRL is not as good as those of Q-learning and CFRL because expected utilities of SFRL do not have redundant information. To solve this problem, we introduce Majority Vote of Neighborhood Conditions for SFRL and call this method MVNC. Majority Vote of Neighborhood Conditions is the rule which agent in an state condition selects an action with circumjacent information. As a result, the performance of MVNC is not less than those of Q-learning and CFRL in addition to make memory smaller and reduce learning time.

Our method was made a comparison to Q-learning and CFRL in some easy simulations. We examined performance and discussed the best usage environment.

2. Chain Form Reinforcement Learning (CFRL)

In Q-learning, \(Q(s,a)\) is a continuous value and shows how good an action is. Speaking in the extreme, we might evenly store useless circumjacent information because we finally need only the sequence of actions. Precisely expected utility \(Q\) is not always required in the policy for modestly successful action selection. Therefore, we propose Chain Form Reinforcement Learning. We store a “GOOD” state-action pair and the pair which leads up to it as a “GOOD” pair in series. In the CFRL, we simply store expected utilities through join good state-action pairs together, where good state-action pairs are the pairs which lead to get a reward or good pairs. The closer a pair is to the state which gets a reward, the better the pair is. We need to store it preferentially. Expected utilities of CFRL are represented by sequences instead of traditional real-values. We show Figure 2 and 3 to explain CFRL briefly through the use of the problem of Figure 1.

We solved the problem of Figure 1 with Q-learning and CFRL. In the problem, there are five states and two actions. An agent gets rewards 10 (-10) at state s5 (state s1) and goes back to initial state s3. Table 1 shows the Q-table of the problem after sufficient learning \((\gamma = 0.9)\) and Table 2 shows the CFRL Value-table of the example problem after sufficient learning. An CFRL agent selects action with \(\epsilon\)-greedy algorithm.

In this paper, we insert a newer allocated state-action set without delay.
3. Sneak Form Reinforcement Learning (SFRL)

In the CFRL, we cannot store some most appropriate paths at a time. We hold unused information in the memory. To hold only the best answer, we proposed Sneak Form Reinforcement Learning (SFRL). Figure 4 shows a diagrammatic illustration of SFRL.

![Diagrammatic illustration of SFRL](image)

Fig. 4. Diagrammatic illustration of SFRL

**Process of SFRL**

```plaintext
procedure Proposed Method
begin
set initial state $s_0$ and goal state $s_n$;
for cycle:= 1 to MAXCYCLE do $s := s_0$;
MOVECOUNT:= 0;
while $s \neq s_n$ do
  $a := $ActionSelect(Value, $s$);
allocate memory $s$ and $a$ in top of the SearchMemory
$s := $GotoNextState($s, a$);
MOVECOUNT := MOVECOUNT + 1;
end
if BEST > MOVECOUNT
  if length(BestMemory) > length(SearchMemory)
    BestMemory := SearchMemory;
  end
  BEST := MOVECOUNT;
else if MOVECOUNT \neq 1000
  BestMemory := ShortCut_M(BestMemory, SearchMemory);
  BestMemory := ShortCut_S(BestMemory, SearchMemory);
end
end
end
```

Figures 5 and 6 show we store a sequence of state-action pairs leading a good state and update a memory. In ShortCut_M showed by Process of SFRL, we explore nodes with reference to a state-action pair which is stored as the best answer (BestMemory) and, in ShortCut_S, we do with reference to the pair which is got after an episode (SearchMemory). Following explanations is in the case of ShortCut_M.

Figure 5: We explore a node through make a comparison to BestMemory and SearchMemory. If we find a node, we store it (node). If we find that the node is already stored, we do not store it.

Figure 6: We consult nodes whether SearchMemory has a shortest path or not. If SearchMemory has it (dis_B > dis_S and dis_S is odd), we erase an indirect path from BestMemory and store the shortest path to BestMemory, where dis_B and dis_S are distance between stored positions of two points which are focused on. In
this explanation, we focus attention on S2 and S5. Position of S2 (S5) is the place where it is stored. We gained \( \text{dis}_S \) (dis\(_B\)) through subtracting Position of S5 from Position of S2.

When we cannot find a shortest path, we finish the process. Repeating these processes after finishing an episode, we quickly gain a most appropriate sequence of state-action pairs.

---

4. Introduction of Majority Vote of Neighborhood Conditions

We proposed SFRL in a previous section. We achieve a reduction of the memory used by an agent after learning a problem and a learning time. However, performance of SFRL is not as good as those of Q-learning and CFRL. This is because expected utilities of SFRL do not have redundant information. To improve the performance, we introduce Majority Vote of Neighborhood Conditions for SFRL and call this method MVNC.

We explain Majority Vote of Neighborhood Conditions with reference to Figure 7. In Figure 7, an agent (A) stays at an unknown state of a two-dimensional field. In the SFRL, an agent selects an action at random. Compared to this, in the MVNC, it does with circumjacent information. Then, an agent uses \( \epsilon\)-greedy algorithm. Therefore, there is the highest possibility of selecting the action 1 (go up) in the MVNC.

---

Fig. 5. Exploring nodes

Fig. 6. Update BestMemory

Fig. 7. Majority Vote of Neighborhood Conditions

Fig. 8. flowchart of MVNC
5. Simulation experiments

We simulated a standard game called “goal search” as a test case. We used two kinds of games. One does not have blocks (Experiment 1) and the other has blocks (Experiment 2). In this game, agent actions are up and down per each dimension. A total of agent actions are $2 \times \text{dim}$, where a dimension number is $\text{dim}$. If the agent goes to the goal, it gets a reward and goes back to the start state. A total of agent states are $n^{\text{dim}}$, where a side size of a field is $n$. In Experiment 2, we set $\text{dim}=2$ and $n=20$. We set blocks at random and arranged 100 fields in which agent takes more than 38 turns to go to the goal. The agent can observe its state completely. Figure 9 shows the field ($\text{dim}=3$ and $n=5$) in Experiment 1 and Figure 10 shows an example of field in Experiment 2.

![Fig. 9. Example field in Experiment 1](image1)

![Fig. 10. Example field in Experiment 2](image2)

**Simulation parameters**

Agents’ policies are $\epsilon$-greedy. The following parameters are set. In Experiment 1, $n$ depends on a dimension of problem (dim) and is showed by Table 5.

- Learning rate $\alpha = 0.1$ in Q-learning.
- Discount ratio $\gamma = 0.9$ in Q-learning.
- Start state is $(1, 1, ..., 1)$
- Goal state is $(n, n, ..., n)$

<table>
<thead>
<tr>
<th>Table 3. side size of each field</th>
</tr>
</thead>
<tbody>
<tr>
<td>dim</td>
</tr>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

If an agent gets a reward (go to goal state) or performs 1,000 actions, the episode ends and the next one begins where agents are set to the initial state.

6. Simulation results

6.1. Experiment 1 (Goal Search)

Figure 11-14 show the result of Experiment 1. The vertical axis shows the goal turn, and the horizontal axis shows the number of episodes. The blue, green, red and cyan lines respectively correspond to a Q-learning agent, a CFRL agent, a SFRL agent and a MVNC agent. The mean and the median of the number of learned actions during 500 episodes are shown in Table 4. Table 5 shows an example of amounts of memories used by each agent after learning a problem in Experience 1.

Table 5 shows that amounts of used memory which used by SFRL and MVNC agents are about $1/12$ of Q-learning and about $1/10$ to $1/4$ of CFRL in two-dimensional fields. In particular, they are about $1/100$ of Q-learning in five-dimensional fields. Therefore, we achieved significant reduction of memory used by an agent.
Fig. 11. Mean learning process ($\epsilon=0.1$, dim=2, 100 agents)

Fig. 12. Mean learning process($\epsilon=0.8$, dim=2, 100 agents)

Fig. 13. Mean learning process ($\epsilon=0.1$, dim=3, 100 agents)

Fig. 14. Mean learning process ($\epsilon=0.1$, dim=5, 100 agents)

Table 4. Mean (and median) of the number of learned actions during 500 episodes (100 agents, Experiment 1)

<table>
<thead>
<tr>
<th>dim</th>
<th>$\epsilon$</th>
<th>Q-Learning</th>
<th>CFRL</th>
<th>SFRL</th>
<th>MVNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.05</td>
<td>34.62(34)</td>
<td>34.32(34)</td>
<td>35.48(34)</td>
<td>34.02(34)</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>34.36(34)</td>
<td>34.14(34)</td>
<td>35.46(34)</td>
<td>34.02(34)</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>34.18(34)</td>
<td>34.10(34)</td>
<td>35.08(34)</td>
<td>34.04(34)</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>34.02(34)</td>
<td>34.22(34)</td>
<td>34.54(34)</td>
<td>34.06(34)</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>34.0(34)</td>
<td>35.44(35)</td>
<td>34.46(34)</td>
<td>34.22(34)</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>28.0(27)</td>
<td>27.38(27)</td>
<td>28.74(29)</td>
<td>27.0(27)</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>16.42(16)</td>
<td>16.26(16)</td>
<td>16.46(16)</td>
<td>16.0(16)</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>15.82(15)</td>
<td>15.32(15)</td>
<td>15.52(15)</td>
<td>15.0(15)</td>
</tr>
<tr>
<td>6</td>
<td>0.1</td>
<td>12.42(12)</td>
<td>12.34(12)</td>
<td>12.52(12)</td>
<td>12.0(12)</td>
</tr>
</tbody>
</table>

Table 5. Amount of used memory (after learning a problem in Experiment 1)

<table>
<thead>
<tr>
<th>dim</th>
<th>$\epsilon$</th>
<th>Q-Learning</th>
<th>CFRL</th>
<th>SFRL</th>
<th>MVNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.1</td>
<td>4(double) × 20 × 20</td>
<td>2(int) ×130</td>
<td>2(int) ×34</td>
<td>2(int) ×34</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>4(double) × 20 × 20</td>
<td>2(int) ×323</td>
<td>2(int) ×34</td>
<td>2(int) ×34</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>6(double) × 10 × 10 × 10</td>
<td>2(int) ×289</td>
<td>2(int) ×29</td>
<td>2(int) ×27</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>10(double) × 4 × 4 × 4 × 4</td>
<td>2(int) ×298</td>
<td>2(int) ×15</td>
<td>2(int) ×15</td>
</tr>
</tbody>
</table>
6.2. Experiment 2 (Goal search with blocks)

Figure 15-18 show the result of Experiment 2. The vertical axis shows the goal turn, and the horizontal axis shows the number of episodes. The blue, green, red and cyan lines respectively correspond to a Q-learning agent, a CFRL agent, a SFRL agent and a MVNC agent. The mean and the median of the number of learned actions during 500 episodes are shown in Table 6. Table 7 shows an example of memories which used by each agent after learning a problem in Experience 2.

Table 7 shows that amounts of memory used by SFRL and MVNC agents are about 1/16 of that of Q-learning and about 2/11 to 2/5 of that of CFRL in two-dimensional fields with blocks.

<table>
<thead>
<tr>
<th>dim</th>
<th>ϵ</th>
<th>Q-Learning</th>
<th>CFRL</th>
<th>SFRL</th>
<th>MVNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.05</td>
<td>119.62(43)</td>
<td>110.02(42)</td>
<td>46.18(45)</td>
<td>45.42(44)</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>107.28(42)</td>
<td>127.58(42)</td>
<td>54.94(44)</td>
<td>54.56(44)</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>98.02(34)</td>
<td>107.56(42)</td>
<td>45.92(44)</td>
<td>45.88(44)</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>108.94(42)</td>
<td>120.46(42)</td>
<td>45.14(44)</td>
<td>44.50(44)</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>73.32(40)</td>
<td>91.48(44)</td>
<td>44.04(44)</td>
<td>44.54(44)</td>
</tr>
</tbody>
</table>
Table 7. Amount of memory (after learning a problem in Experiment 2)

<table>
<thead>
<tr>
<th>dim</th>
<th>ϵ</th>
<th>Q-Learning</th>
<th>CFRL</th>
<th>SFRL</th>
<th>MVNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.1</td>
<td>4(double) × 20 × 20</td>
<td>2(int) × 124</td>
<td>2(int) × 46</td>
<td>2(int) × 45</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>4(double) × 20 × 20</td>
<td>2(int) × 125</td>
<td>2(int) × 55</td>
<td>2(int) × 54</td>
</tr>
<tr>
<td>2</td>
<td>0.4</td>
<td>4(double) × 20 × 20</td>
<td>2(int) × 139</td>
<td>2(int) × 45</td>
<td>2(int) × 44</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>4(double) × 20 × 20</td>
<td>2(int) × 275</td>
<td>2(int) × 44</td>
<td>2(int) × 45</td>
</tr>
</tbody>
</table>

6.3. Discussion

We accomplish the aim which is to make memory smaller and reduce learning time in SFRL and MVNC. The results implied that the performance of SFRL is equal to or better than that of Q-learning and CFRL in Experiment 1. However, SFRL agents do not work well in Experiment 2. On the other hand, the performance of MVNC is equal to or greater than that of Q-learning and CFRL in Experiments 1 and 2.

In Mean learning processes of SFRL and MVNC, if ϵ is large (Figure 12, 17 and 18), it appears that SFRL and MVNC agents do not work well. However, this is no problem because results showed by Table 4 and 6 are not meager.

7. Conclusion

We developed Chain Form Reinforcement Learning (CFRL) in terms of Contextual Learning and called this method Sneak Form Reinforcement Learning (SFRL). The aim of SFRL is to make memory smaller and reduce learning time. If a sequence of state-action pairs has a shortest path, a SFRL agent cuts and saves the path. End of an episode, the memory table has only the most appropriate path. To improve the performance, we introduce Majority Vote of Neighborhood Conditions for SFRL and call this method MVNC. Majority Vote of Neighborhood Conditions is the rule which agent in an unknown state selects an action with circumjacent information.

Our methods were compared to Q-learning and CFRL in some easy simulations. We examined performance and discussed the best usage environment. We accomplished our aim which is to make memory smaller and reduce learning time in SFRL and MVNC. However, performance of SFRL is not as good as those of Q-learning and CFRL. In MVNC, the performance is equal to or greater than that of Q-learning and CFRL. Therefore, Introduction of Majority Vote of Neighborhood Conditions for SFRL is effective.

For the future work, we make memory much smaller and make MVNC responsive to continuous states and actions.

References