Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 9, Number 1, pages 69–73, January 2020

Development of AlphaZero-based Reinforcment Learning Algorithm for Solving Partially Observable Markov Decision Process (POMDP) Problem

Tomoaki Kimura
Engineering department,
The University of Electro-Communications
Tokyo, Japan

Katsuyoshi Sakamoto
Engineering deptpartment,
The University of Electro-Communications
Tokyo, Japan

Tomah Sogabe *
i-PERC &
Engineering department,
The University of Electro-Communications
Tokyo, Japan;
Grid, Inc.
Tokyo, Japan
sogabe@uec.ac.jp

***Abstract*—In recent years deep reinforcement learning (DRL) methods have advances rapidly, so DRL is applied in many fields. Most of DRL algorithms assume that the information from the environment is perfectly observed. However, in many real problems, the information from the environment is not fully observed. Such a problem is treated as a Partially Observable Markov Decision Processes (POMDPs). So, algorithms that solve POMDPs are important in applying DRL to real-world. In this paper, we apply AlphaZero, a deep reinforcement learning algorithm that achieve great performance in game, to POMDPs and show that the algorithm may be effective for POMDPs using a partially observable maze problem.**

Keywords—reinforcement learning, AlphaZero, POMDPs, maze problem

I. Introduction

With the development of deep reinforcement learning (DRL) in recent years, DRL has begun to be applied to many fields such as robot control and games. The algorithm AlphaGo[1] announced by Google Deep Mind in 2016 defeated the top Go player Lee Sedor, and the successor algorithm AlphaZero[2] fought 100 times against AlphaGo and won 100times. Thus, the deep reinforcement learning algorithm is progressing rapidly.

Many famous deep reinforcement learning algorithms like DQN[3] assume that the state is fully observed. This is not a problem in an ideal environment such as a simulation, but is a problem in real-world environments because accurate information is often not available. A problem with incomplete state obtained from an environment is treated as a Partially Observable Markov Decision Processes (POMDPs). Since real-world problems are often this POMDPs, it is important to develop an algorithm that can solve this problem.

Therefore, in this study, we developed an algorithm that solves POMDPs by extending the DRL algorithm AlphaZero, which shows overwhelming performance in games Go, to be applicable under POMDPs and verified its effectiveness using a partially observable maze problem.

II. Background

1. POMDPs

In the POMDPs, the agent cannot fully observe the state, but instead receives an observation from the environment.

The initial state is determined from the initial state distribution. When the agent acts a in the environment, the state transitions from s to s' according to the transition probability $Pr(s'|s,a)$, receives an observation o from the environment according to the observation probability $Pr(o|s',a)$, and receives the reward r according to the reward function $Pr(r|s',a)$. The history h_t is the time series of actions and observations and is expressed as $h_t = \{a_1, o_1, a_2, o_2, \ldots, a_t, o_t\}$. The agent's policy is denoted $\pi(h,a) = Pr(a|h)$. The purpose of the agent is to learn a policy π^* that maximizes the expected total reward $\mathbb{E}_\pi[\sum_{i=t}^{T} \gamma^{i-t} r_i]$.

The belief state B is denoted $B(s,h) = Pr(s|h)$, it can be updated by the following formula.

$$B(s', hao) = \frac{\sum_s Pr(o|s',a) Pr(s'|s,a) B(s,h)}{\sum_s \sum_{s''} Pr(o|s'',a) Pr(s''|s,a)\, B(s,h)}$$

2. AlphaZero

In Alpha Zero, the policy is determined by MCTS. The Neural Network (NN) combined the policy network and the value network into a single network, unlike AlphaGo. By making predictions using NN in MCTS, the rollout for getting a reward that was necessary in the AlphaGo has become unnecessary.

The flow of training begins with self-play by executing an action according to the policy computed by MCTS. When the self-play is completed and a reward is obtained, each state experienced by the self-play, the policy at that time, and the reward are stored and used for training. The policy is trained with the cross-entropy error and the value is updated with the mean squared error.

III. AlphaZero for POMDPs

AlphaZero is an algorithm that can be used only when the state can be completely observed, so it is difficult to apply it to the problem of the POMDPs. Therefore, we made the following extensions.

1. Dealing with time series in the NN
2. Treating POMDPs in the MCTS

The training pipeline is shown in Algorithm 1. The architecture of NN and MCTS flow are explained below.

1. Neural Network Architecture

In the POMDPs, the policy π is denoted as a function of history h such as $\pi(h,a) = Pr(a|h)$. Therefore, when the policy is expressed in NN, NN must treat history h, which is time series of actions and observations. So, with reference to the method[4], we use LSTM to deal with time series and express history h, and the expressed h is processed by FNN to obtain policy p and value v. In the training, the policy p is updated by the cross-entropy error with a policy $\pi(h,a)$ obtained by MCTS and the value v is updated by the mean squared error with a reward z obtained at the end of an episode (Figure 1).

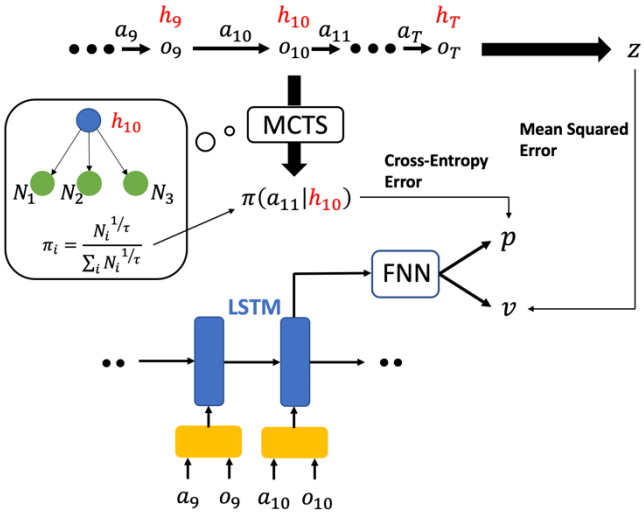

Figure 1: Neural Network training procedure

2. Partially Observable MCTS

We extended MCTS in AlphaZero to treat POMDPs with reference to the method[5] which expanded MCTS to deal with POMDPs. Figure 2 shows MCTS pipeline.

In the Select, a state is first sampled from the belief state of the root node. The action with the largest UCB1 is selected, and the process proceeds to the next node. Then, from the sampled state and this action, a state transitioned and an observation are received using the simulator, then process proceeds to the next node. The above processes are repeated using the transitioned state and the tree is searched. At this time, the state received from the simulator (s_3 in the Figure 2) is saved.

In the Expand, a leaf node is expanded.

In the Evaluate, the expanded node is evaluated using NN. Then, p calculated by NN are stored and a hidden state and a memory state of LSTM are stored on the expanded node.

In the Backup, the nodes that have been visited are traced back from the expanded leaf node to the root node, and the node value is updated using v calculated by NN. At this time, the states saved at the time of Select (s_3 in the Figure 2) are added to the belief state of the nodes.

The above four processes are repeated to finally find a policy proportional to $N^{1/\tau}$. N is the number of visits from the root node to each action node.

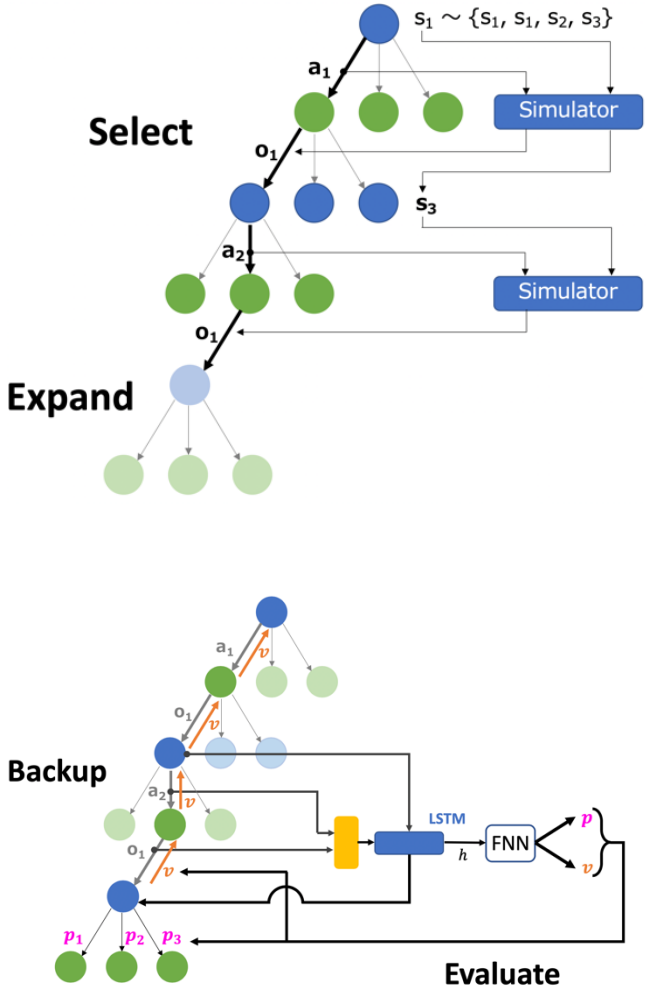

Figure 2: MCTS in POMDPs

Algorithm 1

Initialize NN parameter θ, θ' and replay memory D
For episode = 1 to N do
  Initialize MCTS and episode batch D_E, $a_0 = a_{\text{init}}$, $h_0 = h_{\text{init}}$
  Reset Environment, obtain rsulting o_0
  Reset MCTS with o_0
  While not done do
    Calculate MCTS, obtain $\pi(a_{t+1} \mid h_t)$
    Execute action $a_{t+1} \sim \pi(a_{t+1} \mid h_t)$, obtain resulting o_{t+1}
    Store $(a_t, o_t, \pi(a_{t+1} \mid h_t))$ in D_E
  Obtain resulting reward z
  Store sequences $< a_j, o_j, \pi(a_{j+1} \mid h_j), z >$ in D using D_E and z
  $\theta' \leftarrow \theta$
  For train = 1 to T do
    Sample a minibatch B from D
    Optimize θ with Loss and B
  End For
  If ScoreCompare(θ, θ') < 0 then
    $\theta \leftarrow \theta'$
End For

IV. Experiments

In order to confirm that the proposed method can learn, we performed a benchmark test using a partially observable maze problem. In addition, we performed a comparative experiment with different observation probabilities.

1. Partially Observable maze problem

A partially observed maze problem (Figure 3(left)) was designed as a POMDPs environment. The problem is that the agent goes from the start S in the lower left (1, 1) to the goal G in the upper right (8, 8), but the agent cannot know exactly where it's position is. The state was the position of the agent, and the observation that the agent obtains was generated according to the observation probability which is showed in (Figure 3(right)). Any of the states, the bottom, the left, the top, and the right was selected according to their probabilities and became the observation. At each step, the agent moves to the right, top, left, bottom, or as-is depending on the action. Each Episode ended when the agent reached a black trap by movement. The maximum number of steps per episode was 30. The reward was given only when the agent reached goal G, according to the number of steps to reach goal.

$$r_t = \begin{cases} {}^{t_{min}}/_{t} & s_t = G \\ 0 & else \end{cases}$$

t_{min} is the minimum number of steps required to reach goal, and in this case it is 14.

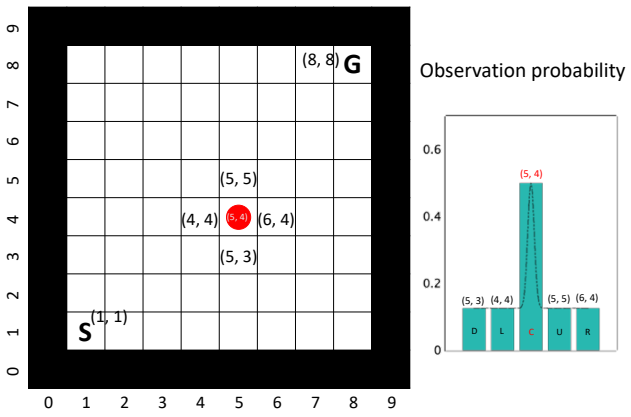

Figure 3: (left)Partially observable maze problem (right)Observation probability

Figure 4 shows the result of learning maze problem by proposed method. It can be confirmed that the proposed method can actually learn.

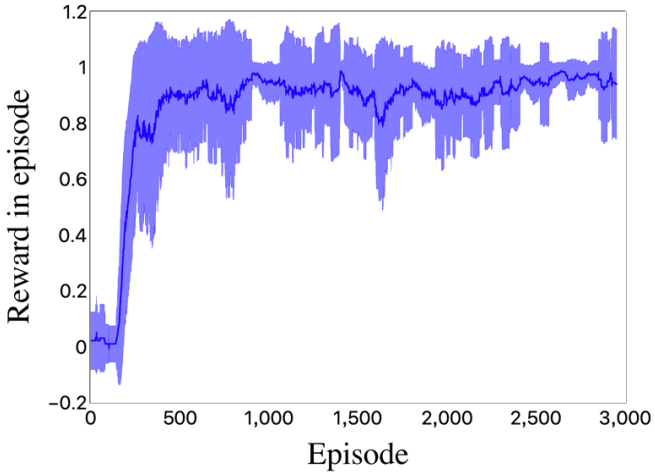

Figure 4: reward curve in maze problem

2. Comparative experiment

We changed the observation probability that is showed in (Figure 3(right)) and performed a comparative experiment with three types of observation probability as shown in Figure 5.

Figure 6 shows the result of comparative experiment. It can be seen that proposed method can learn for three types observation probability. Comparing the three types, it can be seen that the higher the probability of observing true state, that is, the closer to fully observation, the faster the reward rises, and the more stable in higher reward.

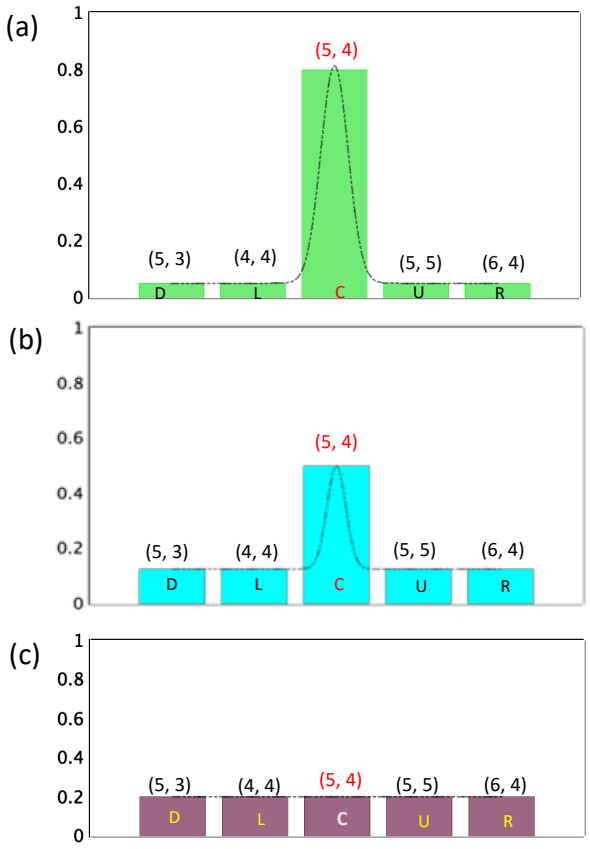

Figure 5: different types of observation probability

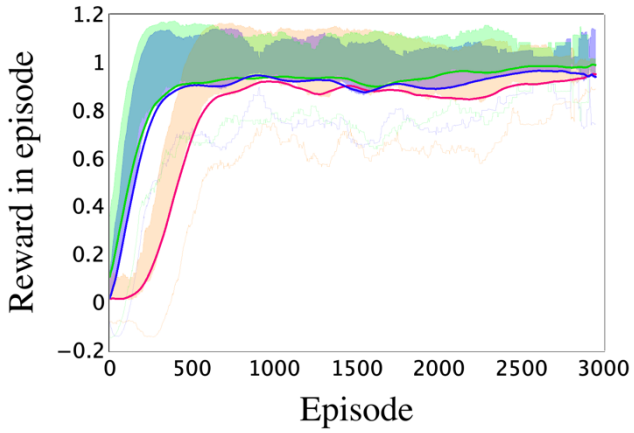

Figure 6: reward curve in comparative experiment

Figure 7 shows movements of an agent in an episode using learned parameters (red arrow) and movements of observations at that time (blue arrow) with three types of observations shown in Figure 5. Even if the observation is different from the true state, it can be confirmed that the agent can go to goal.

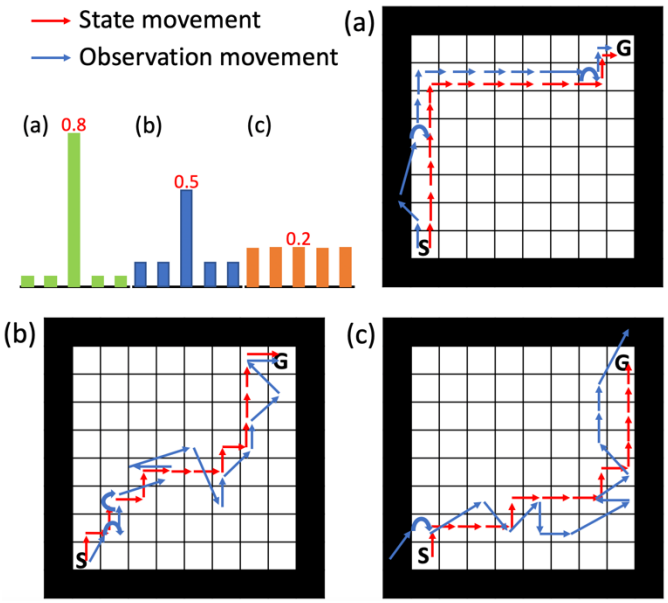

Figure 7: state and observation movement with three types of observation probability

V. Conclusion

In this study, we proposed a method to extend the deep reinforcement learning algorithm AlphaZero, which assumes complete observation, to be applicable to the problem of POMDPs. The proposed method was applied to the partially observable maze problem, and it was confirmed that proposed method can learn. In addition, we performed a comparative experiment with changing the observation probability of the problem, and confirmed that proposed method can learn at three types observation probability, but more stable learning was possible as it approached complete observation.

As future prospects, we will confirm whether this method can be applied to more complex problems which have many state spaces and observation spaces. In addition, we will compare proposed algorithm with other algorithms for POMDPs.

VI. Reference

[1] D Silver, A Huang, C Maddison, A Guez, L Sifre, G Driessche, J Schrittwieser, I Antonoglou, V Panneershelvam, M Lanctot, S Dieleman, D Grewe, J Nham, N Kalchbrenner, I Sutskever, T Lillicrap, M Leach, K Kavukcuoglu, T Grapel, D Hassabis, “Mastering the game of Go with deep neural networks and tree search”, Nature, 2016

[2] D Silver, J Schrittwieser, K Simonyan, I Antonoglou, A Huang, A Guez, T Hubert, L Baker, M Lai, A Bolton, Y Chen, T Lillicrap, Fan Hui, L Sifre, G van den Driessche, T Graepel, D Hassabis, “Mastering the game of Go without human knowledge”, Nature, 2016

[3] V Mnih, K Kavukcuoglu, D Silver, A Rusu, J Veness, M Bellemare, A Graves, M Riedmiller, A Fidjeland, G Ostrovski, S Petersen, C Beattie, A Sadik, I Antonoglou, H King, D Kumaran, D Wierstra, S Legg, D Hassabis,

"Human-level control through deep reinforcement learning", Nature, 2015

[4] P Zhu, X Li, P Poupart, G Miao, "On Improving Deep Reinforcement Learning for POMDPs", arXiv: 1704.07978v6, 2018

[5] D Silver, J Veness, "Monte-Carlo Planning in Large POMDPs", in *Neural Information Processing Systems*, 2010