Deep Learning & Practice - Lab 5
2048 - Temporal Difference Learning

0660836, Dhananjaya Panga

August 28, 2019

1 Introduction

Reinforcement learning (RL) is an area of machine learning involving software agents to take actions based on the environment in order to maximize the cumulative reward. RL focuses on finding the balance between exploration and exploitation. Temporal difference is more kind of exploration in which the agent learns from the environment without having a prior knowledge of the environment. This TD learning is a model-free and unsupervised learning in nature.

In this lab, I will implement reinforcement learning with temporal difference learning on 2048 game. It will compare between the different state of state value and after state value and will throw some light on strategy to gain performance.

2 Temporal Difference (TD) Learning

2.1 Reinforcement Learning

In reinforcement learning, the agent has to decide how to act and perform its task in the absence of prior training data. The agent learns from the experience. It collects training examples through trial-and-error and attempts its task to maximize the long-term reward. The major components of the reinforcement learning are agent, environment and reward. The main goal of reinforcement learning is to find optimal policy and maximize the cumulative rewards.

Almost all reinforcement learning problem can be formulated as an Markov Decision Process (MDP). A MDP is a tuple of $< S, A, P, R, \gamma >$, where S is the set of all states, A is the set of all action, P is the state transition probability, R is the reward function mapping the state to real value, and γ is the discount factor. The Markov property state that the future state is independent ot the past given the current state

$$p(s_{t+1}|s_t = p(s_{t+1}|s_1, ..., s_t)$$

2.2 TD(0) Algorithm

The value function, $V(S)$ is the long term value of the state s. Once we learn the value function of all state, we learn the policy by taking action toward the state with max state value. One can solve the value function using *Dynamic Programming* but unfortunately we do not know the environment in real application problems or the dynamic programming is too slow when the number of state is too big.

Temporal Difference learning is one of the model-free approach. The long-term reward $V(S_t)$ is divided into two parts. One is immediate reward R_{t+1} and another is discounted future reward $\gamma V(S_{t+1})$. The learning target is to minimize the temporal difference between each adjacent time-step. The TD target is

$$R_{t+1} + \gamma V(S_{t+1})$$

TD learning has lower variance an dis more efficient than MC learning on the problems that exploit Markov property.

2.3 TD(1) Algorithm

TD(1) makes an update to the values after the end of the episode. The discounted sum of reward G_t can be described as

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

The environment keeps track of cumulative reward and sums up together with the discount factor γ and R_{t+1} is the reward at time-step $t+1$, and the successive rewards are discounted with the reward at appropriate time-step. The cumulative discounted error is subtracted from the prior estimate $V(S_t)$ known as TD error.

$$\text{TD } Error : G_t - V(S_t)$$

Finally, the TD *Error* is adjusted by term α to estimate the amount of error we want to update and added with out previous estimate.

$$Update\ value : V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

2.4 Before State Value

In 2048 game, after sliding teh board, the board state will change based on the action and randomly generate another tile which is also another state transition.
It can be formulated as

$$s_t^{before} \xrightarrow{\text{action}} s_t^{after} \xrightarrow{\text{generate}} s_{t+1}^{before} \xrightarrow{\text{action}} ...$$

Thus, we have two candidate of the estimation of value function. The before state values estimate the long-term value of the board state before action. The next before state s_{t+1}^{before} is unknown to choose the optimal action based on action reward and the estimated value of discounted future reward $\gamma V(s_{t+1}^{before})$.

The learning of before state value can be described in following steps. First, the game is played based on the policy that pick the action that has maximal expected action value. The action value of a state s and action a is

$$r + \sum_{s"\epsilon S} p(s"|s')V(s")$$

where s' si after state of takin gaction and $s"$ is the next before state. The episode of the game is recorded in the form of $(s, , ar, s', s")$. Finally the estimate is updates before state value after the end of the episode to the beginning by

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

2.5 After State Value

Another option of value function is the after state value, which estimate the long-term value of the board state after action (before randomly generate tile). In this case, the action value could be evaluated without the uncertainty because it's independent to the random generation of tiles, so it is total model-free since we don't have to know the transition probability. The learning of after state value can be described in following steps. First, we play game based on the policy that pick the action that has maximal expected action value. the action value of a state s and action a is

$$r + V(s^{'})$$

Second, we record the episode of the game with the form $(s, a, r, s', s")$. Finally, we update the estimated before state value form the end of episode to the beginning by

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

where G_t is $R_{t+1} + \gamma V(S_{t+1})$ and s_{t+1} is the after state of s_t by taking the optimal action according to the current estimated value function and R_{t+1} is the reward of that action.

3 Implementation

3.1 n-tuple network

n-tuple is a sequence $[a_{i0}, ..., a_{in-1}]$ of n different board game locations where each a_{ij} codes a specific cell of the board. Each location possesses one of P possible states $z[a_{ij}] \in 0, ..., P - 1$. It depicts an n-tuple of length n thus has P^n possible states $k \in 0, ..., P^n - 1$. we have used 4*6 tuples with all possible isomorphisms

The main advantages is to include the simplicity and capability of realizing non-linear mappings to spaces of higher dimensionality. The low dimensional board is projected into high dimensional sample space by the n-tuple indexing process.

Temporal difference learning is an agent learning from an environment without any prior knowledge of the environment.

Learning rate kept at 0.1 and trained for 1 million episodes.

4 Experimental Results

The training ran for 32 hours on Intel Xeon CPU. A plot showing episode scores of 100,000 training episodes.

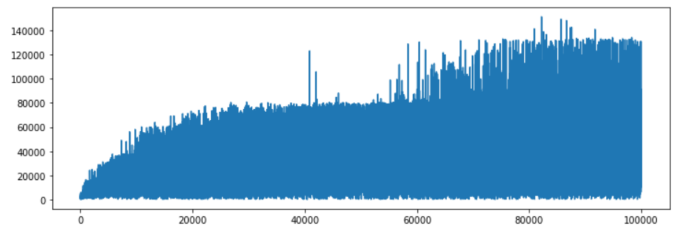

Figure 1: Plot of 1M episodes

4.1 Agent Performance

The agent played 2048 with 1000 times.

Tile	Win Rate	End Rate
128	99.9	0.1
256	99.8	1.4
512	98.4	2.4
1024	96	5.3
2048	90.7	13.7
4096	77	75.4
8192	1.6	1.6

The 2048-tile win rate in 1000 games is 90.7%.

5 Discussion

5.1 TD-Update Bootstrapping

TD-update bootstrapping is performed to use more than one estimated values in the update step for the same kind of estimated value. But the main disadvantage of using bootstrapping is that it tends to bias to the initiated starting values. This tends to destabilize the update system of the agent because of relying on self reliance rather than learning.

5.2 Training - on-policy or off-policy

- Off-policy learning: The policy (π) improvement through the experience of another policy (μ)
- On-policy learning: The policy (π) improvement is from the sampled experience from the same policy (π)

Since, we are using Q-network, we use off-policy for our training purpose.

5.3 Improvements

We have not employed any explicit exploitation mechanism during the learning. Since the environment is stochastic, it provides wide variety of experience. It is also little bit tough to obtain the entire state transition. If one can include the entire state transition, we may get much better accuracy at 4096 and 8192 tile.