Midterm Report

Dongqiangzi Ye
dye25@wisc.edu

Haozhen Wu
hwu84@wisc.edu

Jennifer Cao
qcao44@wisc.edu

1 Introduction

As Reinforcement Learning combined with deep neural networks making amazing achievements recently, we explore such applications in Computer Vision. Open AI Gym contains a bunch of Reinforcement Learning environments with a common interface, including some interesting Computer Vision scenarios, provides a simple way to explore these applications.

Reinforcement Learning is a kind of goal-oriented algorithms, which can reward a prediction step by step instead of immediately giving the outcomes at once. That is, Reinforcement learning is dynamic and the prediction will be based on agent state and environment state. For example, consider maximizing the points won in a game over multiple moves. In Reinforcement Learning Model, they can start from a blank slate, and then achieve superhuman performance.

Reinforcement learning in computer vision is a task that treats images as inputs and optimizes final game scores. Instead of using Q-table to find actions for each time step, we apply deep neural networks, where parameters are far less than Q-table, to map states (images) to values (value function) or map states to policy (policy function). In our project, we will analyze different methods in OpenAi Gym Atari, such as DQN, double DQN, policy gradient, actor-critic and so all.

2 Methods

Reinforcement Learning is a task which an agent interacts with an environment with a sequence of actions, observations and rewards. More specifically, an agent learns to optimize a policy π thats maps a sequence of observation $h_t = (s_0, a_0, x_1, ..., a_{t-1}, x_t)$ to next action a_t. These sequences is Markov decision process (MDP) that each sequence is a distinct state. Existing reinforcement learning methods can be categorized into value-based techniques, policy-based techniques and actor-critics. In this project, we will try these three approaches correspondingly.

2.1 Value-Based Methods

Q-Learning[2] and Deep Q-Learning[1] are basic concepts for reinforcement learning. They update Q-value (Q-table) for each action based on different states. Q-value is defined as this:

$$Q^*(s,a) = E_{s'}[r + \gamma \max_{a'} Q^*(s^{'}, a^{'})|s, a] \tag{1}$$

Where $Q^*(s,a)$ is an action-value function whose input is a image (state) and output are rewards for each action.

We use following equation to update $Q^*(s,a)$:

$$\bigtriangledown\theta = E_{s,a}[(r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta) - Q(s,a;\theta)) \bigtriangledown Q(s,a;\theta)] \tag{2}$$

The first term $r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta)$ is next Q-value and the second term $Q(s,a;\theta)$ is present Q-value. So this error means Q-value is expected to be convergent.

[1] used a data-set D to store the agent's experiences, then sample minibatch transitions (like SGD) from D to update θ.

As we can see from (2), the first term and the second term use the same parameters to calculate the error. We call the first term as Q-target and second term as Q-current. If we use the same parameters θ to estimate these two terms, there is a big correlation between errors and the parameters. To solve this problem we fix the parameters for Q-target as θ^-, then the first term becomes:

$$Q - target = r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta^-) \tag{3}$$

Where we will update the θ^- after some time steps.

Double DQN [3] handles the problem of the overestimation of Q-values. This problems is caused by (3). Since we use θ^- to chose action, this can lead to false positive. So we modify (3):

$$Q - target = r + \gamma Q(s^{'}, \underset{a}{argmax}\, Q(s,a;\theta); \theta^-) \tag{4}$$

Where we use current Q-values to choose actions.

2.2 Policy-Based Methods

There are several flaws in value-based methods. At first, they can have a big oscillation while training since small changes of the parameter will change choices of actions dramatically. Second they have poor performance if the action space is high dimensional or even continuous. Third they can not handle stochastic policies. Instead of learning value function, policy-based methods [4,5] learn policy function directly which maps states to actions. Policy function is defined as this:

$$\pi_\theta(a|s) = P(a|s)$$

And we want to use a score function to measure the policy function:

$$J(\theta) = E_{\pi_\theta}(r)$$

There are three methods to calculates the score function.

First if we always start at some state s, we could use this:

$$V(s) = \sum \gamma^k r_k$$

$$J(\theta) = E_{\pi_\theta}(V(s))$$

Second if we calculate the score function in a continuous environment, we could use this:

$$J(\theta) = \sum d(s)V(s)$$

Where $d(s)$ is the distribution of each state.

Third we can use the average reward per time step:

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

Then we can update the parameters by this:

$$\begin{aligned} \bigtriangledown_\theta J(\theta) &= \sum_s d(s) \sum_a \bigtriangledown_\theta \pi_\theta(s,a) R(s,a) \\ &= \sum_s d(s) \sum_a \pi_\theta(s,a) \bigtriangledown_\theta log[\pi_\theta(s,a)] R(s,a) \\ &= E_{\pi_\theta}(\bigtriangledown_\theta log[\pi_\theta(s,a)] r) \end{aligned}$$

By Stochastic Gradient Ascent, we can remove the expectation term:

$$\theta = \theta + lr * \bigtriangledown_\theta log[\pi_\theta(s_t, a_t)] r_t \tag{5}$$

2.3 Actor-Critics

The policy-based methods [6] have a problem that they would update the parameters after the whole episode and the value-based methods also have some other problems. Actor-critics is a hybrid method to solve these problem. Its idea is use two neural networks to represent value function and policy function. Then it will update one parameters with other neural network. We can modify the (2) and (5).

$$\theta_p = \theta_p + lr_p * \bigtriangledown_{\theta_p} log[\pi_{\theta_p}(s_t, a_t)] Q(s_t, a_t; \theta_v) \tag{6}$$

$$\theta_v = \theta_v - lr_v * [r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \bigtriangledown_{\theta_v} Q(s_t, a_t)$$

Where the action a_{t+1} is generated by policy function and the reward $Q(s_t, a_t; \theta_v)$ is generated by value function $Q(s_t, a_t; \theta_v)$ in (6).

We can define this Advantage function to replace the value function:

$$\begin{aligned} A(s,a) &= Q(s,a) - V(s) \\ &= r + \gamma V(s`) - V(s) \end{aligned} \tag{7}$$

Where (7) is obtained by the loss of value function.

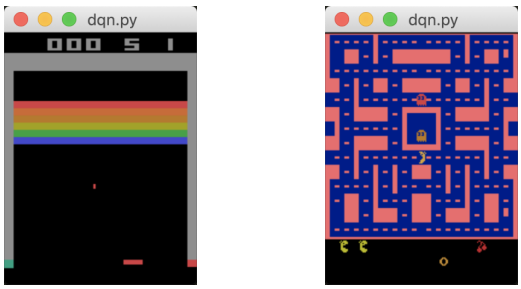
However hyperparameters for reinforcement learning is hard to decided and the training process is much unstable. The state-of-art algorithm PPO solve this problem by considering another loss function:

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

$$L^{CLIP} = E_t(min(r_t A_t, clip(r_t, 1-\epsilon, 1+\epsilon) A_t))$$

3 Current Result

Now, our team have tried some Atari examples, DQN and double DQN. And we read some related readings which we discussed above. Figure [1] shows our OpenAi Gym sample results.



(a) BreakOut (b) MsPacman

Figure 1: OpenAi Gym Sample Results

4 Future Work

Next, we will continue implement all methods we mentioned and comparing their performance and characteristics. Then we will do a webpage to show our final results and some game video by our reinforcement learning algorithms. Also we will prepare our final presentation.

5 References

1 Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

2 Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

3 Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Thirtieth AAAI Conference on Artificial Intelligence. 2016.

4 Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992): 229-256.

5 Levine, Sergey, and Vladlen Koltun. "Guided policy search." International Conference on Machine Learning. 2013.

6 Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. 2016.