Offline Reinforcement Learning

Sequential Decision-Making

Member:

- Duo Xu <dxx128>

1. Introduction

In this paper, I begin by introducing the offline reinforcement learning (RL) which also called Batch reinforcement learning. Compared to the online RL, I try to use a batch of samples to generate experience in the environment and use the fixed set of experience to optimize a policy. To be more specific, these samples can be represented by state-action-reward-state pairs. In my opinion, the best difference between online RL and offline RL is that when we utilize offline RL, the current policy doesn't generate data. And least-squares policy iteration (LSPI), the algorithm we will explain later and implement, is a kind of offline RL algorithm.

LSPI is based on least-squares temporal-difference (LSTD) learning algorithm (Bradtke and Barto, 1996) [1]. Compared to the traditional temporal difference (TD) algorithm, LSTD algorithm can capture more useful information from the experience and improve the validity of the data even though each time step will cost larger calculation time. However, LSTD algorithm can only solve predict problems instead of control problems. So Lagoudakis [2] integrated LSTD with Q function and proposed LSPI algorithm which could be used to solve control problems (problems where a set of actions or a good policy will be learned).

As a standard offline RL approach, LSPI algorithm aims to solve control problems. Notice that the main idea of LSPI algorithm is to use a batch of samples collected in advance to learn and evaluate the current policy. In other words, LSPI algorithm is a wrapper around LSTDQ algorithm, and we can learn a Q-function for current policy given the batch of data, which means we don't need to collect samples from current policy. Because LSPI algorithm does not require an approximate policy representation, it is stable and efficient and doesn't diverge or give meaningless answers.

This paper is organized as follows: First, I specifically describe the LSTDQ (Section 2) and the process of the LSTD algorithm (Section 3); then I implement this algorithm into a game called "carpole" from the openai gym environment (Section 4); next I compare this algorithm with other RL method like Deep Q-learning algorithm which is a very common approach in RL (Section 5).

2. Least-Squares Temporal-Difference Learning for Q-function

As previously mentioned, the paper details a method to learn a Q-function for current policy given a batch of data. We assume that there are k linearly independent basis functions in the linear architecture, and a sample (s, a, r, s') is used to learn a Q-function, so the samples set D could be defined as

$$D = \{(s_i, a_i, r_i, s_i')|i = 1,2, \dots, L\}.$$

And LSTDQ uses a linear Q-function with features Φ_k and weights w_k. So, the Q-function could be represented as

$$\hat{Q}_w(s, a) = \sum_k w_k \Phi_k(s, a)$$

In this way, we could define greedy policy:

$$\pi_w(s) = \operatorname{argmax}_a \hat{Q}_w(s, a)$$

Notice that if the action space A is very large or continuous, it is impossible for us to utilize this strategy, so in my experiments, I use an environment with only two discrete action space.
For each sample (s, a, r, s') in the sample set,

$$\tilde{A} \leftarrow \tilde{A} + \varphi(s,a)(\varphi(s,a) - \gamma\varphi(s', \pi(s')))^T$$

Here we need to clarify that A is the sum of many rank one matrices of the form, and b is the sum of many vectors of the form which could be defined as:

$$\tilde{b} \leftarrow \tilde{b} + \varphi(s,a)r$$

Therefore, the learn parameters $\widetilde{w}^{\pi}$could be yielded by the learn linear system:

$$Aw^{\pi} = b$$

Which w^{π}could be represented as:

$$w^{\pi} = \tilde{A}^{-1}\tilde{b}$$

Finally, LSTDQ will output the parameters w^{π} of the approximate value function of the policy π and the iteration continues in the same manner.

3. Least-Squares Policy Iteration

Now, I need to specifically dive into the main idea and the process behind the LSPI algorithm. This algorithm can wrap up the process of LSTDQ so that at each iteration, a different policy is evaluated and certain sets of basis functions may be more appropriate than others for representing the state-action value function for each of these policies.

I detail the process of LSPI based on the matlab code provided by the author of the paper, and I implement it in the python language, here are the specific process of LSPI algorithm:

1. We generate a database of (s, a, r, s') experiences
2. Then we start with random weights and random policy, which could be achieve by some function of the gym environment.
3. We repeat the following process:
    a) We valuate current policy against database by running LSTDQ to generate new set of weights and these new weights can imply new Q-function and hence new policy
    b) Then we replace current weights with new weights
    c) The process stops until convergence

Notice that the offline property of LSPI algorithm could be represented by the fact that sampling distribution can be controlled and samples can be reused and generated randomly rather than by current policy.

4. Experiments

In my experiments, I utilize a domain from the gym environment. The domain is entitled “Cart Pole” with version “V0”. In this domain, A pole is attached to a trolley by a powerless joint, and the trolley moves along a frictionless track. The pendulum is placed upright on the trolley, and we need to balance the pole by applying forces in the left and right directions on the trolley, as shown in Figure 1 below.

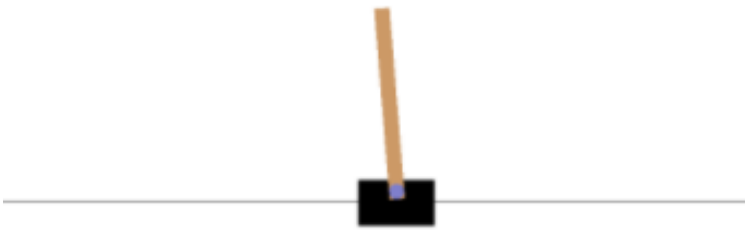

Figure1 CartPole-V0

Here we need to clarify some details of the data collection or the sample space. In this domain, our action values only take two values (0,1). “0” means we push the cart to the left while “1” means we push the cart to the right. Besides, the observation space is shown in an array taking 4 values (0,1,2,3), and “0” here means the cart position, “1” contains the velocity of the cart, “2” represent the pole and “3” means the angular velocity of the pole. Since the goal is to keep the bar upright for as long as possible, a +1 reward is assigned to each step taken, including the termination step. When some conditions are met, the cart will stop which means the episode terminates. These conditions are: 1. If pole angles is greater than $\pm 12^\circ$ 2. The position of the cart is greater than $\pm 24^\circ$ 3. The length of episode is greater than 200 in this version of environment.

When I implement LSPI algorithm into this domain, our inputs of LSPI algorithm are the samples, basis functions, some parameters and the initial weight of matrix and our output is the matrix of weight after convergence. To be more specific, the samples can be shown in the format of tuple (s, a, r, s') and basis functions are in the format of list. Some parameters including the gamma (the discount factor) and the epsilon (the stopping criterion) are all float types. In our experiment, I set gamma with the value 0.95, and the epsilon with the value 0.01, and I use 1000 trail samples and for each sample (s, a, r, s') we will get a matrix of weight after convergence through LSPI algorithm function. By calculating this weight, I get an action and ask the cart to conduct that action.

LSPI algorithm was implemented using Python language and was tested on the cartpole balancing problem, we can watch the video named “LSPI_demo” in our file or run the “LSPI_main.py” file to see the performance of this algorithm. The picture below shows that even though the cart is moving from side to side, it can still balance the pole very well. We can see in the video that just through a few episodes, LSPI algorithm can perform well in the balancing problem.

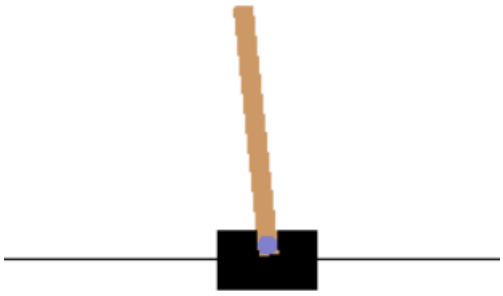

Figure2 CartPole with the LSPI algorithm

Then I compared this algorithm with Deep Q-learning algorithm, the result will be represented in the next section.

5. Extensions

The bad news of LSPI algorithm is that LSPI doesn't address the exploration problem which means it decouples data collection from policy optimization. For example, the source the samples are not from the current policy. This is not a major issue, but can be in some cases. Besides, LSPI is limited to linear functions over a given set of samples.

Our final project was supposed to use some Q-learning methods like Deep Q-Network (DQN) [3] to address the problem, but due to some adjustments of our group member we didn't show the result. Therefore, after considering some concept and the property of the DQN, I try to utilize it to compare with LSPI algorithm.

Based on the drawback of LSPI algorithm mentioned in the first paragraph, I try to use some suitable or fitted Q-iteration methods to perform in the same domains. Some fitted Q-Iteration allows us to use any type of function approximator for the Q-function. Therefore, I choose DQN method to compare.

DQN is a kind of classic online learning approach developed by DeepMind. Its main idea is to integrate some ideas from deep learning like neural network into Q-learning by approximating a state-value function. Based on the concepts from the paper, I detail the process DQN method with my understanding, and the process shows below:

1. We set our sample sets $D = \{(s_i, a_i, r_i, s_i') | i = 1,2, \dots, L\}$ and let D be our batch of transitions.
2. Then we initialize some parameter of neural network to θ.

We repeat the following process:

3. we choose a mini batch of sample B transition from D, which could be defined as $\{(s_k, a_k, r_k, s_k')$
4. Then we update θ for each parameter based on the mini batch we choose, which is

$$\theta \leftarrow \theta + \alpha \sum_{k} (r_k + B \max_{a'} \hat{Q}_\theta(S'_K, a') - \hat{Q}_\theta(s_k, a_k) \nabla_\theta Q(s_k, a_k))$$

In my experiments, I set the episodes 300, which means the number of the iteration is 300 both in LSPI experiment and DQN experiment. If we run the file entitled “DQN_main.py” we can see that after about 200 episodes, the cart can balance the pole very well as shown below. The performance will display more visual when you look at the video or run the codes.

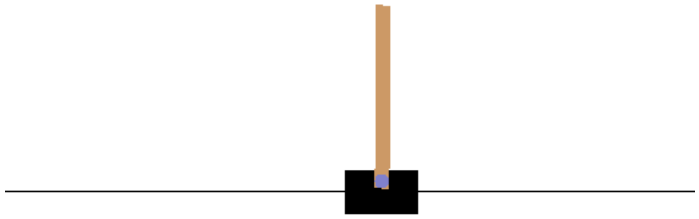

Figure3 CartPole with the DQN algorithm

Notice that these result pictures are similar in the paper, but they are shows great difference when we watch the videos or run the code.
Then I plot the result of both LSPI experiment and DQN experiment, here is the result which could be visualized.

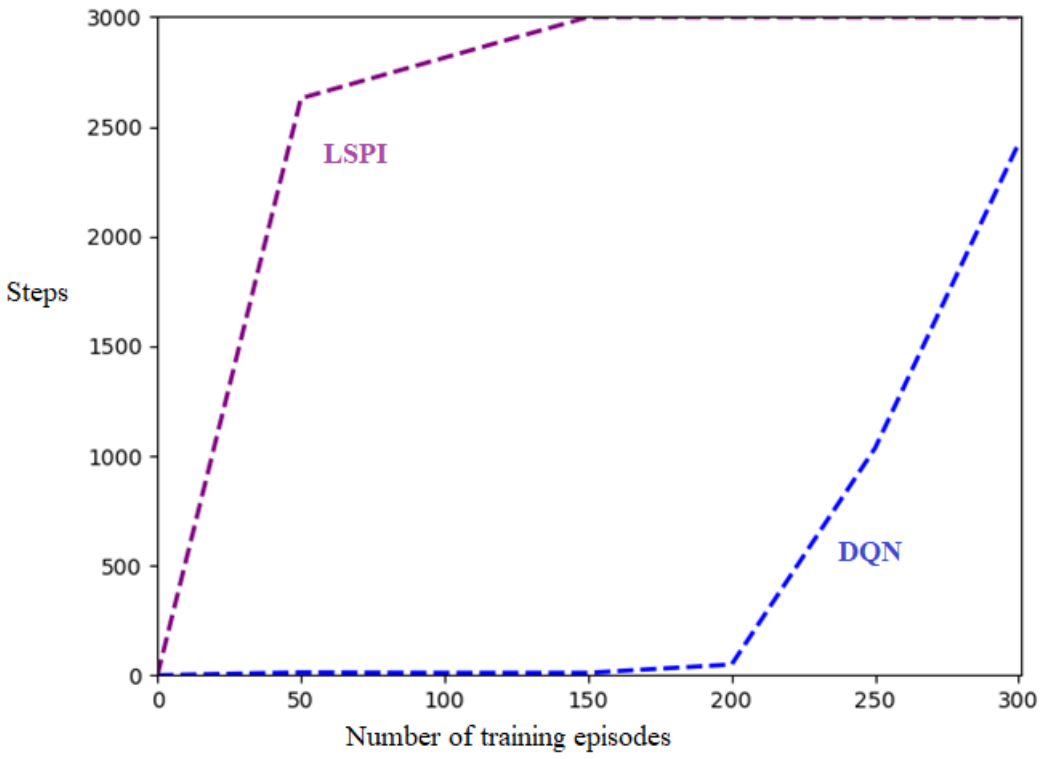

Figure4 the performance of LSPI and DQN

Here the x-axis means the number of training episodes, and the y-axis means the steps that algorithm need to take. The purple dotted line in this picture shows the performance of LSPI algorithm while the blue dotted line shows the performance of DQN algorithm.
The visual results of this comparison can be seen by running the "LSPI_DQN_main.py". The process is time-consuming which will cost several hours for plotting.

Conclusions

In this paper I present the LSPI algorithm and implement it into a classic problem called "cartpole". LSTD produce a value function, LSTDQ produces a Q-function for current policy and LSPI algorithm wrap up LSTDQ to learns the state-value function so that we can select the action without the model.

Besides, I compared the LSPI algorithm against Deep Q-learning algorithm. The result demonstrates the potential of LSPI, which is much more powerful than DQN method. In my experiments I only use small number of samples like 1000 trail samples, but it shows great performance. And the linear approximation architecture makes LSPI algorithm easier to understand rather than Deep Q-learning which has complicated neural network structure.

References

[1] Bradtke, Steven J. and Barto, Andrew G., "Linear Least-Squares Algorithms for Temporal Difference Learning" (1996). Computer Science Department Faculty Publication Series. 9.

[2] Lagoudakis, Michail G., and Ronald Parr. "Least-squares policy iteration." The Journal of Machine Learning Research 4 (2003): 1107-1149.

[3] "Methods and Apparatus for Reinforcement Learning, US Patent #20150100530A1" (PDF). US Patent Office. 9 April 2015. Retrieved 28 July 2018.