2019 Deep Learning & Practice - Lab 6
DQN & DDPG

0660836, Dhananjaya Panga

August 28, 2019

1 Introduction

Reinforcement Learning is about the learning from interaction with the environment. Value-based methods such as Q-learning is to learn the expected return for all possible states and select the action that maximize the expected return. However, if we use tabular form to represent the value function, it will be hard to store and learn a huge table for large MDPs and the tabular approach can not generalize to unseen states which will make the learning process inefficient. Thus, using value function approximation is more appropriate choice for large MDPs. From the recent developments in deep learning, it is common to use neural network to approximate the value function.

In this lab, I will implement two famous deep reinforcement learning models. One is Deep Q-Network (DQN) and another is Deep Deterministic Policy Gradient (DDPG). The DQN is trained with discrete-action game $CartPole - v1$ and DDPG is trained with continuous-action game $Pendulum - v0$.

2 Deep Q-Network (DQN)

2.1 Neural Network as a function approximator

DQN is a value-based method to estimate the action value (Q-value) for a state action pair (s, a) given a policy π. As mentioned above, the traditional tabular form to record the Q value is inefficient for large MDPs. Thus, the neural network is used as a function approximator in DQN.

To train the neural network, we can use the Q-learning target to compute the gradient and update hte model parameter using SGD.

$$target = R(s, a, s^{'}) + \gamma \max_{a} Q(s^{'}, a^{'}|w)$$

$$L(w) = \mathbb{E}[(target - Q(s, a|w))^2]$$
$$\frac{\partial L(w)}{\partial w} = \mathbb{E}(target - Q(s, a|w)) \frac{\partial Q(s,a|w)}{\partial w}$$

Unfortunately, this approach will face two problems. DQN proposes to solve these two problems

2.2 Experience Replay

One problem is that if we train the network with online learning, the training data is not independent to each other in an episode because the state and action in an episode is largely affected by the current policy. This will violate the i.i.d assumption of machine learning theory and might cause the model to overfit the data.

To decorrelate the training data, DQN uses a replay buffer to store transitions and randomly sample a mini-batch for training. Since the sampled data are note in the same episode or under the same policy, the data is more independent and closer to the i.i.d assumption.

2.3 Fixed Q-Target

Second problem is that the target and the action are determined by the current policy. After updating the model parameters, not the target and action are changed. That is the model is chasing a moving target and it might lead to instability in training.

To stabilize the training, DQN uses two network which are Q network θ and target Q network θ^-. The loss function is

$$L(\theta) = \mathbb{E}[(R(s, a, s^{'}) + \gamma \max_{a} Q(s^{'}, a^{'}|\theta^-) - Q(s, a|\theta))^2]$$

In the training, the target Q network is fixed and only update the Q network which avoids the problem of chasing moving target. Then synchronize the θ^- with the θ after certain number of updates (e.g. 1,000 updates).

3 DDPG

Deep Deterministic Policy Gradient (DDPG) uses the actor-critic approach adn neural network as value function approximation. The difference between DQN and DDPG is that the former one can only output action s in discrete space dn the latter one can output action s in continuous space.

In actor critic approach, both the value function and policy distribution is computed explicitly. In DDPG, both action and value function $Q(s, a)$ and deterministic policy distribution $\mu(a|s)$ are approximated using neural network.

$$\widehat{Q}(s,a|\theta^Q) \approx Q(s,a)$$
$$\widehat{\mu}(s,a|\theta^\mu) \approx \mu(a|s)$$

where the $\widehat{Q}(s,a|\theta^Q)$ is called critic netowrk and $\widehat{\mu}(s,a|\theta^\mu)$ is called acotr network.

Similar ot DQN, DDPG also uses the experience replay to decorrelate the training data and target network $Q^{'}$ and $\mu^{'}$ to stabilize the training.

3.1 Update of Critic Network

The critic network is used to estimated the action value of a state. The target and loss is similar to DQN except for the action is selected by actor network

$$target_1 = R(s_t, a_t, s_{t+1}) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$$

$$L(\theta^Q) = \frac{1}{N}\sum_{t=1}^{N}[(taret_1 - Q(s_t, a_t|\theta^Q))^2]$$

The model parameters of critic network can be updated using SGD. For the target critic network, DDPG uses soft update (e.g. $\tau = 0.001$)

$$\theta^{Q^{'}} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}}$$

3.2 Update of Actor Network

The actor network is used to select a deterministic action for a given state. The training objective is to maximize the expected Q value

$$J(\theta) = \mathbb{E}[Q(s,a)|s = s_t, a = u(s_t|\theta_\mu)]$$

The gradient of objective function with respect to actor network is

$$\nabla_{\theta\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta\mu}\mu(s_t|\theta^\mu)$$

Since the learning is off-policy with batch of transition experience, we compute the sample mean gradient and do gradient ascent to update the actor network parameters

$$\nabla_{\theta\mu} J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}[\nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta\mu}\mu(s_t|\theta^u)|_{s_i}]$$

For the target actor network, DDPG also uses soft update (e.g. $\tau = 0.001$)

$$\theta^\mu \leftarrow \tau\theta^\mu + (1-\tau)^{\theta^{\mu^{'}}}$$

4 Implementation

4.1 DQN Implementation

4.1.1 DQN Architecture

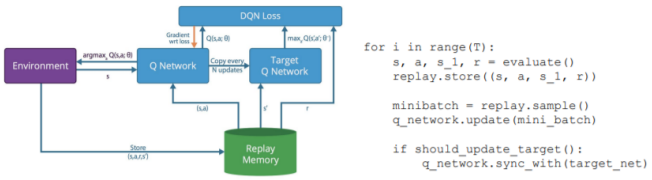

Figure 1: DQN architecture

The neural network parameters to estimate Q-value function.

```
DQN(
  (eval_net): Sequential(
    (0): Linear(in_features=4, out_features=32, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=32, out_features=2, bias=False)
  )
  (target_net): Sequential(
    (0): Linear(in_features=4, out_features=32, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=32, out_features=2, bias=False)
  )
)
```

Figure 2: A simple caption

4.2 DDPG Implementation

4.2.1 DDPG Architecture

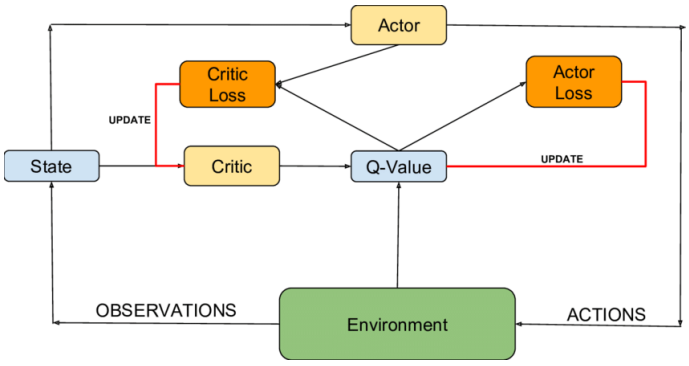

Figure 3: DDPG architecture

This is the implementation of DDPG.

5 Experimental Results

5.1 Episode Rewards in $Cartpole - v1$

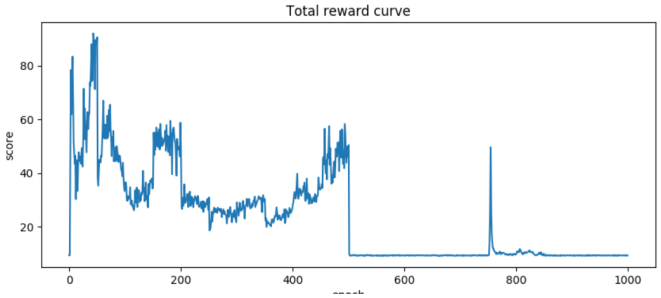

Figure 4: Total Reward Curve

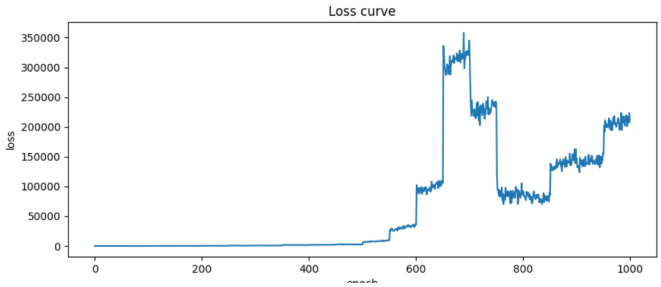

Figure 5: Loss Curve

5.2 Episode Rewards in $Pendulum - v0$

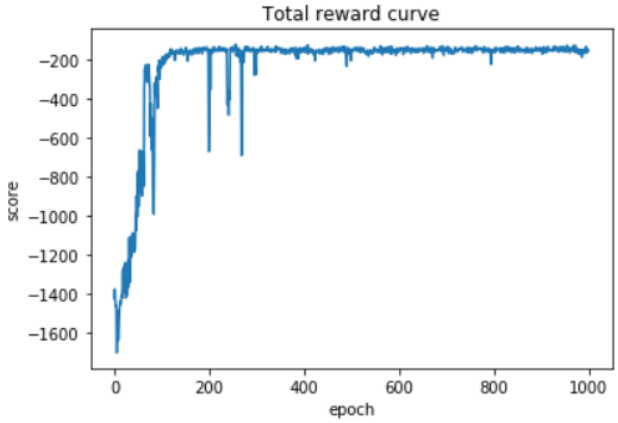

Figure 6: Total Reward Curve

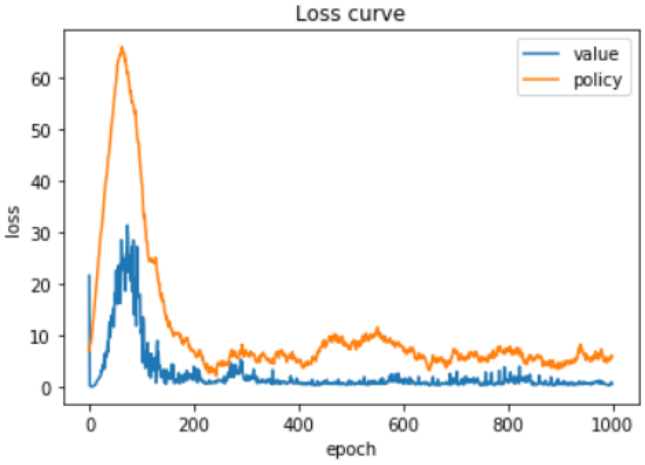

Figure 7: Loss Curve

6 Discussion

6.1 Difference between DQN and DDPG algorithm

DQN uses neural network to estimate the Q-value function. The output is the corresponding Q-value for the action. But the action space is discrete. For real life applications and actions, the action space is continuous.The action space becomes large if we discretize the action space finely. It becomes difficult to converge large action space.

6.2 Effects of discount factor, γ

The discount factor reveals the type of reward the agent interested in (i.e immediate or long term). γ of value 0 indicates the agent targets on immediate rewards with myopic evaluation and value of 1 indicates the long-term rewards with far-sighted vision. γ greater than 1 tends to divulge the actions of the agent on the environment. The discount rate bounded to be in the range of $(0, 1)$ to make the infinite sum finite. This helps in converging certain algorithms.

6.3 Benefits of ϵ-greedy over greedy action selection

Selecting from these two is based on the value of ϵ. We use epsilon-greedy method with low value of epsilon, such that there is a strong bias towards exploitation over exploration, favoring choosing the action with the highest Q-value over a random action. This helps to mitigate the negative effects of over-fitting and under-fitting.

Value of 0 for ϵ is kind of fully exploitative choice. However, keeping a small ϵ factor in policy allows to converge and choose for optimal policy

6.4 Necessity of target network

The target network is used an approximation of ground truth. The target network tells us when to tweak the network parameters with backpropagation to likely predict the state. A stable target network is used to measure the error of the learning of an agent.

6.5 Effect of replay buffer size

The size of buffer size affects the agents learning behavior. The size of memory size affects the agent to learn from earlier memories to speed up learning and break undesirable temporal correlations. There is a trade-off of amount of experiences you have to hold and you can hold. If you include lots of experience, it may slow the learning process.

7 Bonus

7.1 Choice of random process over normal distribution

Random process are essential for the exploration process. Normal distribution provides an prior knowledge about the environment which can be detrimental for choosing the optimal policy and the learning process. This random process distribution not only provides an optimal behaviour, but also inculcates the ability to change when the conditions in environment change.