DDPG Algorithm

Deep Deterministic Policy Gradient (DDPG) algorithm is a kind of Actor-Critic algorithms. It’s a model-free, off-policy algorithm using deep function approximators. DDPG was fist proposed by Deepmind in 2016, it’s based on deterministic policy gradient (DPG). Unlike Policy Gradient, the policy network in DPG and DDPG produce deterministic policy, that is, action of the agent in any given state can be directly output with a deterministic value, rather than probabilities in stochastic policy.

The structure of critic network is similar with the one in DQN, it also use Q value at given state and action (Q(s,a)) to evaluate the performance of a policy. But DQN cannot be straightforwardly applied to continuous domains, DDPG take advantage of the successful designs in DQN such as the off-policy approach and a target network, and can learn policies in high-dimensional, continuous action spaces.

Like the other actor-critic algorithms, DDPG also contains two different type of neural networks. The actor network is also called the policy network, it’s goal is to provide a optimal policy for the agent, that is, to choose the best action at any given state. And the performance of policy can be evaluated by the critic network, which output the Q value at given state and action. So the corresponding Q value can be used to calculate the loss function of the actor. And the way critic updates is very similar to the DQN algorithm.Both the actor and the critic have a local copy and a target copy. The local ones will be updated at every time step, while their targets are used to calculate the target values for them.The weights of these target networks are then updated by having them slowly track the learned networks. this technique is known as soft update , which can greatly improving the stability of learning.

And like DQN , DDPG also has a Replay Buffer, which stores experiences including state transitions, rewards and actions inside an internal memory of the agent. Whenever the agent need to learn, they can be sampled into mini-batches to update the network. Because the sampling is random, using experience replay can reduces correlation between experiences in updating DNN. It can also increases learning speed and reuses past transitions to avoid catastrophic forgetting.

In order to deal with the challenge of exploration in continuous action spaces, an exploration policy is constructed by adding noise sampled from a noise process N to the actor policy .

Work and Results

In my work , I have completed the implementation of the DDPG algorithm. Both the actors and critics are 3-layer MLP neural networks with 256 units at each hidden layer. And for the update of local actor and local critic, I use Adam optimizer with learning rate of 10-4 .

At each time step, local policy network output an action at given state, and an Uhlenbeck-Ornstein noise is been added to the original action value, hence the behavior policy. the agent then interact with environment using the behavior policy and store a experience tuple into the Replay Buffer with size of 10^6. and when there's enough experiences in the memory, agent starts the learning progress. But after trying to update the actor and critic network every episode, I found that my agent were constantly get stuck at a certain point with low total reward and could be hard to improved. So I change to a new strategy, which only let the agent to learn every 8 episodes, and when it's time to learn, it will call the learn() function for 6 times in a row. And after a lot of experiments, it turned out setting the discount value gamma to 0.8 will give a better performance.

After training, the agent can get +30 accumulated reward scores over 100 episodes. I trained it for over 500 episodes and the agent solved the environment after 525 episodes. I plot the rewards per episode for both training and testing, as shown below.

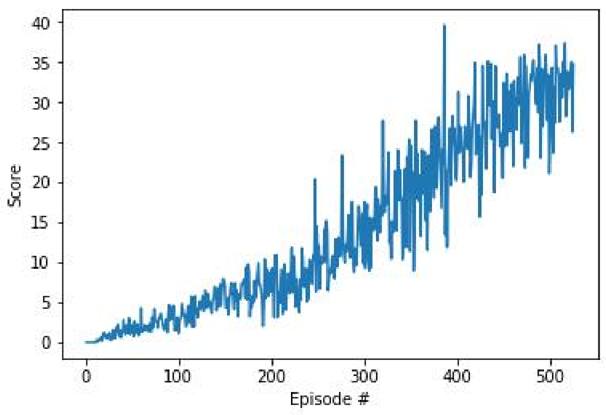

Figure 1. Average reward scores for training

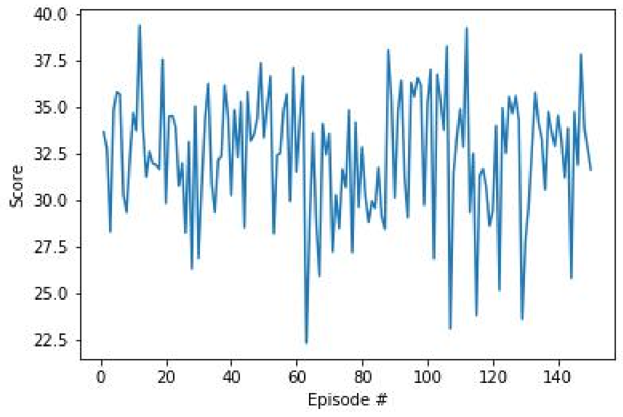

Figure 2. Average reward scores for testing

Plans for Future Work

For this project, I only implemented the DDPG algorithm. But for this kind of continuous control problems, there are a lot of other approaches, such as A2C,A3C,PPO and so on. So after this work, I intend to try out all the possible algorithms which are able to deal with continuous action space. I can also compare the performance between different kind of algorithms on this environment, and hopefully, I will get some importance insight out of this process. Other than that, I only solved the first version of the environment this time. So I planed to work on the second version in the near future.