CD294-112 Deep Reinforcement Learning HW3: Q-Learning and Actor-Critic

Ke Wang, EECS Ph.D. student, kewang@berkeley.com

2018/10/7

1 Part 1: Q-Learning

1.1 Question 1: basic Q-learning performance

Learning curve of the deep Q-Learning implementation on the game Pong.

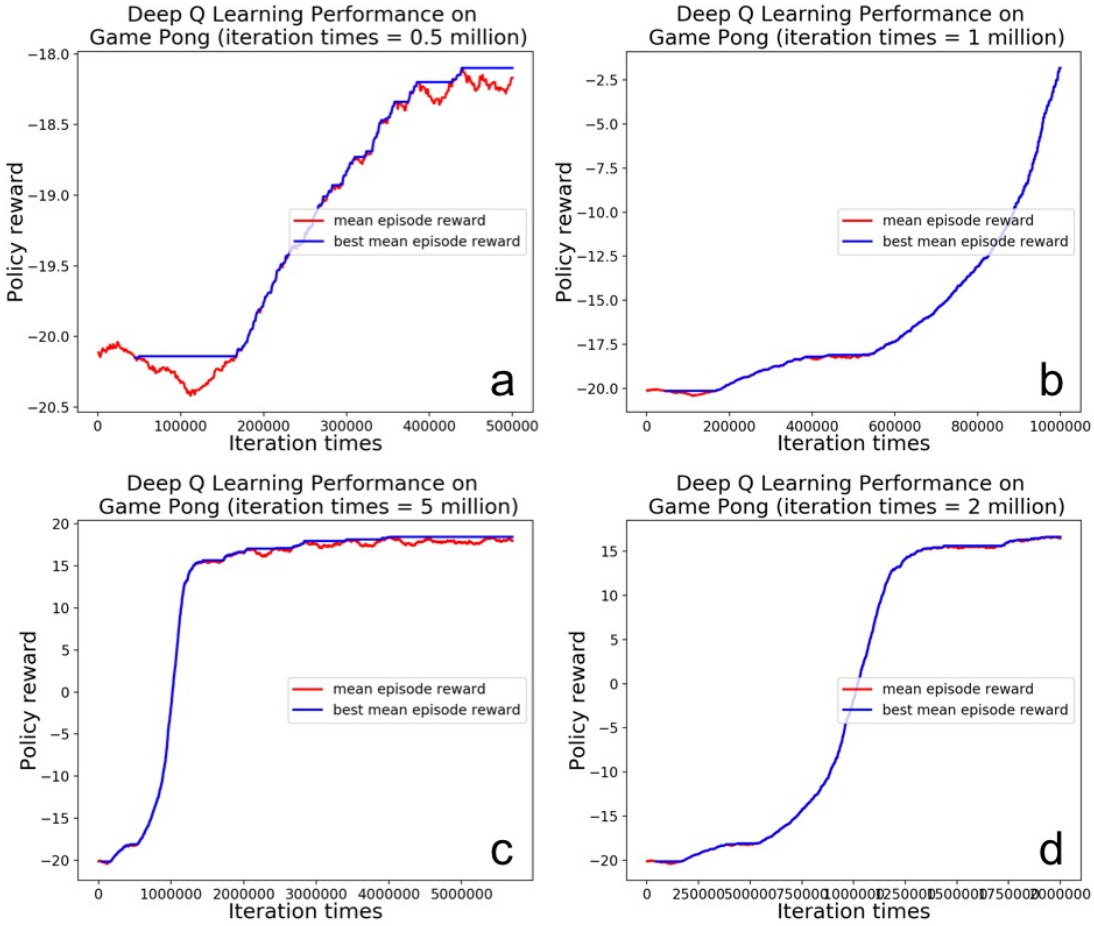

Figure 1: Comparations between different iteration times (0.5m, 1m, 2m, 5m), both mean reward and best reward are shown in the figures
. With the increase of the iteration times, the reward function become close to 20. The best reward is 19.05.

Hyper-parameter I change:

target_update_freq=10000

self.log_every_n_steps = 1000

self.save_every_n_steps = 50000

1.2 Question 2: double Q-learning

Tensorflow implementation of Double DQN:

```
self.q_value = q_func(obs_t_float,self.num_actions,scope = 'q_func_scope', reuse = False)
self.q_value_next = q_func(obs_tp1_float,self.num_actions,scope = 'q_func_scope', reuse = True)
self.prc = tf.argmax(self.q_value,axis = 1)


q_value_tar = q_func(obs_tp1_float,self.num_actions,scope = 'q_func_tar_scope', reuse = False)
q_func_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES,scope = 'q_func_scope')
target_q_func_vars = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES,scope = 'q_func_tar_scope')

best_action = tf.argmax(self.q_value_next,axis = 1)
print("kewang")
print(best_action.get_shape())
tar_val = self.rew_t_ph + (1-self.done_mask_ph)*gamma*tf.reduce_sum((q_value_tar*tf.one_hot(best_action,self.num_actions)),axis = 1)

q_val = tf.reduce_sum(self.q_value * tf.one_hot(self.act_t_ph, self.num_actions), axis=1)
self.total_error = huber_loss(q_val - tf.stop_gradient(tar_val), delta=1.0)
```

Figure 2: Tensor flow implementation of Double DQN

Resutls:

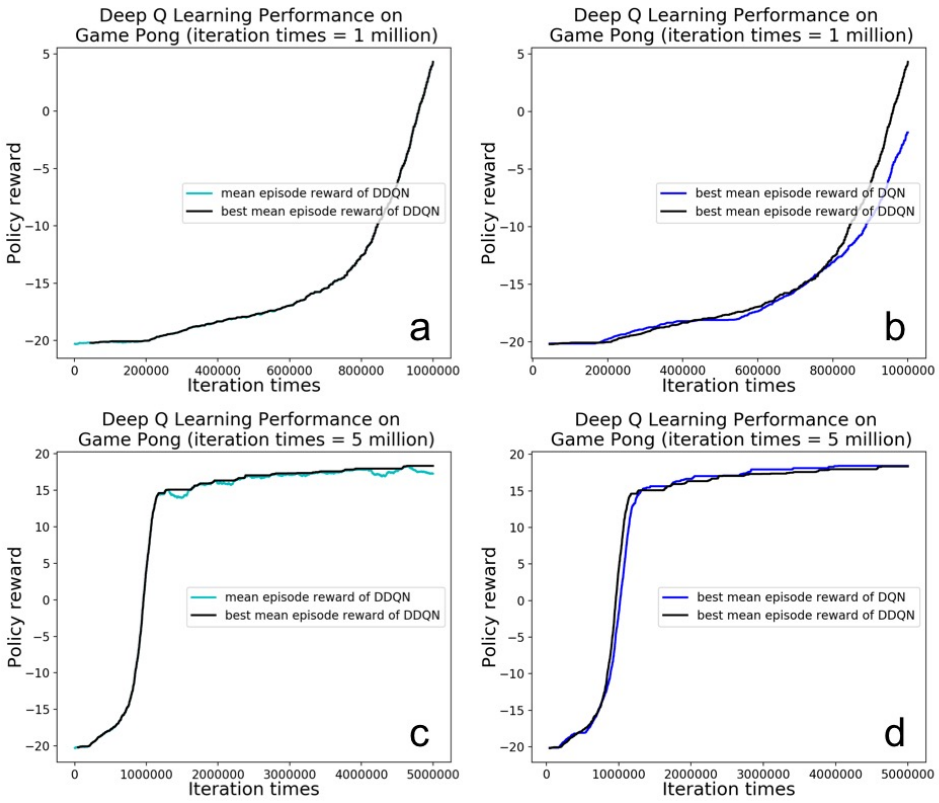

Figure 3: Figure a),c) show the mean reward and the best reward of double dqn. Figure b),d) show the comparation between ddqn and dqn

From the results, it is clear to see that double dqn converge faster than the basic dqn, which demonstrate the effectiveness of doubole dqn.

1.3 Question 3: experimenting with hyperparameters

In this section, I tried to analysis the sensitivity of Q-Learning on the Game Lunar Lander.a 1979 arcade game (also made by Atari) that has been implemented in OpenAI Gym. The parameter I chose was the neural network architecture, the output size of each fully connected network. The parameters chosen is this experiment are : 16,32,64,128. The comparations between different parameters are shown in the following figure:

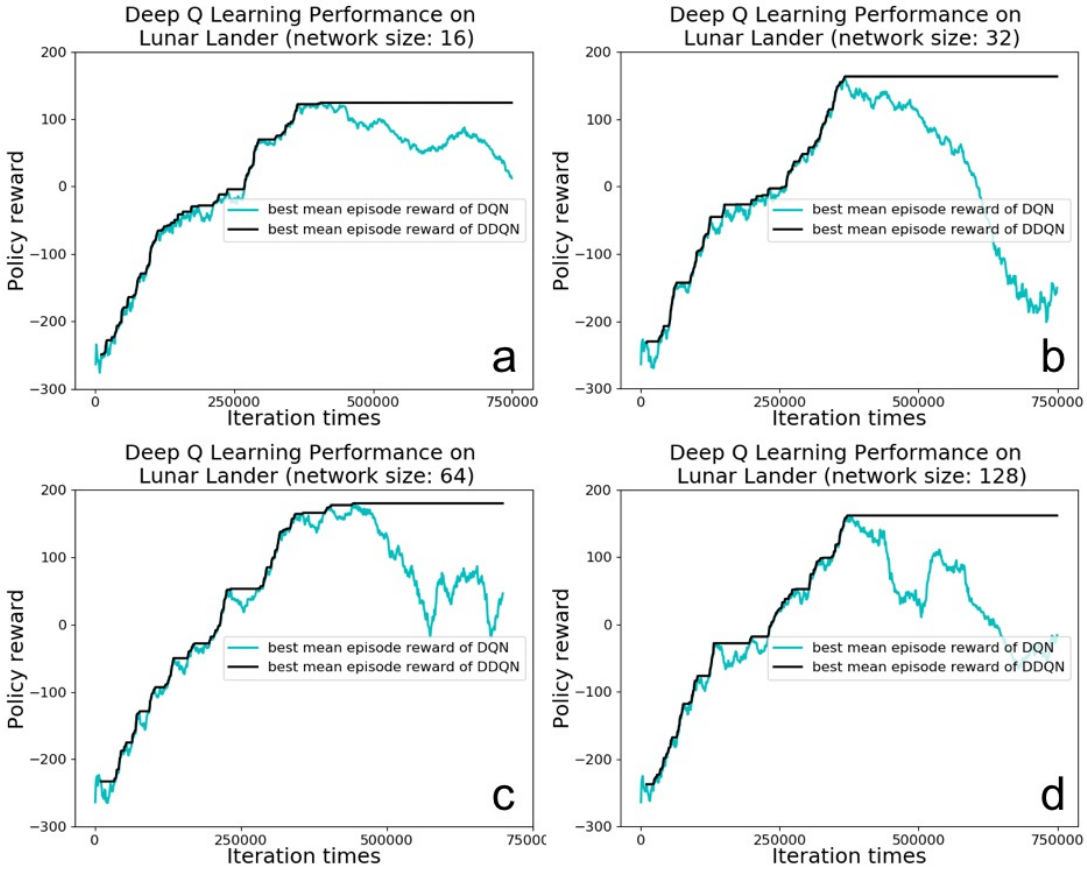

Figure 4: Figure a),b),c),d) show the learning curve under different neural network architecture

From the results, we could clearly see that under the network size equals to 64, the policy gets the best performance. Different parameters have non-trivial differences, when the network size is small, the simple architecture could not fully represent the complexity of the non-linear part of the deep Q-Learning Network. However, when the network size is large, we will have over $128 \times 128 \times 128$ parameters to train, which needs more training data. So, when the network size equals to 64, the policy could achieve the highest reward.

2 Part 2: Actor-Critic

2.1 Question 1: Sanity check with Cartpole

In this section, we tried to find out the best combination of number of target updates and number of gradient updates. As we use two network to train our model, we could not simpy use target updates or gradient updates. After the comparation, when we use number of target updates equals to 10 and number of gradient updates equals to 10, we will get the best performance, as shown in the Figure 5:

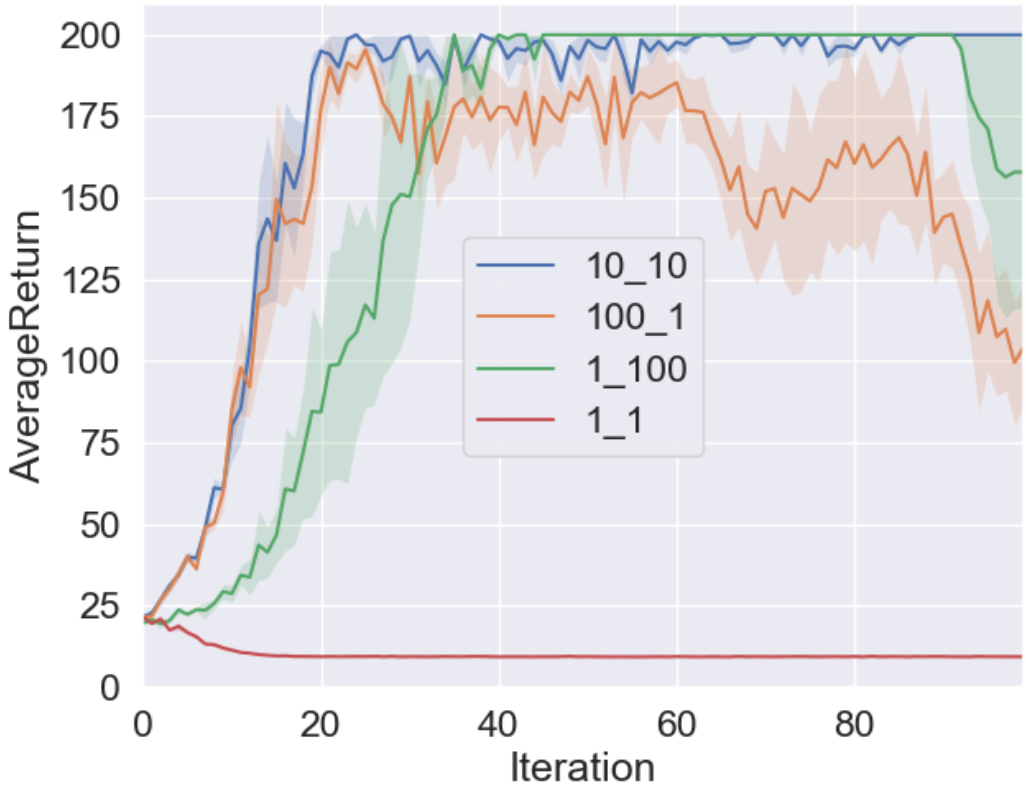

Figure 5: Learning curve of Actor-Critic task under different parameters

When using (10,10), we could get the best results after 100 iterations.

2.2 Question 2: Run actor-critic with more dicult tasks

2.2.1 InvertedPendulum

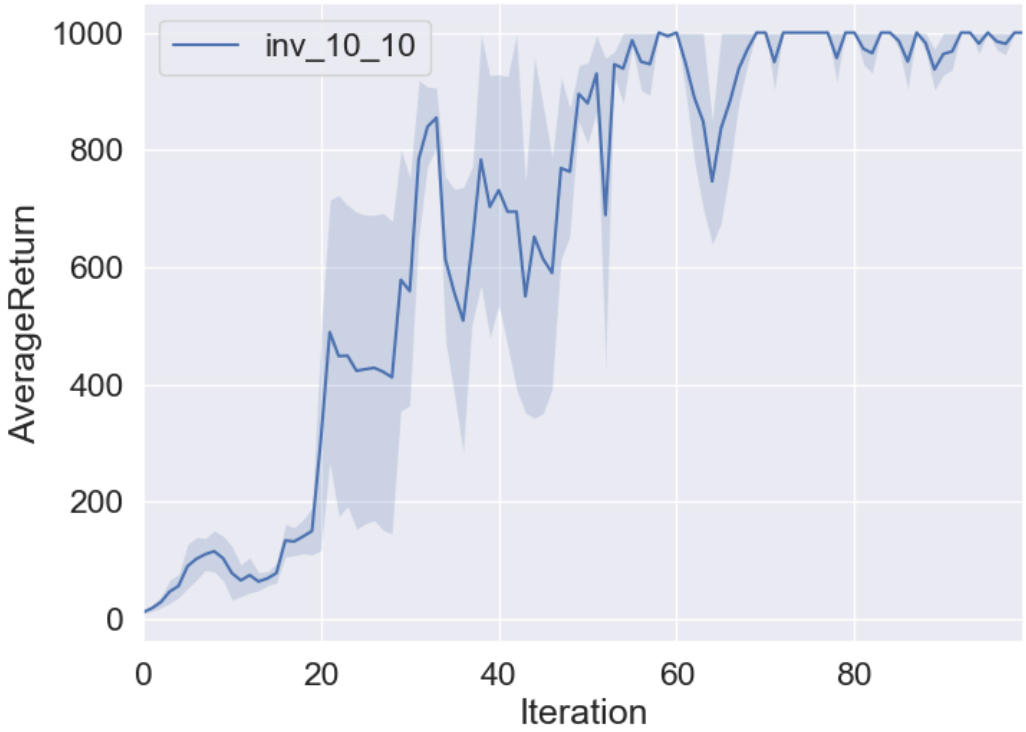

Figure 6: Learning curve of invertedPendulum

2.3 HalfCheetah

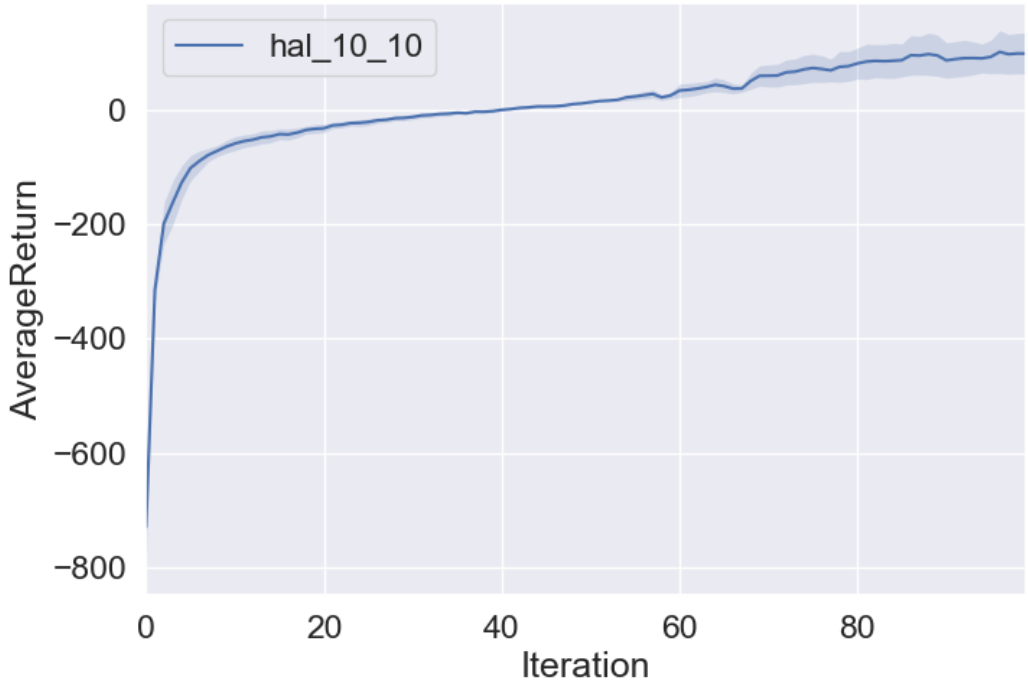

Figure 7: Learning curve of HalfCheetah

The results clearly show that by using Actor-Critic, we could get a relatively good result, a little bit worse than policy gradient.

2.4 Bonus

Here, we present a more expensive neural network. Instead of 2 layers, we use 3 layers with 96 hidden units. Also, we increase the number of target updates to 20, and decrease to number of action updates. The learning rates of these two network are also set to be different: for the actor network: 0.01, the action network:0.02. Here is the results:

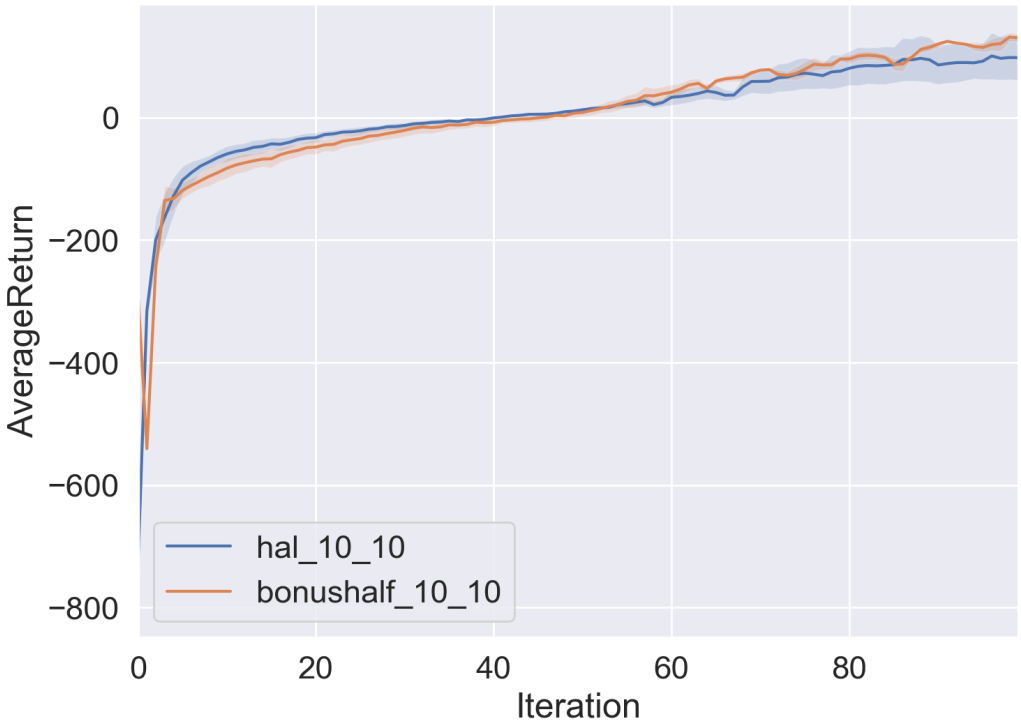

Figure 8: Learning curve of HalfCheetah with optimization

The results shows that, after optimization, we could achieve a better reward using deeper network and more hidden units.