1 Q-Learning

1.1 Question 1: basic Q-learning performance.

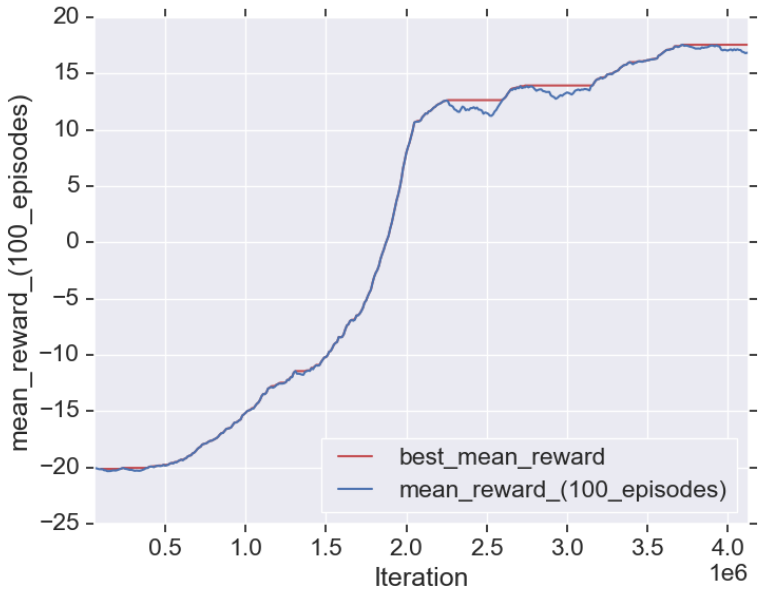

Figure 1: Deep Q-learning of atari game Pong using raw images.
I did NOT change any default hyperparameters. However, I stopped the simulation after four million steps so the maximum reward doesn't reach the maximum 20.
We can tell from the learning curve that my deep Q-learning algorithm is learning and it's just a matter of time for it to get to 20.(It should be noted that the y axis, although labeled as mean reward(100 episodes), it's actually just reward. Red: best mean reward, Blue: mean reward (100 episodes))

1.2 Question 2: double Q-learning.

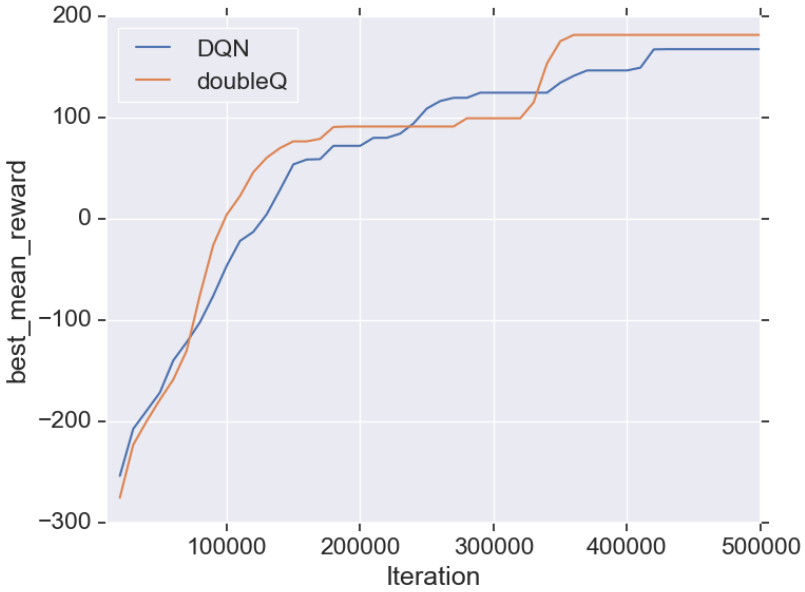

Figure 2: Performance comparison of double Q-learning and vanilla Q-learning Lunar Lander.
My double DQN (doubleQ) is slightly better than the vanilla DQN (DQN).I did NOT change any default hyperparameters.

1.3 Question 3: experimenting with hyperparameters.

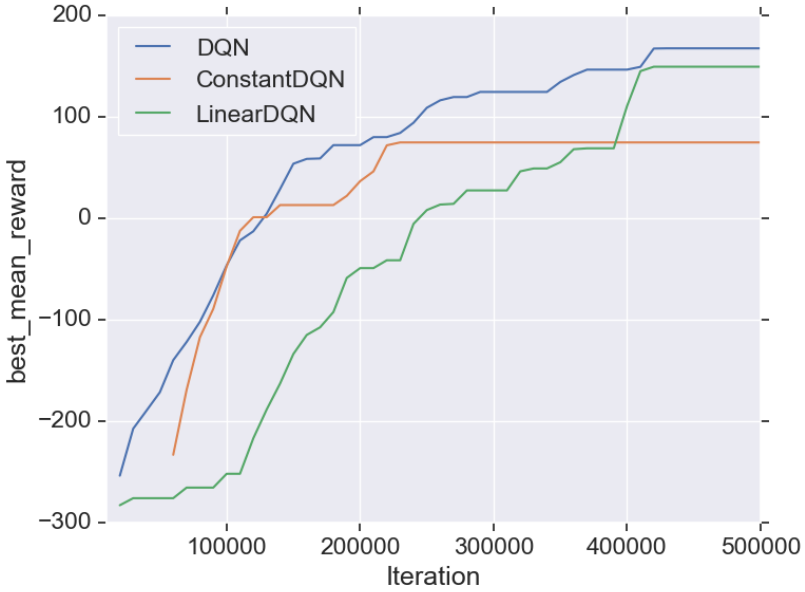

Figure 3: Hyperparameter tuning of exploration schedule using vanilla DQN on Lunar Lander.
I used the three given Schedule classes in dqn_utils.py: ConstantSchedule (ConstantDQN), LinearSchedule (LinearDQN), PiecewiseSchedule (DQN). The default piecewise schedule performs the best in Lunar Lander followed by the linear schedule. Constant schedule performs the worst among all three.

Constant schedule doesn't do well because exploration at later steps makes the system hard to converge to a stable solution and it always has to consider slightly different new observations.

Piecewise schedule learns faster than the linear schedule might be because the exploration rate decreases so fast in the linear schedule that the system has not picked up enough variety of observations yet. In an environment with diverse states/observations, it's often good to take a two-step approach: exploring more at the beginning and fitting the collected data later.

```
# Question 1
# To run the analysis, use the following line
python run_dqn_atari.py
# To plot the result, use the following line
python multiplot.py data\qlearn_exp_PongNoFrameskip-v4_24-09-2018_10-10-51 --value
    best_mean_reward mean_reward_(100_episodes) --legend best_mean_reward
    mean_reward_(100_episodes)

# Question 2
# To run the analysis, use the following line
python run_dqn_lander.py
python run_dqn_lander.py --doubleQ --exp_name doubleQ
# To plot the result, use the following line
python plot.py data\qlearn_DQN_LunarLander-v2_25-09-2018_22-47-04
    data\qlearn_doubleQ_LunarLander-v2_27-09-2018_18-56-47 --value best_mean_reward

# Question 3
# To run the analysis, use the following line
python run_dqn_lander.py
python run_dqn_lander.py --schedule ConstantSchedule --exp_name ConstantDQN
python run_dqn_lander.py --schedule LinearSchedule --exp_name LinearDQN
# To plot the result, use the following line
python plot.py data\qlearn_DQN_LunarLander-v2_25-09-2018_22-47-04
    data\qlearn_ConstantDQN_LunarLander-v2_25-09-2018_23-52-17
    data\qlearn_LinearDQN_LunarLander-v2_26-09-2018_00-15-31 --value best_mean_reward
```

2 Actor-Critic

2.1 Question 1: Sanity check with Cartpole

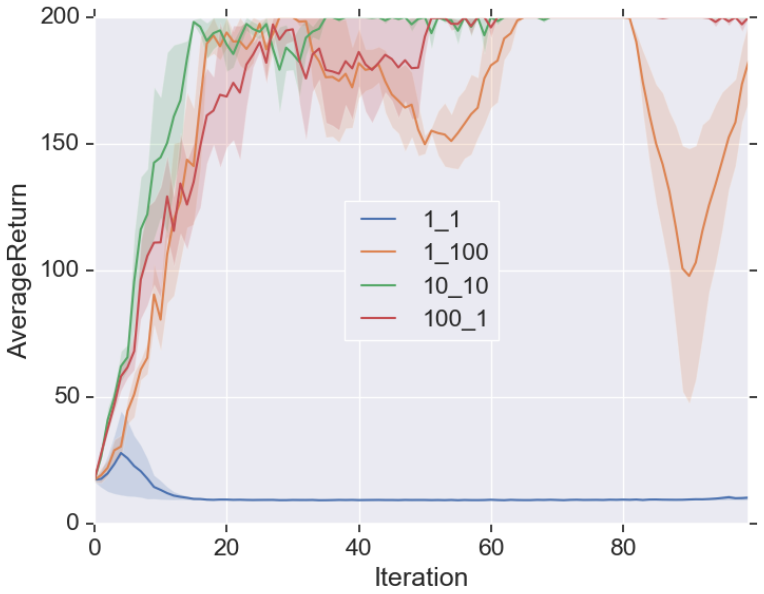

Figure 4: Target update and gradient update balancing of Actor-Critic algorithm on CartPole-v0.
I used four different setting as required in the instruction:
1 target update and 1 gradient step per target update (1_1),
1 target update and 100 gradient step per target update (1_100),
10 target update and 10 gradient step per target update (10_10),
100 target update and 1 gradient step per target update (100_1).

As we can see, all except 1_1 reach the maximum reward. However, 1_100 is not very stable because running 100 gradient steps per target update might result in a inaccurate target value during gradient update. Once target function is updated, there might be large changes in the reward. 10_10 gets to the maximum reward faster than 100_1. This is simply because policy gets updated more early on due to multiple gradient steps taken. Only taking one gradient step per update is not enough to have a good performance. Overall, there is a balance between target update and gradient step per target update.

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

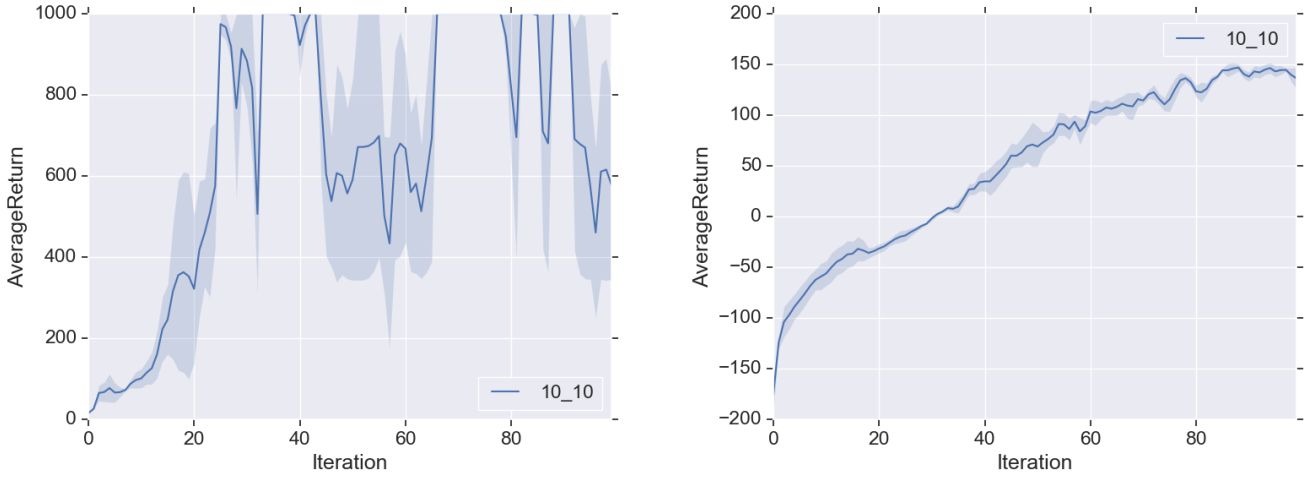

Figure 5: Actor-Critic algorithm on InvertedPendulum (left) and HalfCheetah (right). I did NOT change any hyperparameters. Both match the results from policy gradient in HW2.

3 Bonus

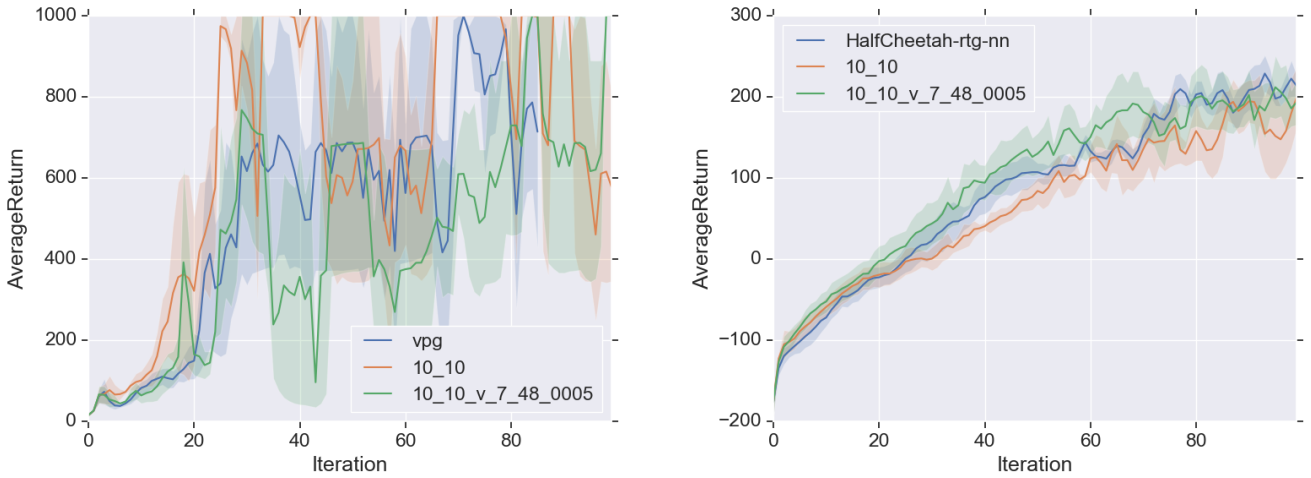

Figure 6: Hyperparameter tuning of the critic network on InvertedPendulum (left) and HalfCheetah (right). Left: policy gradient (vpg), actor-critic (10_10), actor-critic with independent critic network (10_10_v_7_48_0005); Right: policy gradient with normalization (HalfCheetah-rtg-nn), actor-critic (10_10), actor-critic with independent critic network (10_10_v_7_48_0005).

We can see that by implementing an independent critic, the overall performance is indeed improved in both environments. However, because the original solution is already close to optimal, we don't really see a big improvements by having a more expressive critic network.

The hyperparameters I used for the critic network are as following:

number of hidden layers: 7, number of hidden units per layer: 48, learning rate: 0.005. I also changed the discount factor of HalfCheetah to 0.95. All other hyperparameters are the same as the default in actor-critic.

```
## Actor-critic
# Question 1.
# To run the analysis, use the following line
python train_ac_f18.py CartPole-v0 -n 100 -b 1000 -e 3 --exp_name 1_1 -ntu 1 -ngsptu 1
python train_ac_f18.py CartPole-v0 -n 100 -b 1000 -e 3 --exp_name 100_1 -ntu 100 -ngsptu
    1
python train_ac_f18.py CartPole-v0 -n 100 -b 1000 -e 3 --exp_name 1_100 -ntu 1 -ngsptu
    100
python train_ac_f18.py CartPole-v0 -n 100 -b 1000 -e 3 --exp_name 10_10 -ntu 10 -ngsptu
    10

# To plot the result, use the following line
python plot.py data\ac_1_1_CartPole-v0_27-09-2018_19-36-12
    data\ac_1_100_CartPole-v0_27-09-2018_19-42-41
    data\ac_10_10_CartPole-v0_27-09-2018_19-45-46
    data\ac_100_1_CartPole-v0_27-09-2018_19-39-10


# Question 2.
# To run the analysis, use the following line
python train_ac_f18.py InvertedPendulum-v2 -ep 1000 --discount 0.95 -n 100 -e 3 -l 2 -s
    64 -b 5000 -lr 0.01 --exp_name 10_10 -ntu 10 -ngsptu 10
python train_ac_f18.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3 -l 2 -s 32 -b
    30000 -lr 0.02 --exp_name 10_10 -ntu 10 -ngsptu 10

# To plot the result, use the following line
python plot.py data\ac_10_10_InvertedPendulum-v2_27-09-2018_19-56-45
    ..\hw2\data\vpg_InvertedPendulum-v2_16-09-2018_17-32-43
python plot.py data\ac_10_10_HalfCheetah-v2_28-09-2018_13-44-24
    ..\hw2\data\HalfCheetah-rtg-nn_HalfCheetah-v2_18-09-2018_10-43-21
    ..\hw2\data\HalfCheetah_HalfCheetah-v2_17-09-2018_23-52-16
# ac_10_10_HalfCheetah-v2_27-09-2018_20-09-29


# Bonus
# To run the analysis, use the following line
python train_ac_f18_bonus.py HalfCheetah-v2 -ep 150 --discount 0.95 -n 100 -e 3 -l 2 -s
    32 -b 30000 -lr 0.02 --exp_name 10_10_v_7_48_0005 -ntu 10 -ngsptu 10 -vl 7 -vs 48
    -vlr 0.005
python train_ac_f18_bonus.py InvertedPendulum-v2 -ep 1000 --discount 0.95 -n 100 -e 3 -l
    2 -s 64 -b 5000 -lr 0.01 --exp_name 10_10_v_7_48_0005 -ntu 10 -ngsptu 10 -vl 7 -vs
    48 -vlr 0.005

# To plot the result, use the following line
python plot.py ..\hw2\data\HalfCheetah-rtg-nn_HalfCheetah-v2_18-09-2018_10-43-21
    data\ac_10_10_HalfCheetah-v2_28-09-2018_13-44-24
    data\ac_10_10_v_7_48_0005_HalfCheetah-v2_28-09-2018_22-07-19
python plot.py ..\hw2\data\vpg_InvertedPendulum-v2_16-09-2018_17-32-43
    data\ac_10_10_InvertedPendulum-v2_27-09-2018_19-56-45
    data\ac_10_10_v_7_48_0005_InvertedPendulum-v2_29-09-2018_00-15-23
```