1 Question 1

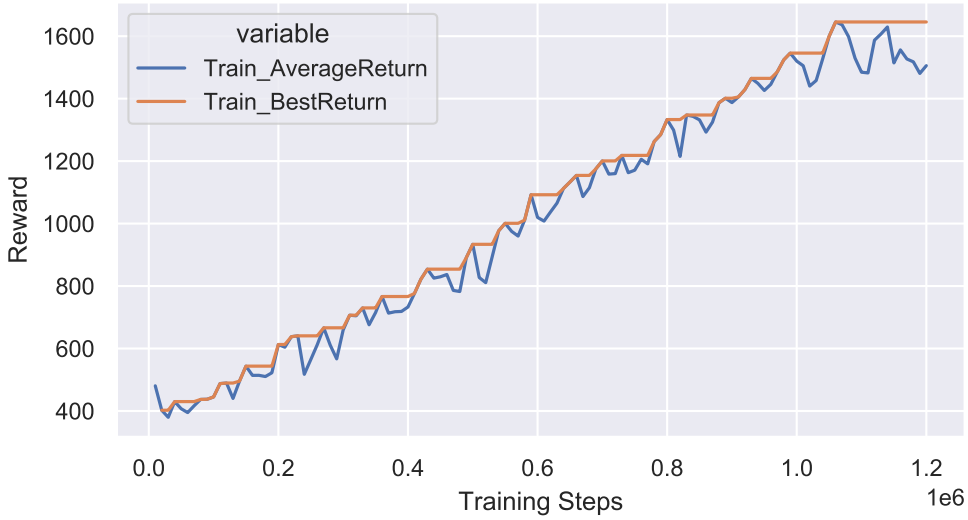

Figure 1: basic Q-learning performance

Listing 1: Exact command line configurations

```
python cs285/scripts/run_hw3_dqn.py --env_name MsPacman-v0 --exp_name q1
```

2 Question 2

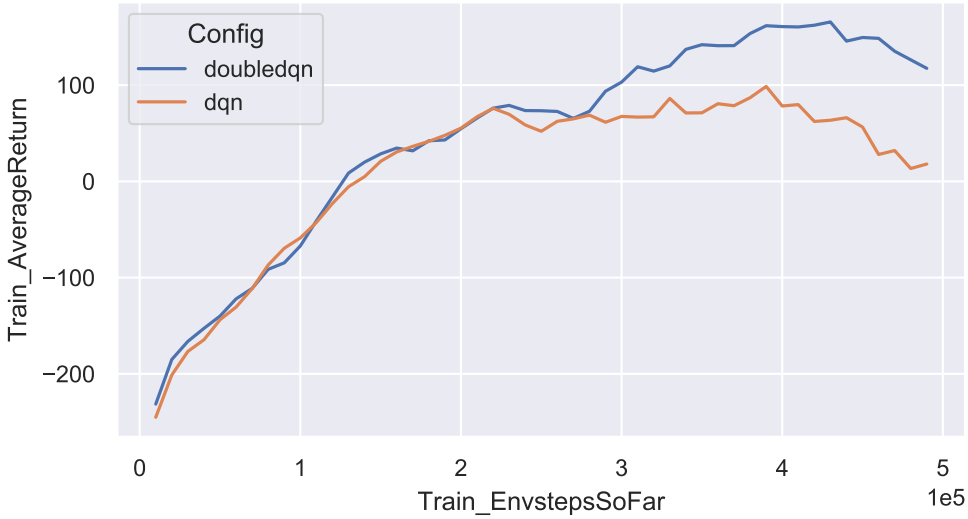

Figure 2: double Q-learning

Listing 2: Exact command line configurations

```
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_dqn_1 --seed 1
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_dqn_2 --seed 2
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_dqn_3 --seed 3

python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_doubledqn_1 --
    double_q --seed 1
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_doubledqn_2 --
    double_q --seed 2
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q2_doubledqn_3 --
    double_q --seed 3
```

As expected, double Q-learning does better than normal DQN. Note that the three runs for DQN and double DQN were averaged together.

3 Question 3

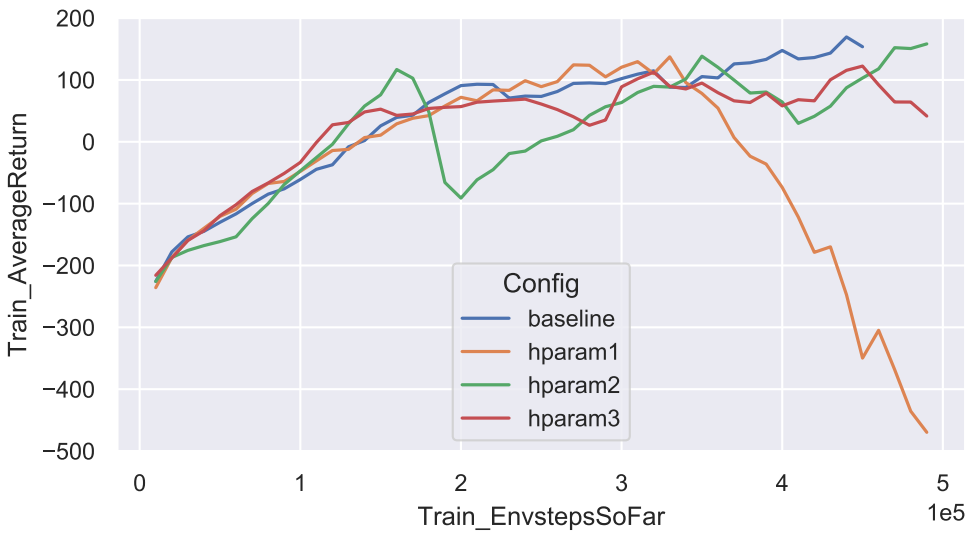

Figure 3: experimenting with hyperparameters

Listing 3: Exact command line configurations

```
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q3_hparam1
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q3_hparam2
python cs285/scripts/run_hw3_dqn.py --env_name LunarLander-v3 --exp_name q3_hparam3
```

For this question, I experimented with the neural network architecture. The baseline is the default network architecture: 2 hidden layers, each of size 64. For `hparam1`, I increased the network size to 3 hidden layers, each with 64 units. For `hparam2`, the network had 2 hidden layers, each of size 128. For `hparam3`, the network had 3 hidden layers, each with 128 units. Most of these experiments performed similarly, except for `hparam1`, which looks to have diverged and achieved a very negative reward!

4 Question 4

Figure 4: Sanity check with Cartpole

Listing 4: Exact command line configurations

```
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b 1000 --exp_name q4_ac_1_1 -
    ntu 1 -ngsptu 1
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b 1000 --exp_name q4_100_1 -
    ntu 100 -ngsptu 1
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b 1000 --exp_name q4_1_100 -
    ntu 1 -ngsptu 100
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b 1000 --exp_name q4_10_10 -
    ntu 10 -ngsptu 10
```

From the graph, 10 target updates, each every 10 gradient steps and 1 target update, each every 100 gradient steps appear to perform the best, although the single target update looks to converge faster, so I used those settings (`-ntu 1 -ngsptu 100`) to test on the harder tasks.

5 Question 5

Figure 5: Run actor-critic with more dicult tasks: `InvertedPendulum-v2`

Listing 5: Exact command line configurations

```
python cs285/scripts/run_hw3_actor_critic.py --env_name InvertedPendulum-v2 --ep_len 1000 --
    discount 0.95 -n 100 -l 2 -s 64 -b 5000 -lr 0.01 --exp_name q5_1_100 -ntu 1 -ngsptu 100
```

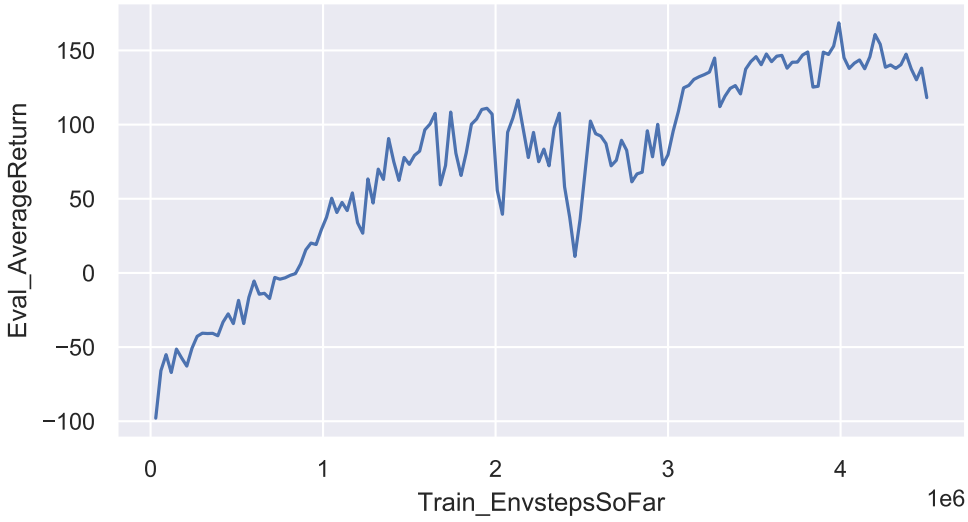

Figure 6: Run actor-critic with more dicult tasks: `HalfCheetah-v2`

Listing 6: Exact command line configurations

```
python cs285/scripts/run_hw3_actor_critic.py --env_name HalfCheetah-v2 --ep_len 150 --
    discount 0.90 --scalar_log_freq 1 -n 150 -l 2 -s 32 -b 30000 -eb 1500 -lr 0.02 --
    exp_name q5_1_100 -ntu 1 -ngsptu 100
```