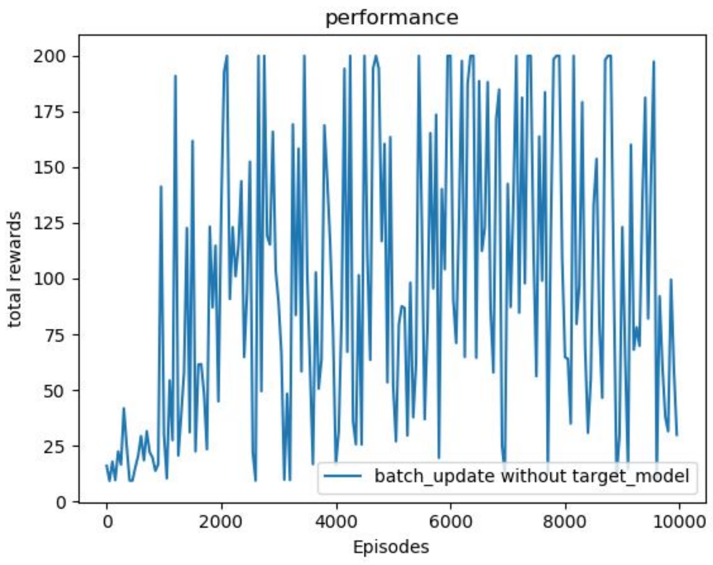
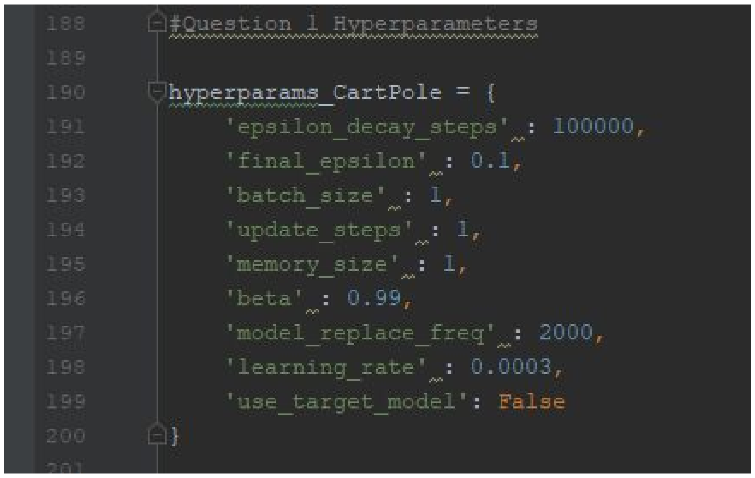
CS 533: Intelligent Agent and Decision
Homework 4

Report by:

Christopher Buss
Sanad Saha

1. **DQN without a replay buffer and without a target network. This is just standard Q-learning with a function approximator. The corresponding parameters are: memory_size = 1, update_steps = 1, batch_size = 1, use_target_model = False**



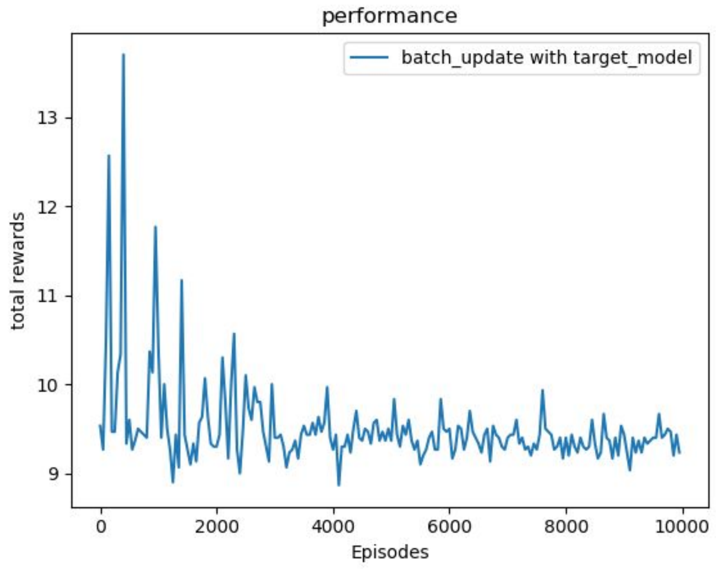


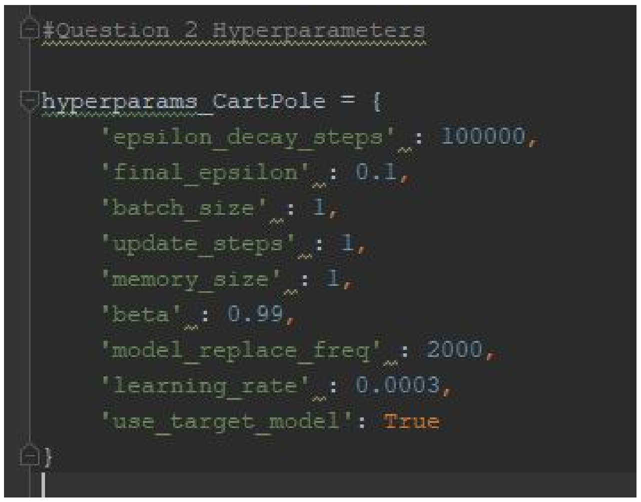
```
#Question 1 Hyperparameters

hyperparams_CartPole = {
    'epsilon_decay_steps' : 100000,
    'final_epsilon' : 0.1,
    'batch_size' : 1,
    'update_steps' : 1,
    'memory_size' : 1,
    'beta' : 0.99,
    'model_replace_freq' : 2000,
    'learning_rate' : 0.0003,
    'use_target_model': False
}
```

Performance of DQN model [Episodes vs Rewards] without replay buffer and without target network

To run the DQN without a replay buffer we used memory size = 1 (as indicated in the instructions). As a result, no experience is stored in memory, thus invalidating the replay buffer. Without the use of random sampling from a memory buffer, each state-action pair is highly correlated with the previous pair (and the future pair), which makes our model's movement along the gradient more biased towards a specific direction and much less stable. See the figure on the left for the hyperparameters used to produce this graph.

2. **DQN without a replay buffer (but including the target network). The corresponding parameters are: memory_size = 1, update_steps = 1, batch_size = 1, use_target_model = True**



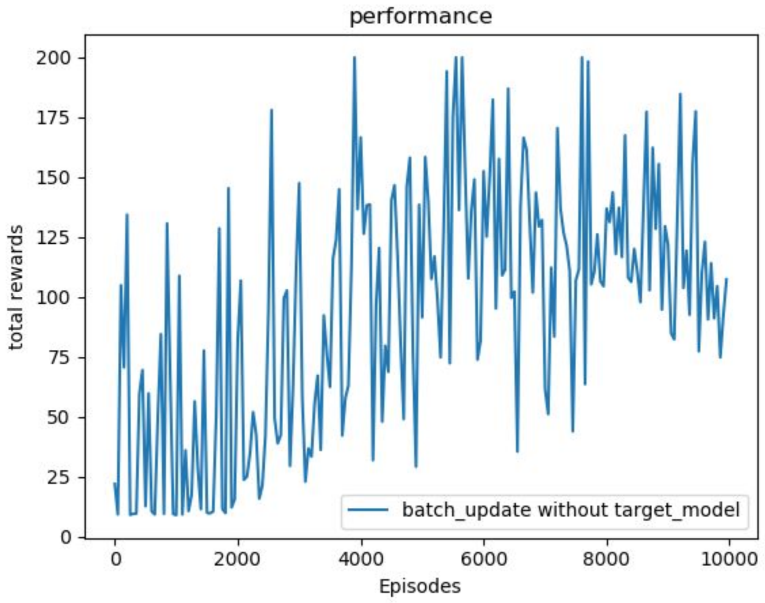


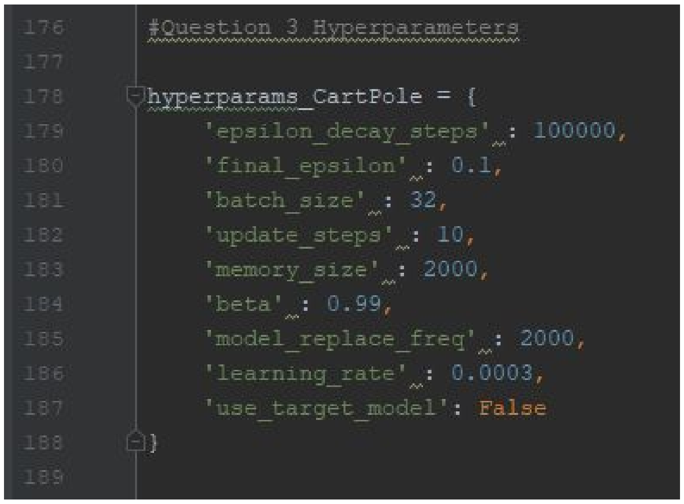
```
#Question 2 Hyperparameters

hyperparams_CartPole = {
    'epsilon_decay_steps' : 100000,
    'final_epsilon' : 0.1,
    'batch_size' : 1,
    'update_steps' : 1,
    'memory_size' : 1,
    'beta' : 0.99,
    'model_replace_freq' : 2000,
    'learning_rate' : 0.0003,
    'use_target_model': True
}
```

Performance of DQN model [Episodes vs Rewards] without replay buffer but with target network

The target model ends up adding more stability to our updates. This makes sense because it doesn't let the highly correlated examples (that we saw in part 1) quickly compound. That is, with the target network, our parameters don't explode because they are growing with respect to a fixed target (our target network). Despite the added stability, the lack of batch sampling still hurts the model a lot. See the figure on the left for the hyperparameters used to produce this graph.

3. **DQN with a replay buffer, but without a target network. Here you set use_target_model = False and otherwise set the replay memory parameters to the above suggested values**



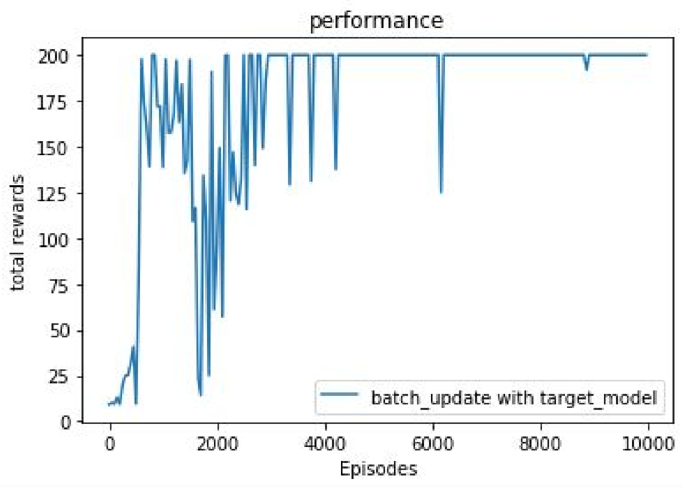


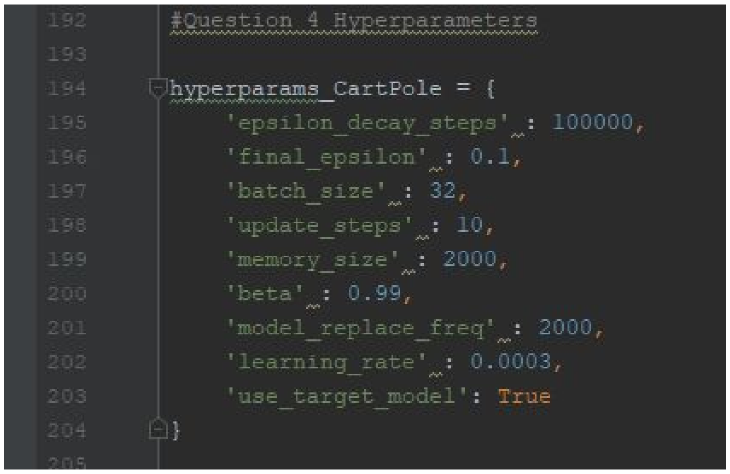
```
#Question 3 Hyperparameters

hyperparams_CartPole = {
    'epsilon_decay_steps' : 100000,
    'final_epsilon' : 0.1,
    'batch_size' : 32,
    'update_steps' : 10,
    'memory_size' : 2000,
    'beta' : 0.99,
    'model_replace_freq' : 2000,
    'learning_rate' : 0.0003,
    'use_target_model': False
}
```

Performance of DQN model [Episodes vs Rewards] with replay buffer but without target network

Here we can see our DQN is now going towards convergence (although really slowly) which can be contributed to the use of the experience replay buffer. The replay buffer lets us reuse past transitions to decorrelate samples, but the absence of the target model reduces some stability. As the TD error is changed frequently through the DQN, training gets difficult with an unstable target. It should also be noted that the use of mini batches with replay buffer reduces execution time four fold over previous experiments. See the figure on the left for the hyperparameters used to produce this graph.

4. Full DQN





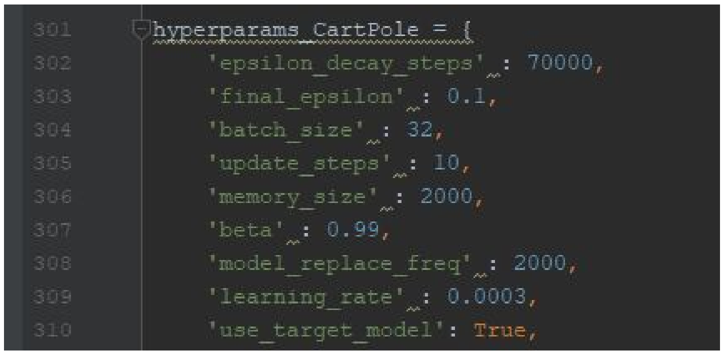
```
#Question 4 Hyperparameters

hyperparams_CartPole = {
    'epsilon_decay_steps' : 100000,
    'final_epsilon' : 0.1,
    'batch_size' : 32,
    'update_steps' : 10,
    'memory_size' : 2000,
    'beta' : 0.99,
    'model_replace_freq' : 2000,
    'learning_rate' : 0.0003,
    'use_target_model': True
}
```

Performance of Complete DQN model [Episodes vs Rewards]

Unsurprisingly, the addition of the target network greatly increases the performance of our model. Not only are our gradient updates more varied (with experience replay), but they are also more stable thanks to the target network. This target network eliminates most of the fluctuation we saw in the previous experiment. See the figure on the left for the hyperparameters used to produce this graph.

5. Distributed Experiments

The following figure shows the parameters used to run these following experiments. Note that we also set *training_episodes* to 7000 and *test_interval* to 50 (these do not appear in the figure below). The only thing we changed from experiment to experiment was the amount of collectors whereas the evaluators was kept at 4. Due to time constraints, we weren't able to complete the experiment with **4 collector and 4 evaluation worker** for 7000 training episodes, so we ran it for **4500 *training_episodes***. Running it for 4500 episodes was enough to reach convergence and show that it was very slow compared to experiments for (8, 4) and (12, 4) (collector, evaluator) workers.



```
hyperparams_CartPole = {
    'epsilon_decay_steps' : 70000,
    'final_epsilon' : 0.1,
    'batch_size' : 32,
    'update_steps' : 10,
    'memory_size' : 2000,
    'beta' : 0.99,
    'model_replace_freq' : 2000,
    'learning_rate' : 0.0003,
    'use_target_model': True,
```

The following table will help us to compare runtime improvement upon increasing the number of workers:

Number of workers	Collector = 4
	Evaluator = 4
		Collector = 8
		Evaluator = 4
			Collector = 12
			Evaluator = 4

Runtime (seconds)	5717.7219 (4500 episodes)	6267.8293 (7000 episodes)	5578.95568 (7000 episodes)

From the table above we can see the timing (runtime) of the Full DQN. As we increase the number of workers runtime is improving significantly, as expected.

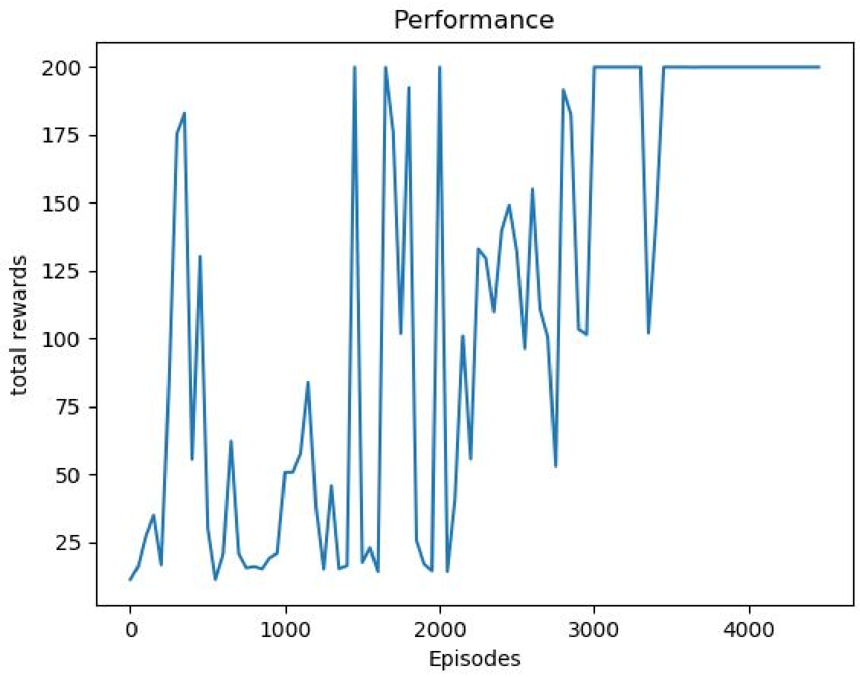


Figure: Performance of Distributed DQN with 4 collectors and 4 evaluation workers ran for 4500 episodes.