ECEN 689 Reinforcement Learning Assignment-3 Report

Question 1: Implement DQN (with experience replay and frozen targets) using a Neural Network as the Q function approximator on the LunarLander-v2 environment. Plot the total episodic reward during training.

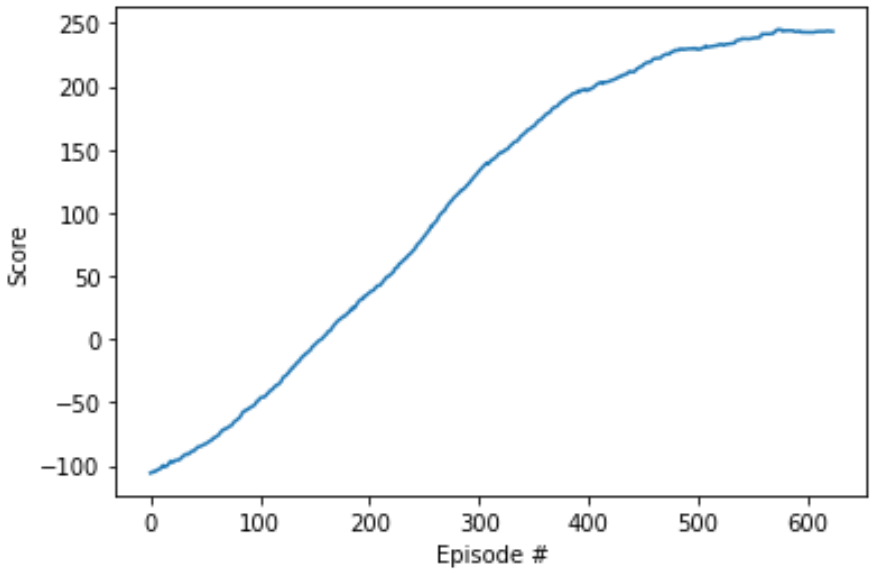
Answer: DQN was implemented with frozen targets and experience replay using a neural network as the function approximator on Lunar-Lander V2 environment. Code is attached in the zip file.

Used PyTorch to implement the neural network used for Q-Network and Target network

In order to achieve a reward of above 200 consistently, training is stopped after we have achieved an episodic reward of 250. Training is done for 1000 episodes. Lunar Lander is observed to land properly in the designated place at reward of 250, hence threshold is taken as 250.

For the plot of episodic reward vs training, sliding window technique is employed to observe clean and clear plot.

Plot of episodic reward during training using sliding window:



Question 2: Describe your Neural Network (number of hidden layers, hypermeters etc)

Answer: I used a simple feedforward neural network for both the q-network and target network. The neural network is inputted with state size i.e., 8. It has two hidden layers, whereas first hidden layer has 128 nodes, and the second hidden layer also has 128 nodes. The second hidden layer maps to action size i.e., 4. At every node, 'ReLu' activation function is used.

Hyperparameters employed are:
Batch size: 1e5 (100000)
Buffer size: 64
Discount factor = 0.99
Learning rate = 0.0005
Target network is updated for every 4 timesteps.
Maximum number of samples in each episode: 1000
Starting value of epsilon: 1
Ending value of epsilon: 0.01
Decay rate of epsilon: 0.095

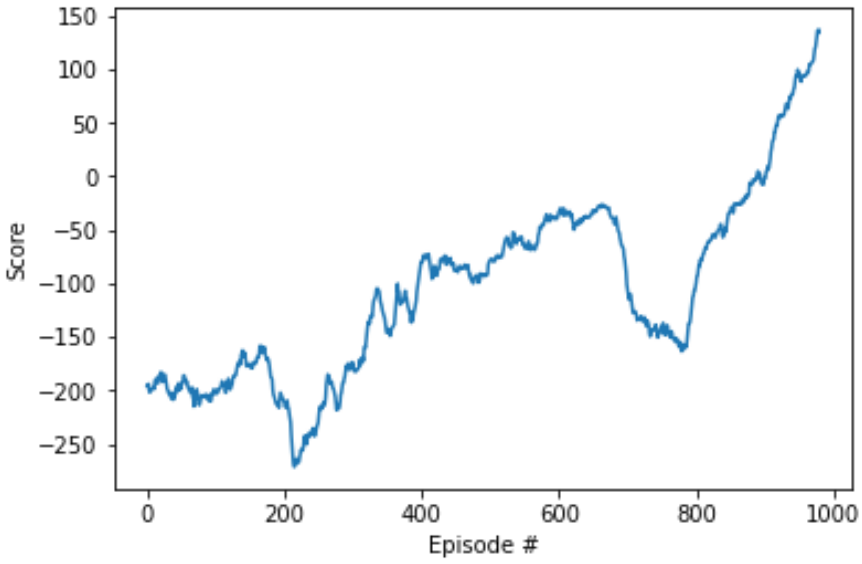
Question 3: You have to convince the world that you have landed on the moon(There are some who will not believe you), Submit a video of your trained agent

Answer: Video of the trained agent is attached in the zip file

Question 4: Implement DQN with frozen targets and without experience replay.Plot the total episodic reward during training. Explain your observations.

Answer: DQN was implemented with frozen targets and without experience replay using a neural network as the function approximator on Lunar-Lander V2 environment. Code is attached in the zip file attached.

Plot of episodic reward during training:



Observations: I have used the same neural network, hyperparameters and same 1000 episodes for both settings of with and without experience replay to have a fair comparison. It can be observed from both the plots of episodic reward during training that the plot without experience replay has more variance in comparison with the plot with experience replay. This high variance is due to insufficient learning since we don't get any knowledge from the experience buffer like we get in DQN. It can also be observed that DQN without experience

replay is only able to achieve a max reward around 100-150 after 1000 episodes whereas DQN with experience replay achieves a reward of around 200-250 which ensure safe landing in the desired place. Hence DQN with experience replay performs better than DQN without experience replay.

For DQN without experience replay, buffer size and batch size are taken to be 1. Since buffer is initialized as a deque, it will always provide and learn with the new sample.

Question 5: How would you train if the action space is continuous

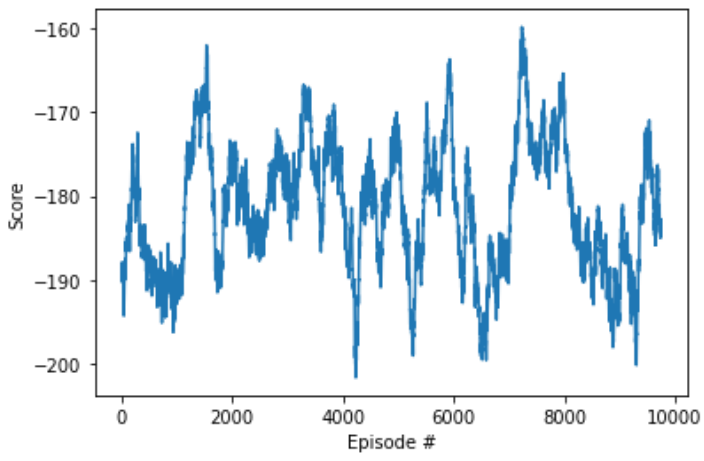
Answer: When the action space is discrete, we can just calculate the Q-values for each action individually and find the action that maximizes the Q-value. If the action space is continuous, it will be very difficult to find out the action that maximizes Q-value by exhaustively searching the complete search space. It would be infeasible to search the whole action space every time an agent decides to take an action in the environment.

Since the action space is said to be continuous, Q (s, a) can be assumed to be differentiable with respect to action space. Hence, we can use a gradient-based learning rule to update the policy to find the ideal action. Instead of exhaustively searching the whole action space to find $\max_a Q(s, a)$, we can approximate it with $Q(s, \mu(s))$ by concurrently obtaining the optimal policy $\mu(s)$ from the gradient-based learning rule.

Deep Deterministic Policy Gradient (DDPG) is usually employed for environments with continuous action spaces. Here we learn an approximator to a*(s) to adapt to continuous action spaces along with learning an approximator to Q*(s, a).

Bonus question: Off policy algorithms such as Q-Learning have the advantage that the policy used to collect data for training can be different from the current policy being trained. Train an agent using DQN where the policy used to collect data for trainings random (agent takes all actions with equal probability in every state). Plot the total episodic reward obtained by the agent's policy during training. Describe why this approach is better/worse than the DQN that you implemented in the previous questions

Answer :
Plot of episodic reward during training :



An agent was trained using DQN where the policy that is used to collect data for trainings is random. This means that agent should take all actions with equal probability in every state. This means that we have implement DQN only with exploitation but no exploration where any policy is followed randomly. Since it is random, all actions are taken with equal probability in every state.

Observations : Here since we are only exploiting but not exploring, we are always learning from the scenario of what worked earlier. Due to this the learning of the network will be significantly less since it will not go into many unexplored horizons. It keeps learning the same initial better scenario and the reward will be around the initial better reward which is obtained to be -170. Hence, the rewards are oscillating in the range of -160 to -200 which is significantly worse than the DQN where we explored along with exploiting.

Question 6 (Project report in next page):

Analysing Independent vs Cooperative Multi-agent RL on Google Research's Football environment.

- Sai Namith Garapati - UIN : 832001176
- Subrahmanyam Arunachalam - UIN: 431000480
- Brent Arnold Basiano - UIN: 130004869

Multi-agent Reinforcement Learning is one of the booming fields of Reinforcement Learning (RL) which is seeing exponential growth in terms of research. In standard RL settings until now or from what we have seen in class, we mostly dealt with single agent learning from the environment. We have not seen the phenomenon of one agent learning through coordination with multiple agents, this type of learning through coordination is explored more in these multi-agent settings. In single-agent RL, our focus mainly relies on developing algorithms that maximize the cumulative reward. But in multi-agent RL, research is mainly focused on evaluating and quantifying metrics like co-operation. One such example is analysing the Independent vs cooperative multi agent RL. In independent multi-agent RL, all agents are independent but still perform better than single agent RL due to the availability of more resources and having a better chance of receiving rewards. Cooperative Multi-agent RL is a way of learning where agents can learn from other agents through instantaneous information, episodic experience and learned knowledge. This interesting idea is taken upon for our project. The environment we have taken to implement is Google Research's Football environment. Google Research's football environment provides multiple scenarios for implementing multi-agent RL and the environment is compatible with OpenAI Gym.

Regarding the environment, we as a team felt that an environment with a football setting is very ideal to analyse cooperation in multiagent RL. Football relies primarily on the cooperation between players and the strategy behind it. It also provides the opportunity to analyse scenarios where players can behave independently without showing much cooperation and scenarios where they can play with cooperation. Google research's football environment not only has the opportunity to train the whole football game, but also provides somewhat simpler scenarios like 3 vs 1 with keeper, easy counter attack, hard counter attack…etc where we can implement multi-agent RL easily. We are planning to analyse Independent and Cooperative multi agent RL by implementing both the settings on the scenarios provided by Google Research's Football environment. We will train the agents or players in these above mentioned scenarios with different distributions modelling them as independent/co-operative agents. After training them in both the settings, we plan to evaluate both the settings with metrics like Average Goal Difference, Number of steps. We are planning to use DQN to train the model since we feel we have enough knowledge and time with us to use DQN and get the results. We will also try implementing PPO alongside DQN and see which one performs better in an independent and cooperative setting. These are the current ideas we have with us going forward. We are concerned whether we have enough resources to train the model and implement it. We will spend time and try our best to implement the things possible to our strength. Please provide any feedback/ valuable suggestions if we can do something better or analyse something else with the same settings and resources available. It would be great if some insightful ideas are also provided that could help us move forward.