Value Function Approximations Deep Q-Networks and Double Deep Q-Networks

Rugved Hattekar 50320920

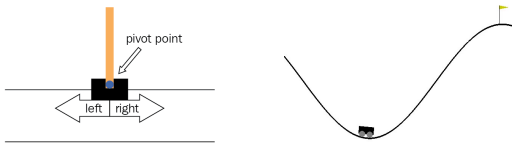
April 2020

1 Introduction

The goal of the assignment is to explore OpenAI Gym environments and implement value function approximation algorithms. In the first part of the project will be focused on implement deep Q learning (DQN). The purpose of this project is to understand the effectiveness of deep neural networks as well as some of the techniques used in practice to stabilize training and achieve better performance. We will train our networks on two OpenAI gym (Cartpole and Mountain Car). In the second part I will implement an improvement to the DQN algorithm, by using double DQN Algorithm.

2 Environments

In this project, I implemented two gym environments. Gym is a toolkit for developing and comparing reinforcement learning algorithms. The first gym environment is 1. Cartpole Environment and second one is 2. Mountain Car.
Cartpole - known also as an Inverted Pendulum is a pendulum with a center of gravity above its pivot point. It's unstable, but can be controlled by moving the pivot point under the center of mass. The goal is to keep the cartpole balanced by applying appropriate forces to a pivot point. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.



(a) Cartole (b) Mountain-Car

Figure 1: Environments

2.1 MDP of Cartpole and Mountaincar Environment

1. **Cartpole**
$Velocity = (-inf, inf)$
$Position = (-2.4, 2.4)$
$Actions = (left, right)$
$reward = 1$

2. **Mountain-Car**
$Velocity = (-0.07, 0.07)$
$Position = (-1.2, 0.6)$
$Actions = (left, neutral, right)$
$reward = -1$

3 What is the benefit of using experience replay in DQN?

Deep Q-learning algorithm differs from normal Q-learning algorithm in a way such that it uses experience replay and a neural network to estimate the next state of the environment, which an important step as of bellman equation, and uses and recursion algorithm. The basic idea is that by storing an agent's experiences, and then randomly drawing batches of them to train the network, we can more robustly learn to perform well in the task. By keeping the experiences we draw random, we prevent the network from only learning about what it is immediately doing in the environment, and allow it to learn from a more varied array of past experiences. Each of these experiences are stored as a tuple of state,action,reward,next state. The Experience Replay buffer stores a fixed number of recent memories, and as new ones come in, old ones are removed. When the time comes to train, we simply draw a uniform batch of random memories from the buffer, and train our network with them. This function is written in my code as replay function for a DQN class.

4 What is the benefit of the target network?

The DQN performs better than simple Q-Learning because is better because it performs utilization of a second network during the training procedure. This second network is used to generate the target-Q values that will be used to compute the loss for every action during training. Why not use just use one network for both estimations? The issue is that at every step of training, the Q-network's values shift, and if we are using a constantly shifting set of values to adjust our network values, then the value estimations can easily spiral out of control. The network can become destabilized by falling into feedback loops between the target and estimated Q-values. In order to mitigate that risk, the

target network's weights are fixed, and only periodically or slowly updated to the primary Q-networks values. In this way training can proceed in a more stable manner. Instead of updating the target network periodically and all at once, we will be updating it frequently after every 10 iterations.

5 What is the benefit of representing the Q function as q^(s, w)?

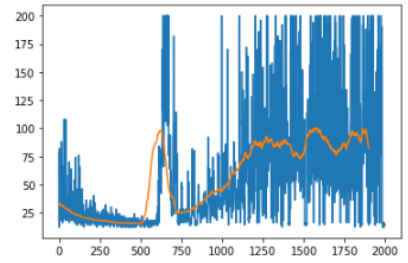
Q function is basically the value function, which is derived from bellman equation. This Q-function basically provides values for each state action par and the maximum of that value is then selected. The Q-function, creates a table of values with rows representing the states and columns representing the actions, but this is very naive representation for a very huge set of state , action space. This does not give a flexibility to a Q-function to be used for larger state data, Hence parametrized representation is considered. We all know that Neural Netwoks are excellent value function approximators for the matter of fact, there are many other value function approximators but the neural networks are widely used once and are ubiquitous these days, so we using the neural networks, with the help of weights we can represent the value function as q(s,w) with s being state and w being weights. These weights are updated to give man=ximum action values, which again gives the liberty to represent the value function more efficiently.

6 Reults of DQN

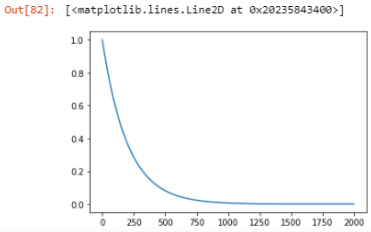
After applying DQN on cartpole and mountain-car following results were obtained. The input of the neural network will be the observations and the number of output neurons will be the number of the actions that an agent can take. For training the neural network the targets would be the Q-values of each of the actions and the input would be the state that the agent is in. The experience replay provides state, actions, next-state,reward and the using these values the weights are calculated and updated. After applying DQN to the cartpole, with 2 hidden layers, with one input layer as observations and output layer as number of actions. Both the environments run for 4000 iterations. The reward can be seen to be in increasing over the number of iterations and the agent seems to learn and win the every episode after particular itertion.

1. **Cartpole Hyper-Paramaters**
$DiscountFactor = 0.995$
$LearningRate = 0.001$
$Epsilon = 1$
$EpsilonDecay = 0.995$

1. **Mountain Car Hyper-Paramaters**
$DiscountFactor = 0.995$
$LearningRate = 0.01$
$Epsilon = 1$
$EpsilonDecay = 0.995$

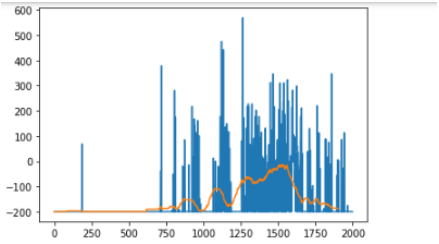


(a) DQN Cartpole

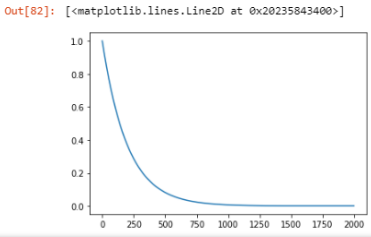


(b) Epsilon Decay Cartpole

Figure 2: Reward and Epsilon Decay Cartpole



(a) DQN Mountain car



(b) Epsilon Decay Mountain car

Figure 3: Reward and Epsilon Decay Mountain Car

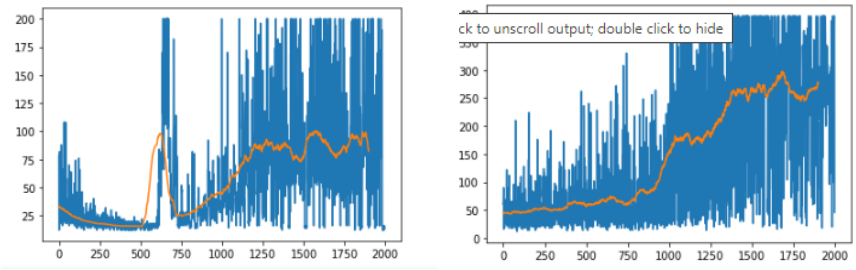
7 Double Deep Q-Learning

7.1 What algorithm you implemented and what is the main improvement over the vanilla DQN?

Double DQN as the name suggests uses two identical neural network models. One learns during the experience replay, just like DQN does, and the other one is a copy of the last episode of the first model. The Q-value is actually computed with this second model. Because, In DQN, Q-value is calculated with the reward added to the next state maximum Q-value. Obviously, if every time the Q-value calculates a high number for a certain state, the value that is obtained from the output of the neural network for that specific state, will become higher every

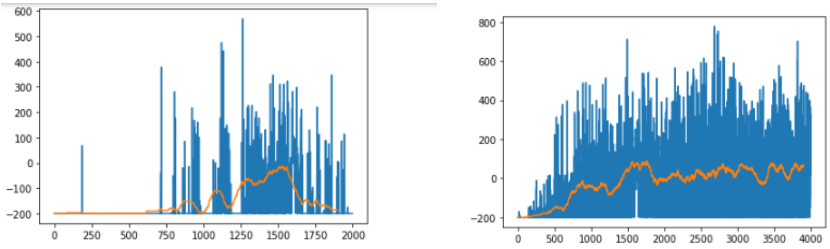
time. Each output neuron value will get higher and higher until the difference between each output value is high. Now if let's say for state s action a is a higher value than action b, then action a will be chosen every time for state s. now consider if for some memory experience action b becomes the better action for state s. then since the neural network is trained in a way to give a much higher value for action a when given state s, it is difficult to train the network to learn that action b is the better action in some conditions. So to bring down the difference between the output values (actions). Use a secondary model that is the copy of the main model from the last episode and obviously, since the difference between values of the second model are lower than the main model, we use this second model to attain the Q-value:

7.2 Comparison between DQN and Double-DQN Alorithms



(a) DQN Cartpole (b) DDQN Catpole

Figure 4: DQN vs Double DQN Cartpole



(a) DQN Mountain car (b) DDQN Mountain car

Figure 5: DQN vs Double DQN Mountain Car

As you can see in the image above that the DDQN algorithm is much faster as well as very much smooth as compared to the vanilla DQN algorithm. For e.g. The uncertain peaks are avoided in DDQN algorithm. This is because the second neural network is used in DDQN algorithm. The second neural next keeps updating its weight after every 10 episods in the code, but the sampling

it does after 1st neural network takes into the consideration all possible actions and does not get keeps improving one particular action over the period of time. The DDQN Algorithm is more robust compared to the Vanilla DQN.

8 Conclusion

In this project I implemented the DQN and Double DQN algorithms on a Gym Environments. While implementing this, one can easily figure out the importance of choosing right hyper parameters. We can easily understand the need for developement of the DDQN algorithm over the DQN algorithm as the DQN algorithm becomes unstable and over estimates certain q-values. The DQN and DDQN algorithms hence thoroughly studied in this assignement.

9 References

1. https://medium.com/@jonathan$_hui/rl-dqn-deep-q-network-e207751f7ae4$
2.$https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288$
3.$IntroductiontoReinforcementLearningbySuttonandBurto.$
4.$https://gym.openai.com/envs/MountainCar-v0/$
5.$https://gym.openai.com/envs/CartPole-v0/$