CSE 510 Reinforcement Learning Spring 2020 Project 2

Using Deep Reinforcement Learning to solve OpenAI Gym Environments

Srisai Karthik Neelamraju
UBID - 50316785
University at Buffalo
Buffalo, NY 14260
neelamra@buffalo.edu

Abstract

Reinforcement learning (RL) is that branch of machine learning which deals with how automated agents learn to take actions in unknown environments so as to optimize some performance metric. Deep reinforcement learning (DRL) is an advanced RL algorithm that combines deep learning with RL. Unlike traditional RL algorithms like Q-learning and SARSA, these algorithms make use of neural networks to approximate Q-value function as opposed to storing the values in the form of tables. In this project, two OpenAI Gym environments, namely CartPole and BreakOut, are used to implement and analyze both vanilla deep Q-learning and double deep Q-learning algorithms. Importance of using techniques like experience replay and target network to make the learning more efficient is detailed. Upon training DRL agents using both the algorithms, it is observed that double deep Q-learning algorithm performs better than the vanilla implementation, in terms of both learning speed and Q-network stability.

1 Introduction

Reinforcement learning (RL) is used for enabling an automated agent learn its underlying environment over time. Many traditional RL algorithms like Q-learning, value iteration, TD-learning and SARSA are tabular methods, since they use tables to store values corresponding to all possible state action pairs in the environment. Deep reinforcement learning (DRL) is a sophisticated RL algorithm that employs deep learning techniques to learn Q-value function through the means of neural networks. DRL is known to work well especially in scenarios with complex environments that have huge state and action spaces. Besides, it is also impactful in scenarios with real-valued and continuous observation spaces. In this project, deep Q-learning is used to solve two games - CartPole and BreakOut, both of which are already framed as MDPs and available from OpenAI Gym. Both vanilla and double deep Q-learning variants of the algorithm are implemented to play the two games. The average reward of the DRL agent against training episodes is analyzed in all the implementations. It is observed that the double deep Q-learning based agent solved the games far better than the vanilla deep Q-learning agent, thus demonstrating the effectiveness of the former algorithm.

2 Deep Q-Learning

Q-learning is a traditional method used to solve reinforcement learning environments. It stores a Q-value for each possible state-action pair in the environment and then updates these values while learning the underlying environment. After learning phase, the optimal policy in each state is the action with the highest Q-value. However, as the complexity of an environment increases, implementation of Q-learning becomes more and more impractical. This is where deep Q-learning finds its importance. It is an advanced RL algorithm that uses a neural network, also called as Q-network, to approximate the Q-value function instead of storing all values in a Q-table. In other words, this technique involves a neural network that takes as input the current state and outputs Q-values corresponding to all possible actions in that state. An alternative approach of achieving this is to design a network that takes as input both the current state and the action, and then outputs the corresponding Q-value. However, this approach is ineffective as it requires multiple forward propagation passes to find the best possible action in the next state. For this project, the former approach is used for creating a Q-network. The deep Q-learning algorithm is described in Figure 1 and the interactions between different entities involved in the algorithm are depicted.

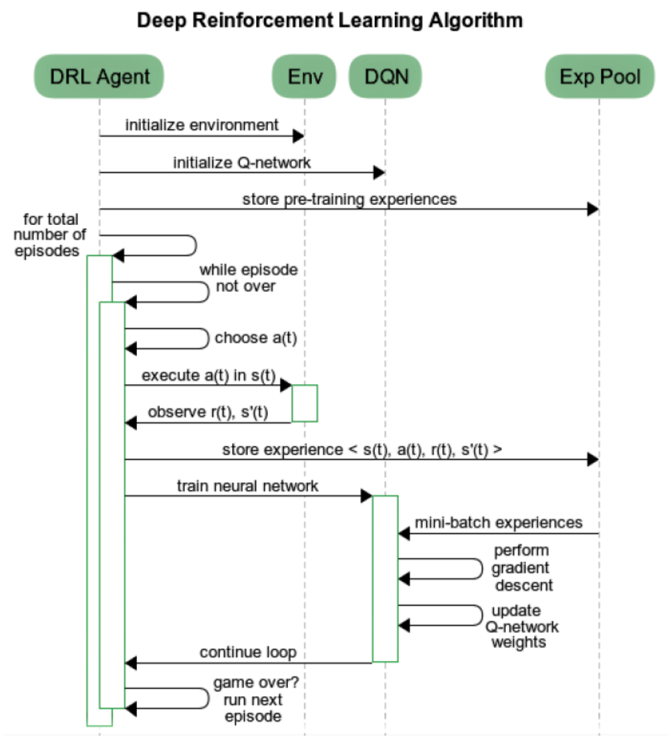


Figure 1: Time sequence of steps involved in deep Q-learning for one iteration

In deep reinforcement learning algorithms, experience replay technique is often used to train the Q-network. It helps in training the neural network better because it makes better use of the previous experiences of the agent. Sometimes, pre-training experiences are stored in the experience pool before the agent starts any learning. An experience is a tuple containing the current state, action, reward, next state and any other relevant information. In each training step, the agent samples a subset of experiences from the pool and uses them to optimize the Q-network parameters (θ). The target Q-value for state *s*(*t*) and the action *a*(*t*) chosen in that state is defined using the formula (here, *s*'(*t*) is the state the agent will transit into),

$$y(t) = r(t) + \gamma \max_{a'} Q\left(s'(t), a'; \theta\right)$$

Further, the loss function is defined as the mean squared error between the target Q-value and the estimated Q-value and any standard optimizer like RMSProp, SGD and Adam is used to update the Q-network parameters (θ). The loss function L(θ) is given by,

$$L(\theta) = \mathbb{E}\left[\left(y(t) - Q\left(s(t), a(t); \theta\right)\right)^2\right]$$

3 Double Deep Q-Learning

Q-learning algorithm has the problem of overestimating action values. In order to overcome this problem, double Q-learning was introduced with the primary idea of having two Q-tables, primary and target. Of these, one table is used for obtaining the best action whereas the other is used for evaluating the action. Similarly, the function approximation version of Q-learning, a.k.a. deep Q-learning also suffers from the same problem of overestimation and double deep Q-learning was introduced to achieve must better convergence stability [3]. The algorithm works exactly as the previously described deep Q-learning algorithm, except for the parts where the action selection is done and where it is evaluated. The only difference is that the best action in the next state is chosen from the primary network (θ) and the target value is calculated from the target network (θ^-) is found using the rule,

$$y(t) = r(t) + \gamma Q\left(s'(t), \arg\max_{a'} Q(s'(t), a'; \theta); \theta^-\right)$$

The primary Q-network is the online network and the agent always chooses this network to find the best action in a state. The target Q-network is the offline network and the agent chooses this network only to evaluate the actions. The weights of the target Q-network are updated periodically and set to the weights of the primary Q-network. The number of steps that defines when weights of the target Q-network must be updated is also a hyperparameter of the double deep Q-learning algorithm. Besides, loss function remains the same as above and any of the aforementioned optimizers could be used to train the primary Q-network.

4 Environment

For this project, two OpenAI Gym environments - CartPole-v0 and BreakoutNoFrameSkip-v4 are used. Cartpole consists of a pole attached to a cart by the means of an un-actuated joint. The cart is placed on a frictionless track and capable of moving along it. The aim is to prevent the pole from falling off the cart by controlling its movement [4]. Figure 2 shows the environment with the cart and the pole. The state in this environment is represented using four real values – cart position, cart velocity, pole angle and pole velocity at tip. Further, there are two possible actions in each state – push cart to the left (action 0) or to the right (action 1). Reward is +1 for every step taken before an episode ends, except for the termination step which gives a reward of –1. In this environment, an episode terminates if either of the following three conditions is satisfied:

i. episode reward is equal to 200
ii. pole angle is more than 12° away from the vertical on either side
iii. cart position is more than 2.4 units away from the center on either side

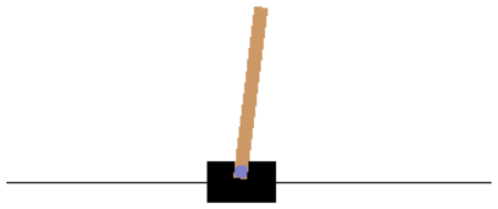
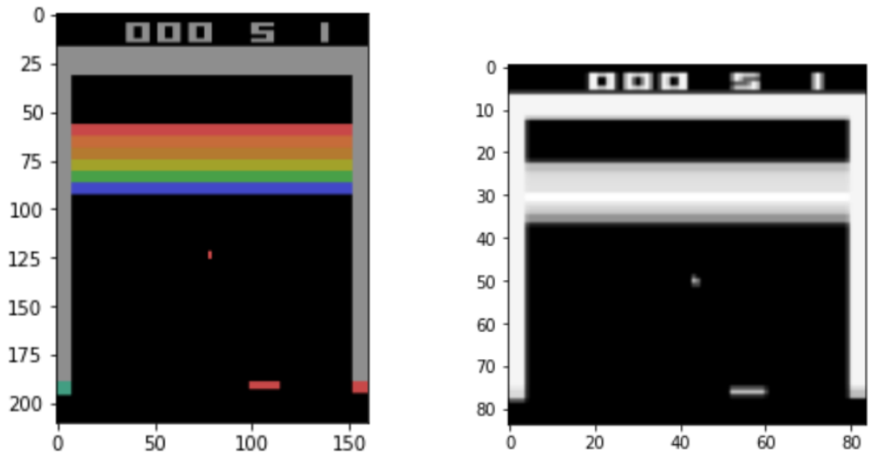


Figure 2: A state in the CartPole-v0 environment of the OpenAI Gym

The other environment used for this project simulates the classic video game Atari Breakout. It consists of a ball, several layers of bricks and a horizontally movable pad as shown in Figure 3(a). The aim is to destroy all the bricks, while ensuring the ball does not go beyond the screen. The pad has to be moved to ensure the ball does not go beyond screen [5]. The default size of the screen returned by the environment is of shape 210 × 160 × 3. However, parts of the returned pixels are redundant and the color of the screens is also not essential. Hence, the environment screen is processed to get a grayscale image of a moderate size of 84 × 84 as shown in Figure 3(b). It can be observed that the most important information of the frame (locations of the ball and the moving pad) is retained in the resized image, making this preprocessing an efficient technique to reduce the computations involved in training an agent on this environment. These image pixels are used to represent state in this environment.



(a) (b)

Figure 3: Initial state of the Breakout environment depicting the ball, reward bricks, moving pad, score and number of remaining lives, as (a) an RGB image of dimensions (210, 160, 3) and (b) a compressed grayscale image of size (84, 84)

However, in this environment, a single frame is not sufficient to determine the best possible action, since it only indicates the position of the ball but not its direction. Thus, the frame prior to the current state is also required to make this decision. Even though this screen provides the direction in which the ball is moving, it does not account much for the ball's acceleration. Hence, following the model used by DeepMind in training the Atari games, four frames are used to represent the current state. This is emphasized in Figure 4, where the rightmost image corresponds to the current frame, and the leftmost image corresponds to the oldest frame. These frames suggest that the ball is moving downwards to the left.

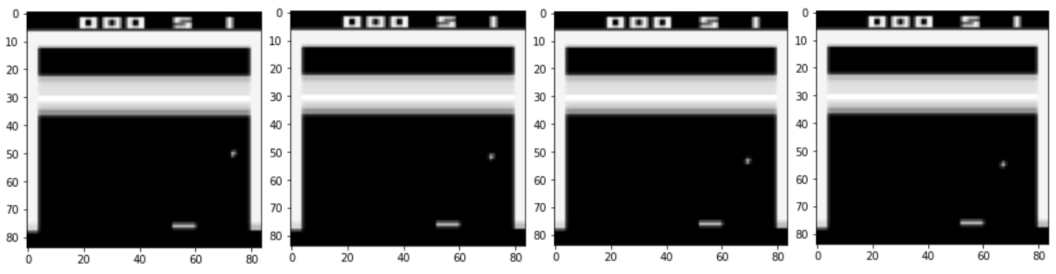


Figure 4: A single state in the Breakout environment consisting of the current frame and the three previous frames, starting from the oldest (left) to the newest (right)

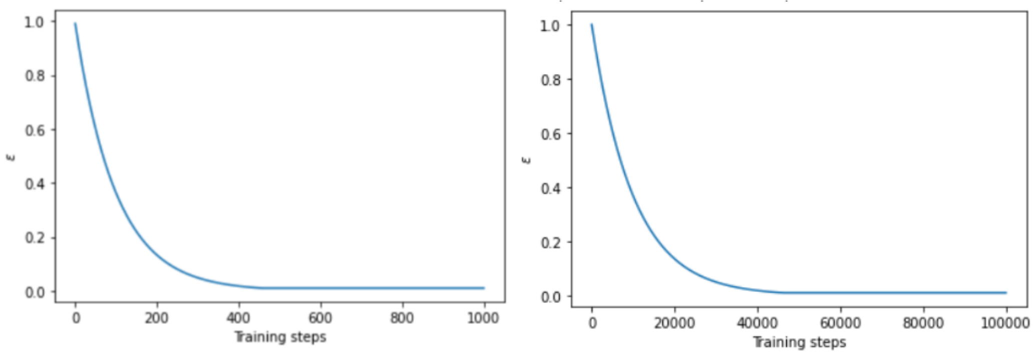
Further, there are four possible actions in each state of the Breakout environment –

i. Action 0: NOOP - do nothing, do not move the pad
ii. Action 1: FIRE - launches the ball, required to start the game
iii. Action 2: RIGHT - move the pad to the right
iv. Action 3: LEFT - move the pad to the left

In addition, rewards in this environment are earned by destroying the bricks. Specifically, destroying the bricks in the upper layers earns much more reward than destroying those in the lower layers. The reward for destroying a brick in the first two layers from the pad is +1, whereas destroying bricks in the subsequent layers towards the top of the screen are +4, +4 +7 and +7 respectively. There are no negative rewards in this environment and there is a zero reward if the moving pad does not hit the ball or the ball does not hit a brick. Besides, even though the original game lasts for 5 lives, for the implementations in this project, a breakout game with only 1 life is considered. Thus, an episode terminates when a single life is lost. This setting is adjusted using the Atari Preprocessing wrapper developed by OpenAI Gym, and available under the Gym's wrapper module.

5 Implementation and Results

For action selection, ϵ-greedy strategy is used by the agents in both vanilla and double deep Q-learning implementations. This is done to ensure proper exploration of the state space by making more random actions in the initial stages of training. For this purpose, the value of ϵ is made to start from 1 and decrease exponentially over time as shown in Figures 5(a) and 5(b). However, since the number of actions in BreakOut are more than that in CartPole, the ϵ decay rate in the later is higher, i.e., the value decreases very slowly. This decay is stopped when the exploration probability reaches 0.01. For all the subsequent training steps, the exploration value is fixed to a constant value of 0.01. A decay constant of 0.99 is chosen for CartPole, whereas a decay constant of 0.9999 is chosen for implementations on BreakOut environment. Besides, the discount factor is set to 0.9 in all the simulations.

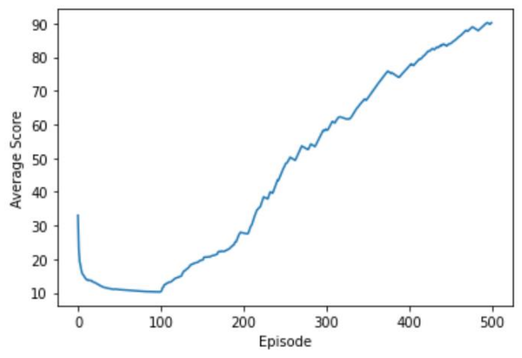


(a) (b)

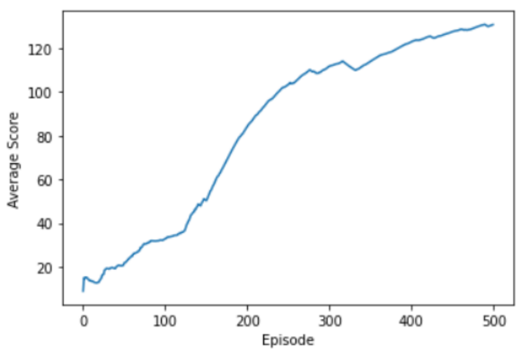
Figure 5: Exponentially decaying exploration probability for (a) Cartpole and (b) BreakOut

Firstly, both the deep reinforcement learning algorithms are implemented on the CartPole environment. A 2-layer fully connected neural network with dense layers and 'tanh' activation function is used for this purpose. The output layer has a linear activation function. The number of nodes in the hidden layer is chosen to be 10. A single experience is used at each instant to optimize the neural network parameters in a manner similar to stochastic gradient descent in supervised learning. Adam optimizer with learning rate 0.01 is used as the neural network optimizer after it was found that both RMSProp and SGD optimizers had poor performance on the network. Besides, mean squared error is the objective function.

Initially, the simulations are performed for 500 training episodes and the average rewards in both cases are shown in Figures 6(a) and 6(b). Both the average reward plots follow an overall increasing trend, indicating that the two agents are learning the underlying environment over time and the Q-network is being trained correctly. It can also be observed that the average reward earned by the DQN agent after 500 episodes is about 90, whereas that earned by the DDQN agent is about 130. This clearly shows the better performance of the DDQN algorithm in terms of the learning speed. Introducing a target network has enabled the agent understand the better actions in a quicker way. Besides, after finishing the training phase, the exploration probability of the two agents is set to zero and they are tested on the environment for 100 episodes. The average testing rewards are as shown in Figures 7(a) and 7(b). The reward obtained by the DDQN agent for each test episode is found to be 200 (maximum), whereas the reward obtained by the DQN agent is about 155 on an average.

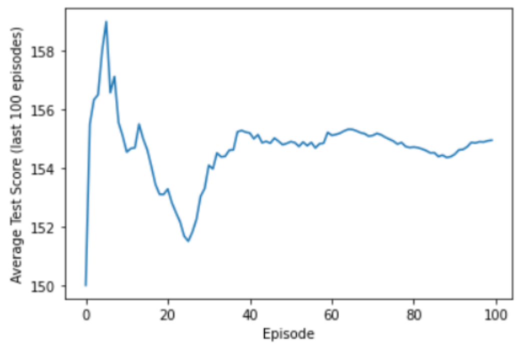


(a)

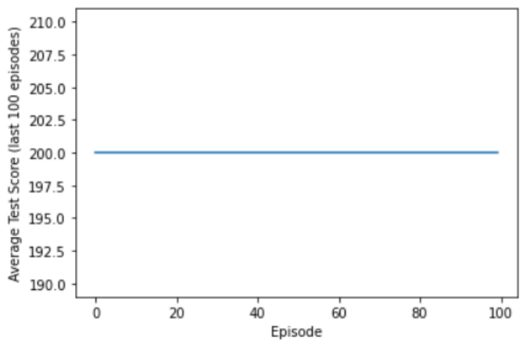


(b)

Figure 5: Comparing average reward per episode for 500 training episodes of CartPole for (a) deep Q-learning agent and (b) double deep Q-learning agent

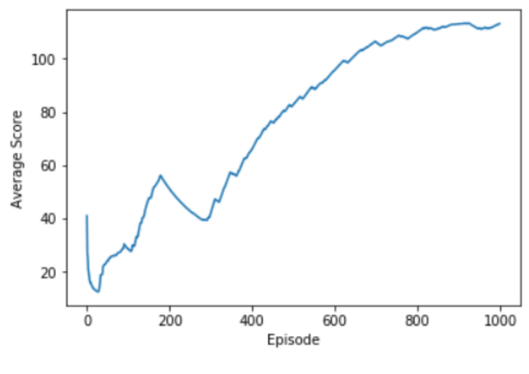


(a)

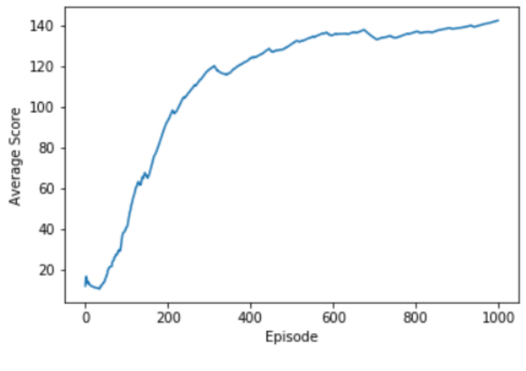


(b)

Figure 6: Comparing average reward per episode for 100 testing episodes of CartPole for (a) deep Q-learning agent and (b) double deep Q-learning agent



(a)



(b)

Figure 7: Comparing average reward per episode for 1,000 training episodes of CartPole for (a) deep Q-learning agent and (b) double deep Q-learning agent

Further, the number of training episodes is increased to 1,000 and it is observed that the performance of both the agents remained similar. The average rewards earned by the agents still follows a similar trend as shown in Figures 7(a) and 7(b). There is still a difference of 30 in the average rewards earned by the DQN and the DDQN agents. However, both the agents attained maximum reward on the 100 test episodes. Besides, the average reward over the last 100 episodes is also plotted for the two agents in Figures 8(a) and 8(b). Since an SGD-like optimization technique is used, an occasional decrease in the average reward is expected. However, the graph in the case of deep Q-learning agent looks to be slightly more susceptible to these changes in the network as compared to the double deep Q-learning agent. This suggests that the network stability in double deep Q-learning with two Q-networks is slightly better than when using a single Q-network in deep Q-learning.

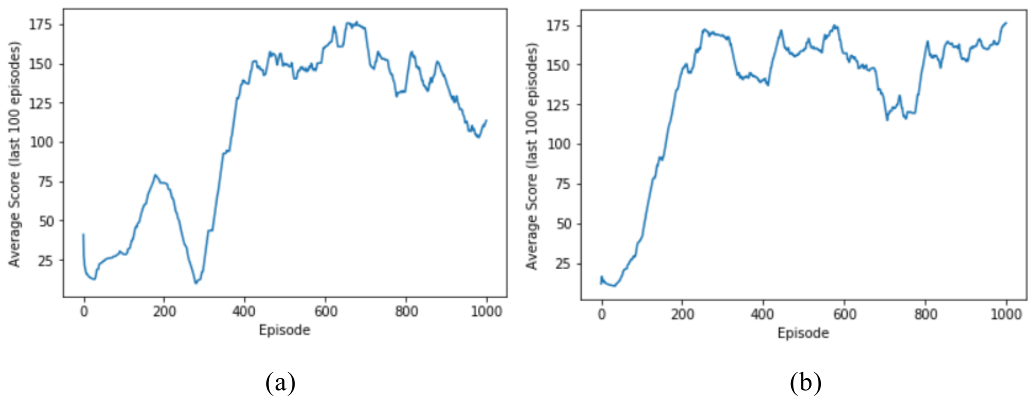


Figure 8: Comparing 100-episode average reward for 1,000 training episodes of CartPole for (a) deep Q-learning agent and (b) double deep Q-learning agent

Secondly, both the deep reinforcement learning algorithms are implemented on the BreakOut environment. A 4-layer convolutional neural network with 'relu' activation function is used for this purpose. The penultimate layer of the network is a dense layer and it is connected to the output layer that has a linear activation function. The neural network architecture is based on the neural network implemented by DeepMind in [6]. The first layer has 32 filters of size 8 × 8, the second layer has 64 filters of size 4 × 4 and the third layer also has 64 filters, but of size 3 × 3. The outputs of the third layer are flattened and connected via a dense layer to the output layer with 4 nodes, corresponding to the 4 actions in the environment. Experience pool of size 100,000 is used to sample a minibatch of 8 examples at each training step. Adam optimizer with learning rate 0.00001 is used as the neural network optimizer. Mean squared error is once again used as the objective function.

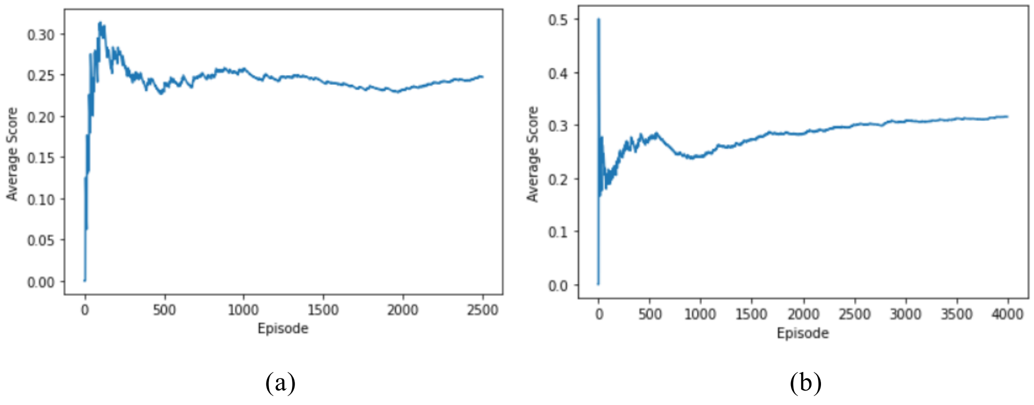
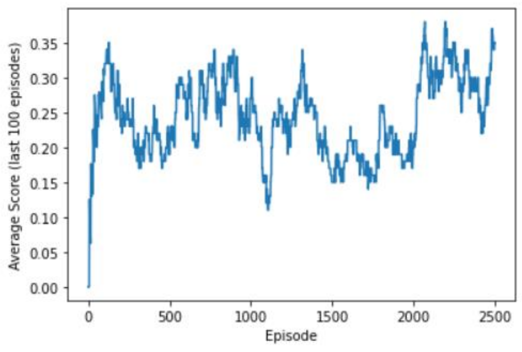
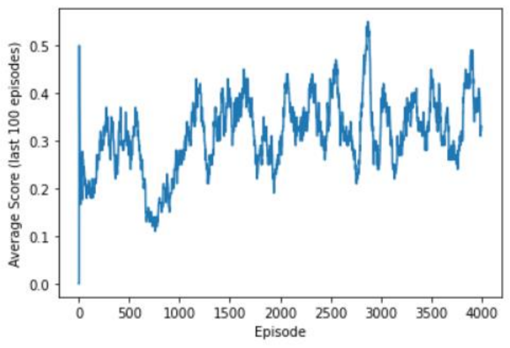


Figure 9: Comparing average reward for training episodes of BreakOut for (a) deep Q-learning agent and (b) double deep Q-learning agent

The simulations are performed for several training episodes and the average rewards in both cases are shown in Figures 9(a) and 9(b). The average reward after so many episodes is still only about 0.3. This indicates that the Q-networks are being trained towards the target values very slowly and consequently, the agents' learning is happening at a slow pace. Besides, unlike the previously discussed CartPole environment, there is no obvious improvement in using the double deep Q-learning algorithm. This suggests that many more training steps are required for the agents to completely learn the underlying environment. Also, the 100-episode average rewards are plotted in Figure 10. Both these reward plots do not give any significant insights into how the algorithms are performing on the BreakOut environment.



(a)



(b)

Figure 10: Comparing 100-episode average reward for training episodes of BreakOut for (a) deep Q-learning agent and (b) double deep Q-learning agent

In order to improve the learning speed of the algorithms, different convolutional neural network architectures were tried by introducing max pooling and dropout layers into the network. The number of filters in each layer was also varied to no avail. Alternatively, the size of the image pixels was also reduced to size 40 × 40 in the hope that a reduction in the size of the input would mean requiring a smaller network with a smaller number of parameters. In addition, using minibatch of sizes 16 and 32 would help in the faster convergence of the algorithm, albeit at the cost of an increase in the required time for implementing each epoch. Therefore, the network could not be trained further and better.

6 Summary

In this project, two novel reinforcement learning algorithms, deep Q-learning and double deep Q-learning, were implemented using Keras library in Python. These algorithms were used to make reinforcement learning agents understand and solve two OpenAI environments, CartPole and BreakOut. The former has a real valued state space, whereas the latter has image pixels as the state space. To train the models on these environments, a fully-connected neural network was used for the former, whereas a convolutional neural network was used for the latter. While training the agents on the environments, both the overall average rewards and the latest 100-episodes average reward were plotted. It was observed that the performance of the double deep Q-learning based agent was superior to that of the other agent, proving that using a target Q-network can significantly impact the agent's learning. Finally, upon completion of the learning phase, the agent's exploration was turned off and its performance was observed for 100 test episodes.

References

[1] Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction", Second Edition
[2] CSE 510 Introduction to Reinforcement Learning Slides by Alina Vereshchaka
[3] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." *Thirtieth AAAI conference on artificial intelligence*. 2016.
[4] https://github.com/openai/gym/wiki/CartPole-v0
[5] https://gym.openai.com/envs/Breakout-v0/
[6] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.
[7] Acknowledgement: Few parts of this project report are based on my Bachelors' final year project which was on the topic of deep reinforcement learning and my CSE 510 RL Project 1.