PROJECT 2 – VALUE FUNCTION APPROXIMATION

Vamshi Krishna Gujjari
vamshigu@buffalo.edu

Part 1 – Implementing DQN

From the DeepMind's paper [mnih2015human] and [mnih-atari-2013]), Deep Q Network is implemented, following the same architecture and hyper parameter values using **Pytorch**. The implemented Deep Q network Agent is then tested on two of the Open AI gym Atari environments viz., **Pong** and **Breakout**. The observed results were satisfactory, where the agent was able to achieve high scores equal to normal humans.

I. Environments

1. **Pong:** In this Open AI gym Atari environment, each observation is an RGB image of size (210, 160). But to make the computations easier, these images are resized to (84,84) and made to grayscale, as the colors in any Atari environment will not make any difference to the gameplay and making the images to grayscale also reduces the training costs significantly. There are 6 actions possible for the agent to perform in this environment viz., ['NOOP', 'FIRE', 'RIGHT', 'LEFT', 'RIGHTFIRE', 'LEFTFIRE']. At each step, the agent performs one of these 6 actions and tries to maximize the reward. Each time the opponent fails to hit the ball, agent gets a reward of +1. When the agent fails to hits the ball, it gets a reward of -1. The reward will be 0 for all the other cases.

2. **Breakout:** In this environment, each observation is again an RGB image of size (210,160). The same preprocessing which was done to the Pong environment is done to this environment. The number of possible actions for Breakout is 4 viz., ['NOOP', 'FIRE', 'RIGHT', 'LEFT']. At each step, the agent performs one of these 4 actions to maximize reward. Each time the ball hits a block, a reward of +1 will be awarded to the agent and **when the agent fails to catch the ball, a reward of -1 is awarded.** In all the other cases, a reward of 0 is awarded.

II. Agent

Two types of agents are used for this environments, Deep Q network and Double Deep Q network.

- **Deep Q Network**: Q Learning is combined with Deep Neural Networks to form a Deep Q network. In this deep neural networks, there are several layers of nodes in between the input and output layers called hidden layers, which makes it possible for the network to learn directly from raw sensory data. In this architecture, Convolution neural networks are used, which uses hierarchical layers of tiled convolution filters to mimic the effects of receptive fields. This network is used to approximate the action-value (Q) function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

The architecture of the Deep Q network is shown below,

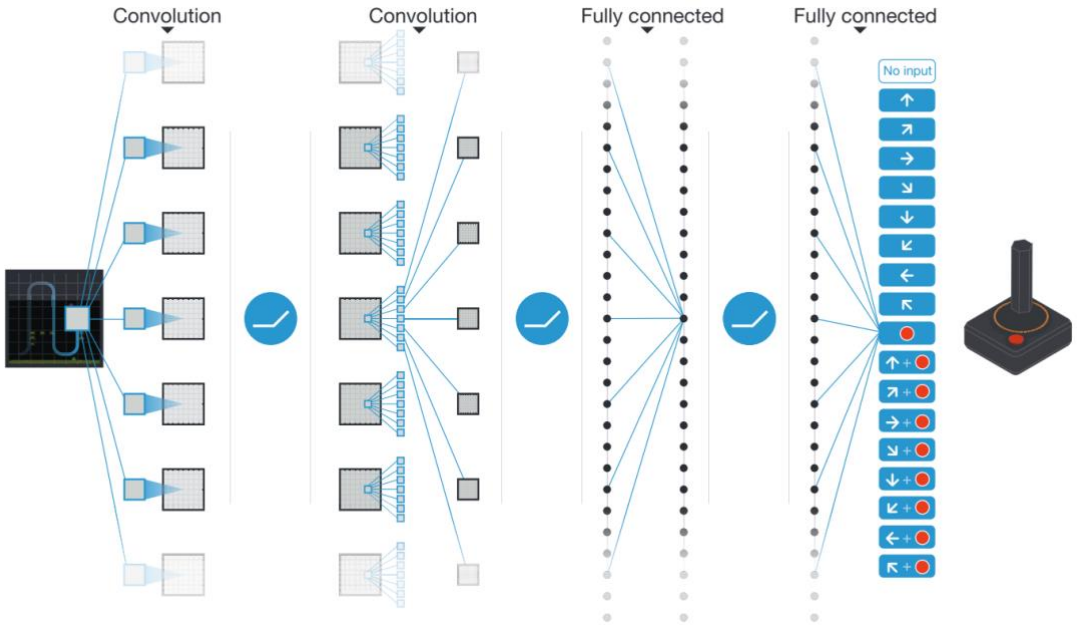

Fig 1: Deep Q Network Architecture

Preprocessing Images: The images in the shape of (210, 160) are resized to (84,84) as it can be demanding in terms of computation and memory requirements. Then these images are stacked in sets of 4 most recent frames and sent as an input to the network.

Architecture: There are 4 hidden layers excluding the input and output layer, where the first hidden layer convolves 32 filters of 8x8 with stride 4 with the input image and applies ReLu activation. The next layer convolved 64 filters of 4x4 with stride 2 with a ReLu activation. Third layer convolves 64 filters of 3x3 with stride 1 followed by another ReLu activation. The next layer is fully connected with 512 nodes, which will produce the output layer, which contains a single output for each valid action.

- **Experience Replay:** In the DeepMind's paper, a technique called experience replay is used, where the agent's experiences are stored at each time step into a replay memory. Samples from this memory are drawn at random form the pool of stored samples. There are several advantages of using this replay memory, First, each step of experience is potentially used in the weight updates of the network, so this would increase the data efficiency. Second, learning from consecutive samples would be very inefficient, because of correlations between them. If these are randomized, that would reduce the variance of updates.
- **Target Network:** Instead of using just a normal Q network, a target network is used, which makes the algorithm more stable. In the normal Q network, when an update to our Q table, increases the $Q(s_t,a_t)$, it also increases the value for $Q(s_{t+1}, a)$ for all a, which increases the target value. This leads to divergence of the policy. But, when a target network is used, generating the updates happens with an older set of values, which would make the chance of oscillations more unlikely.

Part 2 – Improving DQN

Double DQN:

The given vanilla DQN can be further improved, to get faster and better results. In order to improve it, Double DQN method is used. In vanilla DQN, the Q value estimates are calculated based on the maximum Q values of the next states, this would lead to overestimation, which can be tricky at times. In order to reduce the risk of overestimation, two independent Q estimators are used, such that the action will be selected based on an estimator and updates are done to a different estimator. The second estimator updates are done periodically and not at each step. This way, overestimation can be reduced. Expression for Double Q learning is given below.

$$Q^*(s_t, a_t) \approx r_t + \gamma\, Q(s_{t+1}, \text{argmax}_{a'} Q'(s_t, a_t))$$

RESULTS

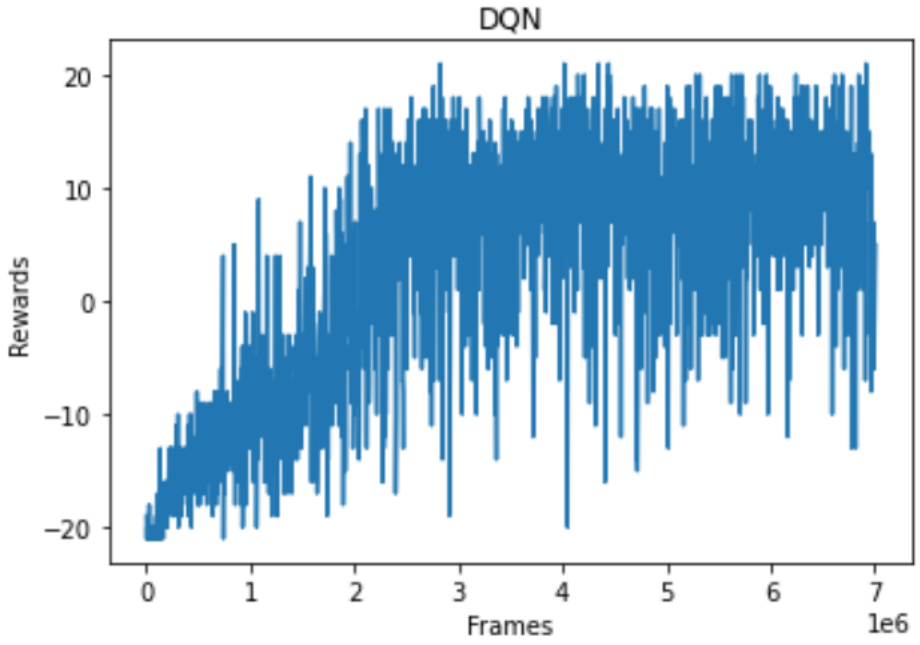

Fig 2: Pong DQN results

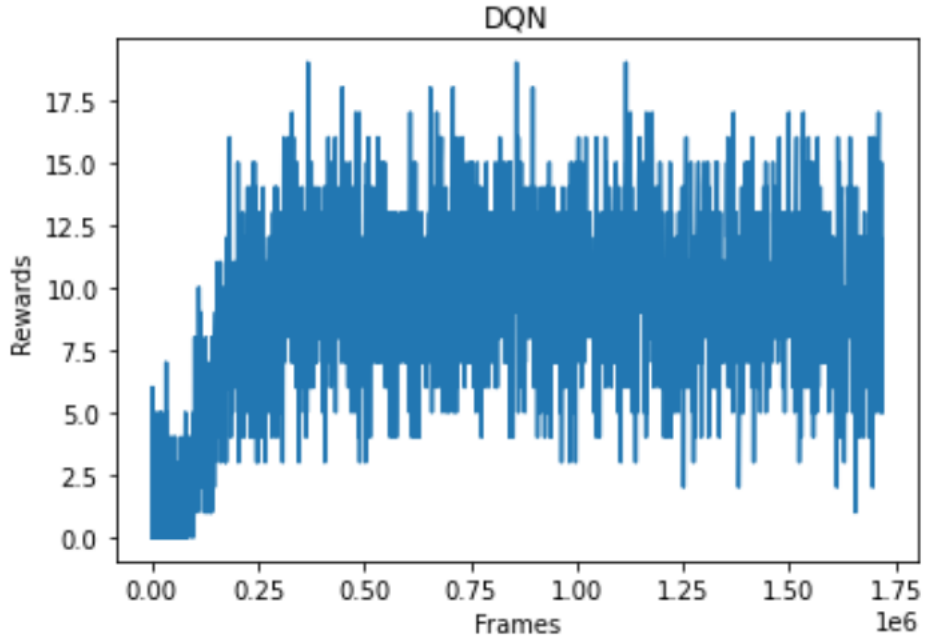

Fig 3: Breakout DQN results

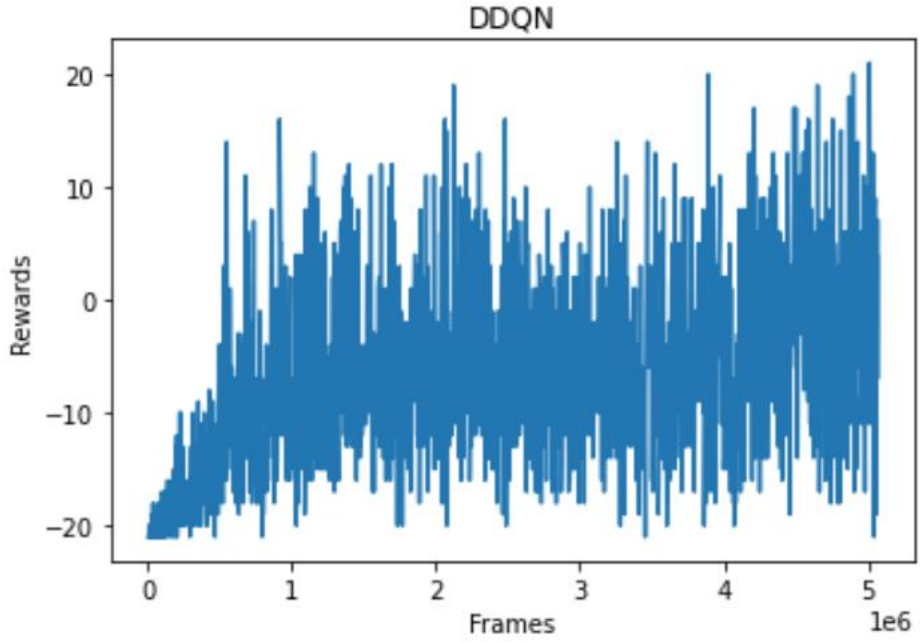

Fig 4: Pong DDQN results

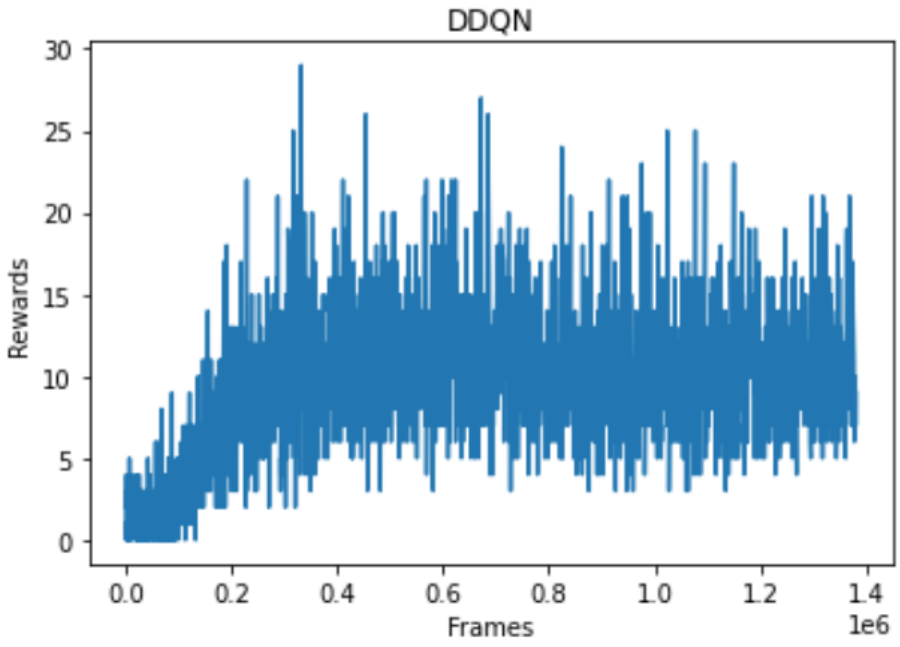

Fig 5: Breakout DDQN results

The graphs given above represent the number of frames vs reward obtained for both the environments, Pong and Breakout, for both DQN and DDQN. Due to some technical issues (GPU memory), Breakout was not able to get trained for the required number of episodes, to get maximum score possible, but Pong was able to achieve the highest possible score in both DQN and DDQN as it was comparatively a simpler environment. From the graphs given above, we can say that the agent started winning games (getting positive rewards) faster in DDQN compared to DQN. This means convergence has started faster in the case of DDQN, compared to DQN.

References:

1. https://github.com/adamtiger/DQN/tree/f4aa6e9fb31ac72568f20509097c9981c77b73e4
2. https://pytorch.org/tutorials/beginner/saving_loading_models.html
3. https://pytorch.org/docs/stable/nn.html
4. https://pytorch.org/cppdocs/api/classtorch_1_1nn_1_1_conv2d.html
5. https://discuss.pytorch.org/t/how-to-convert-array-to-tensor/28809
6. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
7. https://github.com/chaitanya100100/RL-Algorithms-Pytorch/tree/master/DQN_DDQN
8. https://www.neuralnet.ai/coding-a-deep-q-network-in-pytorch/
9. https://www.youtube.com/watch?v=teDuLk3cIeI
10. https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f
11. https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/
12. https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf