Raunaq Jain
50320026

Reinforcement Learning

Assignment 2

A reinforcement learning system can be depicted in the following way –

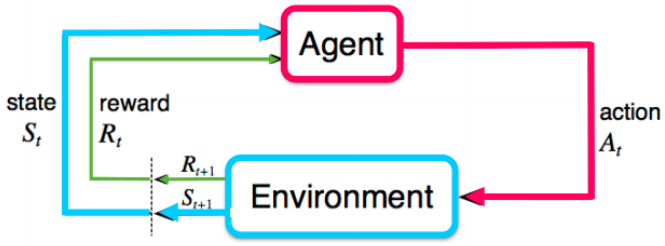

Figure 1

The system can be defined using a Markov Decision Process which is defined by tuple (S, A, P, R, γ), where

- S is the set of states, which characterizes the configuration of the environment.
- A denotes the actions the agent can take.
- R is the reward function.
- P is the state transition probability distribution.
- γ is the discount factor.

Q-learning algorithm was to go to method to solve reinforcement learning tasks but with high-dimensional and complex tasks, Q-learning algorithm breaks down and hence we use deep neural networks to solve these tasks.

Deep Q-Network was introduced in Playing Atari with Deep Reinforcement Learning on NIPS in 2013. It is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics.

DQN overcomes unstable learning by mainly 4 techniques –

1. **Experience Replay**
   Deep neural networks (DNN) can easily overfit on data if not correctly propagated. Moreover, input to the DNN model is a frame and therefore, the training of the model would become incorrect because the frames would be passed in a sequential order which will make the model biased.
   To solve this problem, Experience Replay stores experiences including state transitions, rewards, and actions, which are necessary data to perform network update. Experience Replay has the following benefits –
   - Reduces correlations between experiences while updating DQN
   - Increases learning speed with mini-batches
   - Reuses past transitions to avoid forgetting rare experiences.

2. **Target Network**
   For the training of the DQN model, we need target values and predicted values. Computing both the values from a single model runs into the issue of fluctuating target (as the DQN is being updated). Unstable target function makes training difficult. Therefore, a target network is created which remains fixed for a number of steps before being updated with the current model.

3. **Clipping Rewards**

   According to Human-level control through deep reinforcement learning in Nature in 2015 –

   "We also found it helpful to clip the error term from the update [...] to be between -1 and 1. Because the absolute value loss function |x| has a derivative of -1 for all negative values of x and a derivative of 1 for all positive values of x, clipping the squared error to be between -1 and 1 corresponds to using an absolute value loss function for errors outside of the (-1,1) interval. This form of error clipping further improved the stability of the algorithm."

   Clipping rewards improves the stability of the algorithm because in deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients. These in turn result in large updates to the network weights, and in turn, an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

4. **Skipping Frames**

   Games render at 60 frames per second and the agent doesn't need to take actions in every frame. So skipping frames is a technique where DQN calculates Q values every 4 frames and takes 4 frames as input. This reduces computational cost and gathers more experience.

Environment 1 – CartPole

The Cartpole environment characteristics for the training are as follows:

- **Goal** – The goal of the environment is the maintain the cartpole in the vertical position for as long as possible.
- **State** – It's a $3 \times 40 \times 84$ dimensional image frame formed from the difference of the current frame and the last frame.
- **Actions** – There are two possible actions to be taken – Left and Right
- **Reward** – The reward is +1 for every step that is not terminal

The training characteristics and parameters are:

- **Experience Replay**
  A memory size of 10000 is created. Experience (state, clipped reward, action taken, terminal) is added to memory after every environment step. While training, batches is formed from this memory itself.

- **Exploration-Exploitation**
  The ϵ value is decreased exponentially (Figure) till it reaches a threshold after which the value remains constant. Based on the ϵ, the action to be taken is computed.

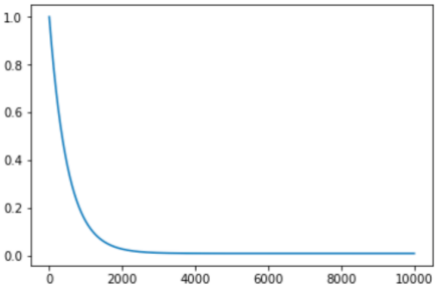

Figure 2 Epsilon Decay

- **Model Architecture**

```
DQN(
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(2, 2))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(32, 32, kernel_size=(5, 5), stride=(2, 2))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (head): Linear(in_features=448, out_features=2, bias=True)
)
```

Figure 3 DQN and DDQN model architecture

- **Vanilla Deep Q-Network**

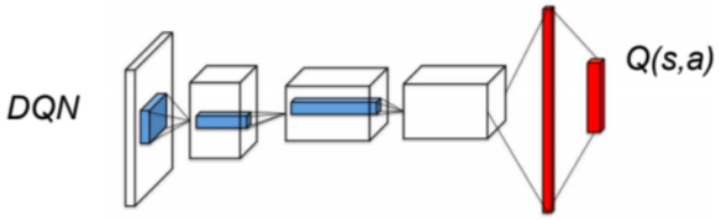

Figure 4 DQN model representation

- **Double Deep Q-Network (DDQN)**

The concept behind DDQN is same as Double Q-Learning that is a single estimator overestimates the Q-values and that the chances of two estimators overestimating at same action is lesser. Therefore, the Q-target:

$$r(s,a) + \gamma \max_{a'} Q_1(s', a')$$

is changed to Double Q-target:

$$r(s,a) + \gamma Q_2(s', arg\ \max_{a'} Q_1(s', a'))$$

DDQN can be used at scale to successfully reduce this over optimism, resulting in more stable and reliable learning. It uses the existing architecture and deep neural network of the DQN algorithm without requiring additional networks or parameters.

- **Observations and Remarks**

Comparing the Figure 5 and 7, it can be observed that the rewards in Figure 7 are more consistent, thereby depicting to us how Double DQN is more stable than DQN which was the reason for its selection. The same can be observed through their loss plots in Figure 6 and Figure 8. The loss for DDQN is more stable and lesser than the DQN loss.

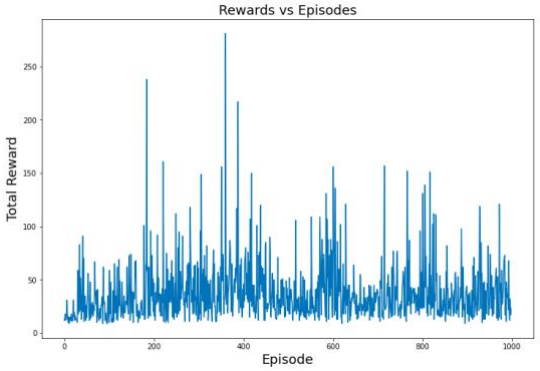

Figure 5 DQN Reward plot

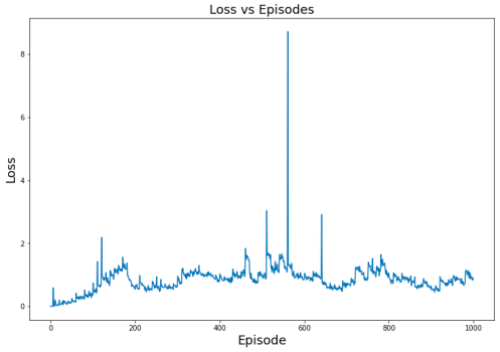

Figure 6 DQN Loss Plot

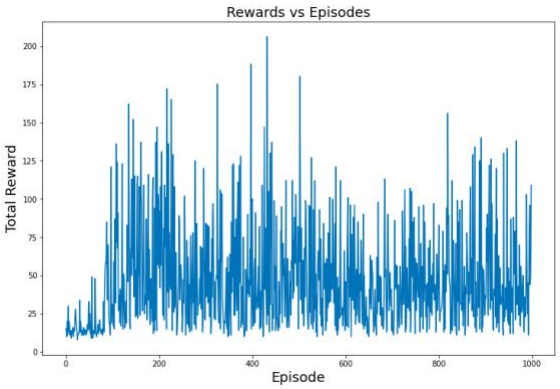

Figure 7 DDQN Reward Plot

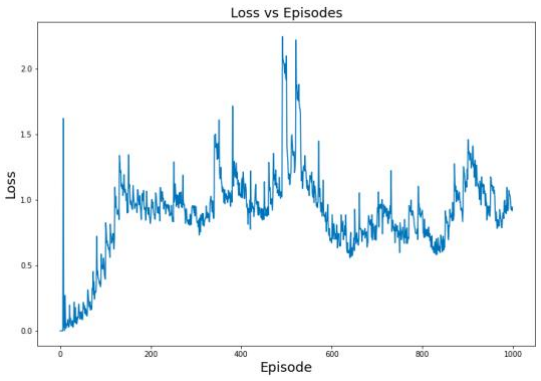

Figure 8 DDQN Loss Plot

Environment 2 – Atari Breakout

The Cartpole environment characteristics for the training are –

- **Goal** – The goal of the environment is to move the paddle so as to accumulate points by breaking cells.
- **State** – It's a $4 \times 84 \times 84$ dimensional image frame formed by stacking 4 continuous frames together.
- **Actions** – There are 4 possible actions to be taken – FIRE, NOOP, LEFT and RIGHT
- **Reward** – The reward is based on the cell destroyed.

The training characteristics and parameters are:

- **Experience Replay**
  A memory size of 1000000 is created. Experience (processed frame, clipped reward, action taken, terminal life lost) is added to memory after every environment step.
  While training, batches is formed from this memory itself.

- **Exploration-Exploitation**
  The ϵ value is computed based on the Figure 2. For the first 50000 frames, random actions are performed to fill the replay memory. Afterwards, ϵ is decreased linearly with a slope m till 1000000 frames after which the slope is changed to m', where $m' < m$. Based on the ϵ, the action to be taken is computed.

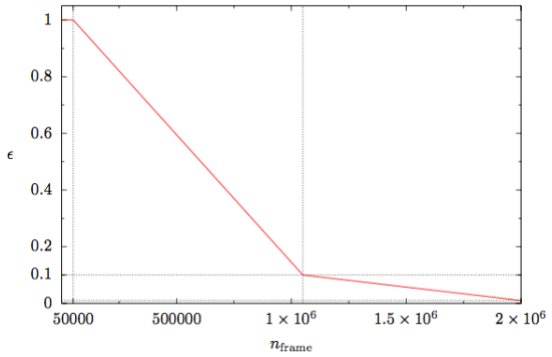

Figure 9 Epsilon Decay for Atari Breakout

- **Dueling Deep Q-Network**

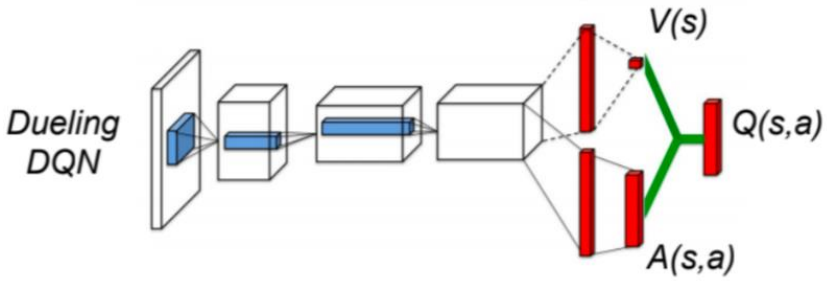

Figure 10 Dueling DQN model representation

In a normal DQN architecture (top network in the figure) the final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action. (see page 6 of Mnih et al. 2015)

"Intuitively, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. This is particularly useful in states where its actions do not affect the environment in any relevant way."

These outputs are the predicted $Q(s,a;\theta)$-values for action a in state s.
Instead of directly predicting a single Q -value for each action, the dueling architecture splits the final convolutional layer into two streams that represent the value and advantage functions that predict a *state value* $V(s)$ that depends only on the state, and *action advantages* $A(s,a)$ that depend on the state and the respective action.

- One stream of fully-connected layers output a scalar $V(s;\ \theta,\beta)$
- Other stream output an $|A|$-dimensional vector $A(s,a;\ \theta,\alpha)$

Here, θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers.

$$Q(s,a;\ \theta,\alpha,\beta)\ =\ V(s;\ \theta,\beta)\ +\ A(s,a;\ \theta,\alpha)$$

The problem with the above equation is that given Q we cannot recover V and A uniquely. Therefore to solve this, we subtract the advantage function estimator with its mean. This increases the stability of the optimization: the advantages only need to change as fast as the mean, instead of having to compensate any change.

- **Model Architecture-**

```
DQN(
  (features): Sequential(
    (0): Conv2d(4, 32, kernel_size=(8, 8), stride=(4, 4))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
  )
  (advantage): Sequential(
    (0): Linear(in_features=3136, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=4, bias=True)
  )
  (value): Sequential(
    (0): Linear(in_features=3136, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=1, bias=True)
  )
)
```

Figure 11 Dueling DQN Model Architecture

- **Observations and Remarks**

  Through Figure 12 and Figure 13, it's observable that Atari Breakout was able to be trained. We achieved a max reward of 346 after training on approximately 8,000,000 frames of data. Training Atari games are computationally expensive tasks and it took more than 24 hours of continuous training to achieve the below results where further improvements are still possible.

  Dueling DQN was chosen as the training algorithm because of its advantages over the vanilla DQN of being able to learn which states are (or are not) valuable, without having to learn the effect of each action for each state.

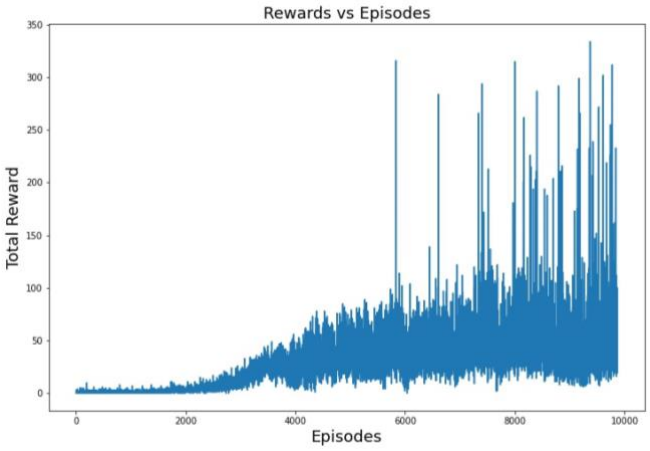

Figure 12 Atari Breakout Reward Plot

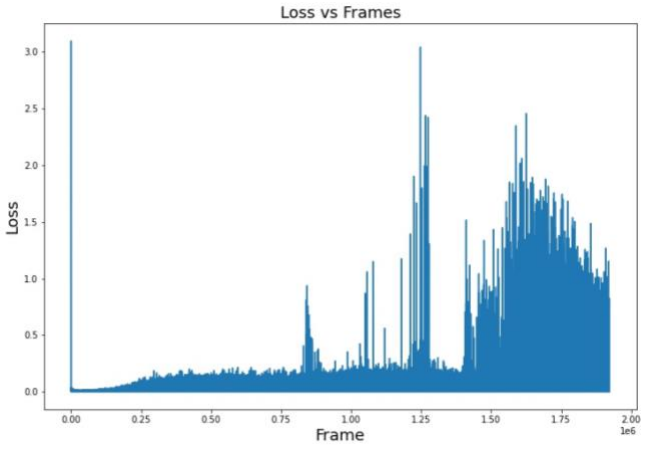

Figure 13 Atari Breakout Loss Plot