CSE574 Introducion to Machine Learning

By Prof: Sargur Srihari

Project 4 Bonus Report.

Submitted by :
Sai Varun Alapati
Ubit : saivarun
Personal Number :50290571

Table of Contents

1. Atari Enviromnment :

1.1 Breakout-v0

Aim is to maximize the score in the Atari 2600 game Breakout. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of k*k* frames, where k*k* is uniformly sampled from \{2, 3, 4\} {2,3,4}.

Sample Environment :

1.2 Learning using pre-implemented DQN function:

Experience replay will help us to handle three main things:

- Avoid forgetting previous experiences.
- Reduce correlations between experiences.
- Increases learning speed with mini-batches.

The main idea behind the experience replay is that by storing an agent 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 (memory capacity), 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.

The Algorithm used is

Algorithm 1 Deep Q-learning with Experience Replay

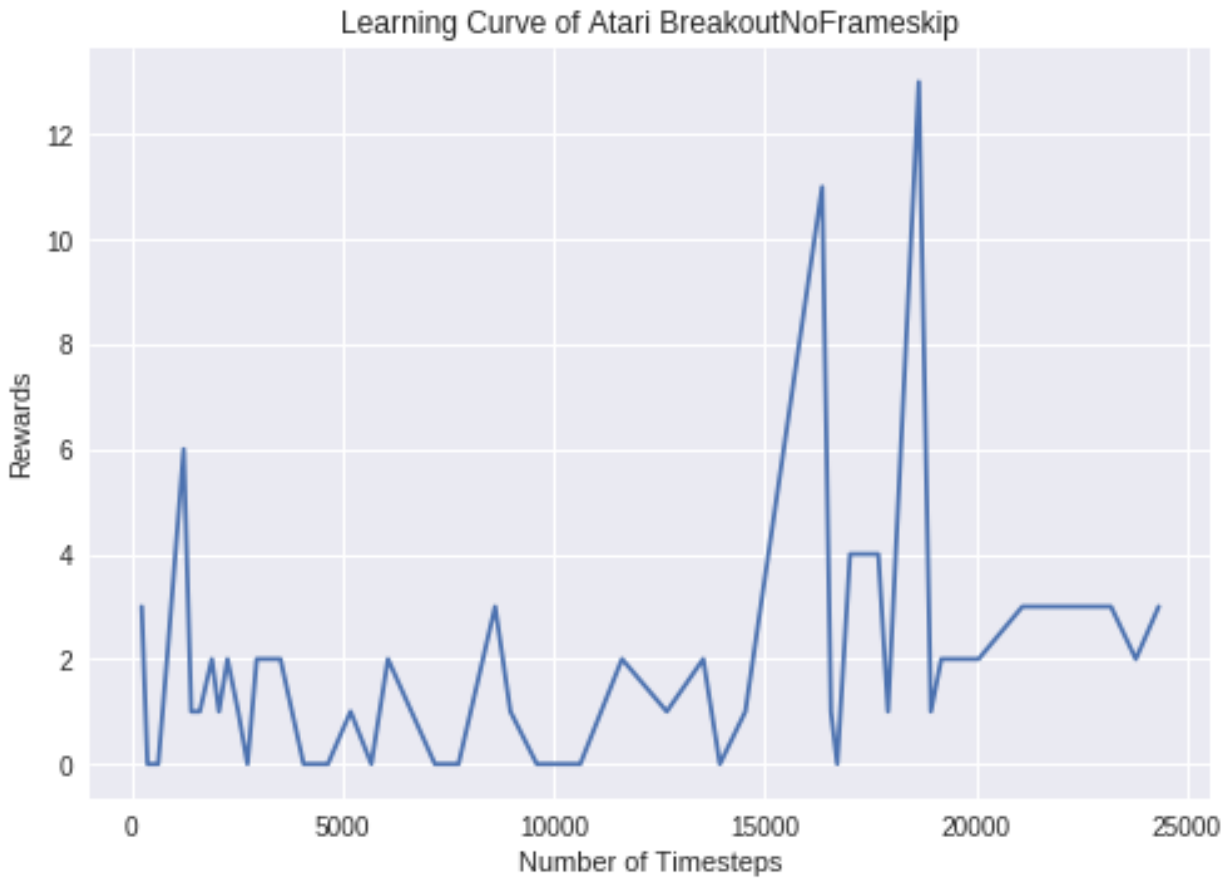
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

1.4 Experimental Results :

I have tested this Atarti Breakout-v4 for 25000 time steps (simply no. of steps) and the values for the reward values are

Timesteps	Best mean reward	Last mean reward per episode
842	-	1.00
1904	1.00	1.67
2966	1.67	1.50
3529	1.67	1.53
4653	1.67	1.28
5682	1.67	1.20
6076	1.67	1.24
7747	1.67	1.08
8977	1.67	1.15
9601	1.67	1.11
10631	1.67	1.07
11620	1.67	1.10
12681	1.67	1.10
13939	1.67	1.09
14542	1.67	1.09
14542	1.67	1.09
16719	1.67	1.33
17922	1.67	1.46
18937	1.67	1.73
19184	1.73	1.74
20068	1.74	1.74
21575	1.74	1.80
22893	1.80	1.85
23793	1.85	1.88
24334	1.88	1.90

The Value of the rewards are plotted below with respect to timesteps :



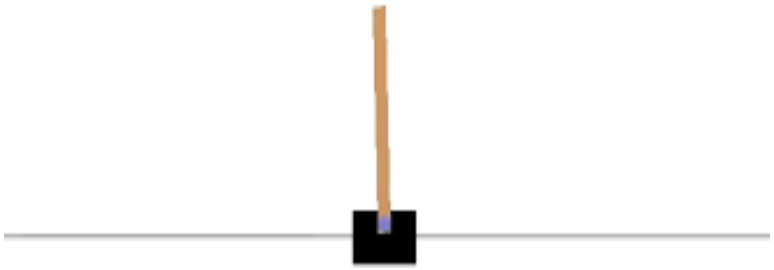
1.5 Observations :

This environment needs a minimum of 1milliion to stabilize and we couldn't find any stable path till the 25000 time steps.

2. Cartpole-v0 Environment :

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. 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.

Sample Environment :



2.2 Learning using pre-implemented DQN function:

Experience replay will help us to handle three main things:

- Avoid forgetting previous experiences.
- Reduce correlations between experiences.
- Increases learning speed with mini-batches.

The main idea behind the experience replay is that by storing an agent 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 (memory capacity), 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.

The Algorithm used is

Algorithm 1 Deep Q-learning with Experience Replay

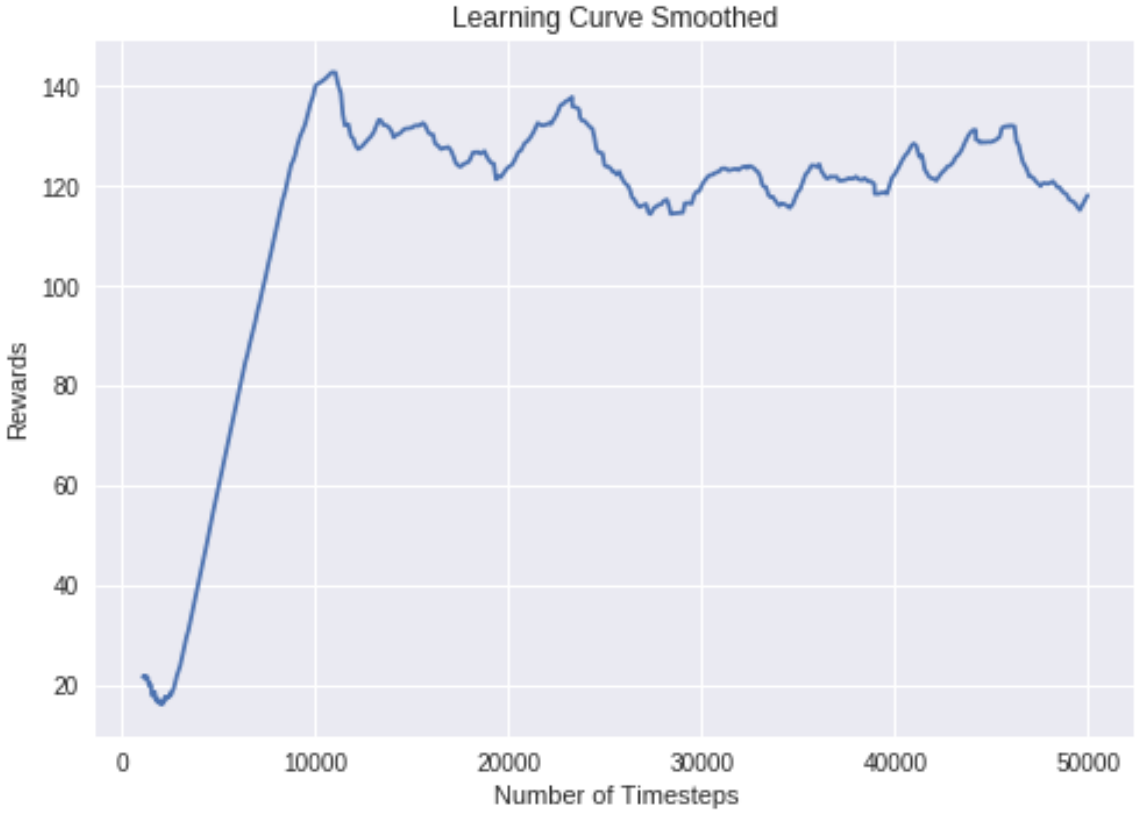
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

2.4 Experimental Results :

I have tested this Cartpole-v4 for 50,000 and 1,00,000 time steps (simply no. of steps) and the values for the reward values are plotted below with respect to timesteps :

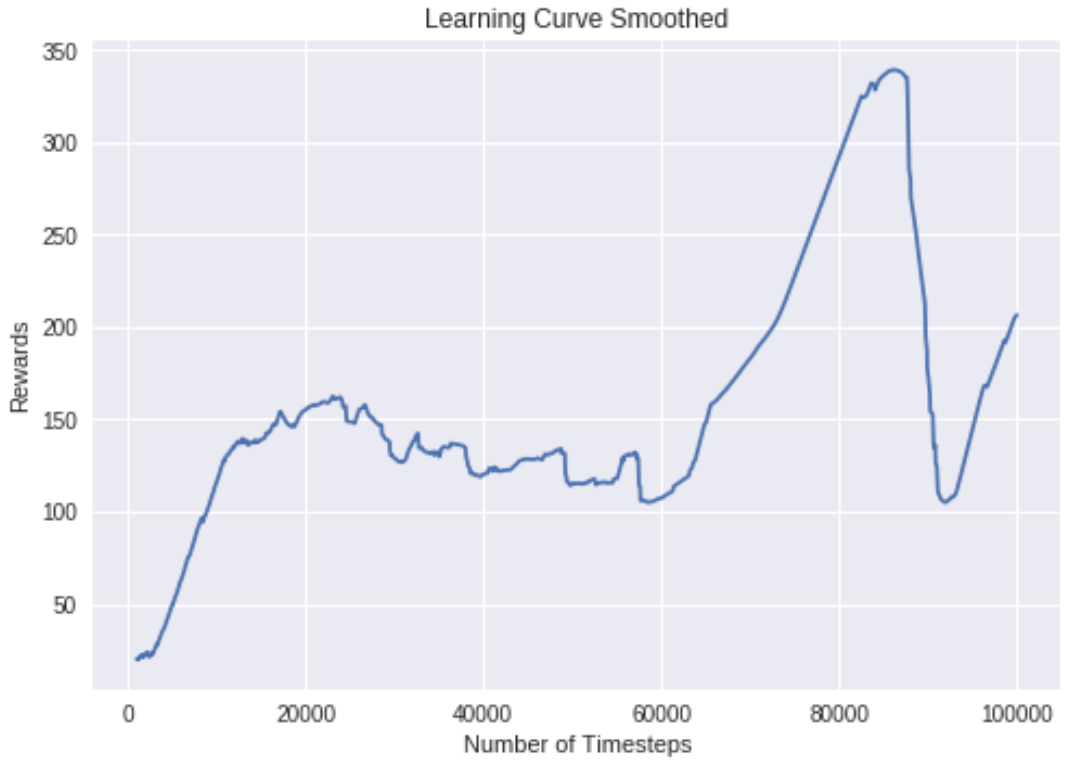
2.4.1 For 50000 Timesteps :

The smoothed Learning curve with respect to timesteps is

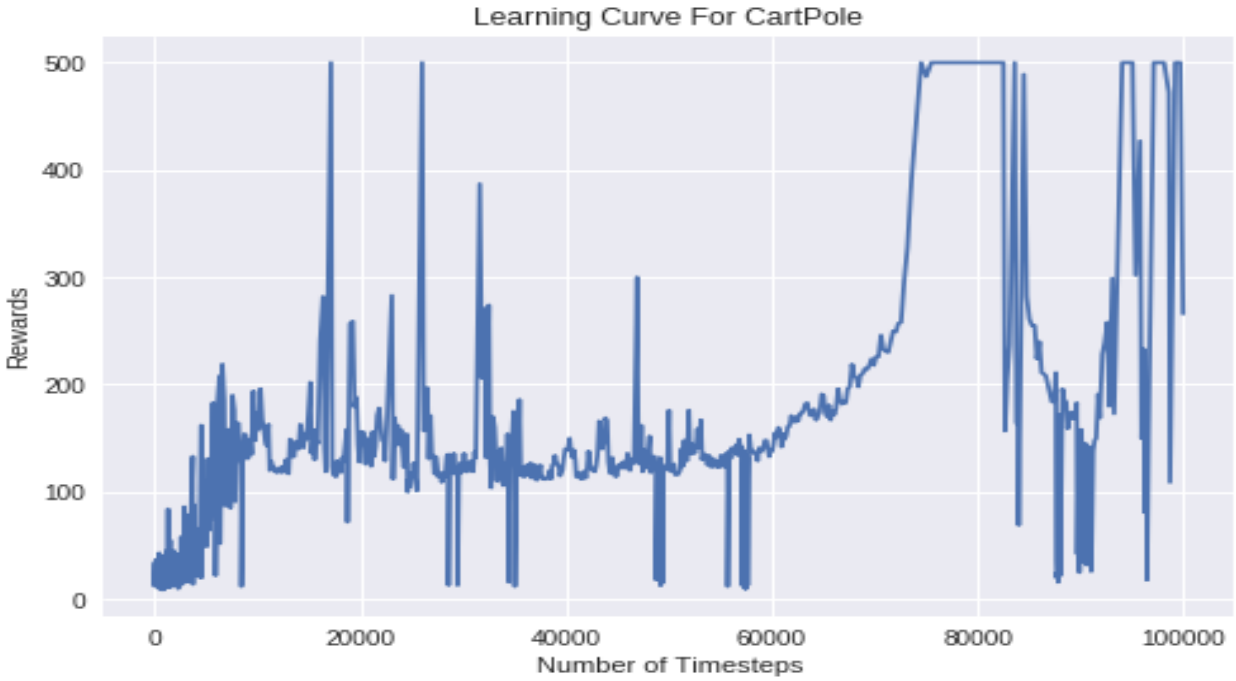


2.4.1 For 1,00,000 Timesteps :

The smoothed Learning curve with respect to timesteps is :



The non-smoothed Learning (rewards) curve is :



2.5 Observations :

We can observe that as the timesteps are increasing the reward values are increasing and the repesctive scores.

3 Conclusion and Summary :

- We have tried to implement the DQN algorithm for different environments and observed the rewards.
- We found that the Atari Breakout environment has lot of possibilities and hence taking huge timesteps to finding stabilized path compared to the Cartpole environment which has less possibities and hence the less timesteps.