The State University of New York Buffalo, New York 14260

CSE 574: Introduction to Machine Learning Project 3 Report Fall 2020

Sargur N. Srihari, Mihir Chauhan, Sougata Saha

Deep Q- Network (DQN) For Solving the Cart Pole Environment in Gym.

Hemant Kumar Das

50356421

hemantku@buffalo.edu

Introduction:

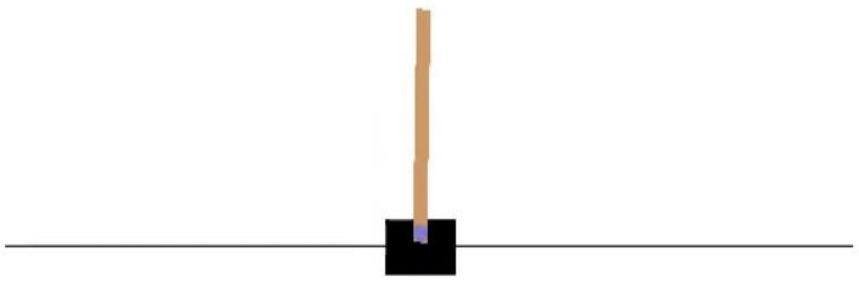
The task of this project is to implement a Deep Q-Network (DQN) for solving the Cart Pole environment in gym. We are required to implement and train a Q-Network using Pytorch, and solve the environment. We are also going to use an experience replay memory to store and sample experiences for training the DQN. Broadly, the project encompasses the following tasks:

1. Neural Network using Pytorch: Build a Neural Network using Pytorch to approximate Q values from a state.

2. Training the DQN using Pytorch: Train the DQN by implementing a target Q network which generates the training targets, and stores it in an experience replay buffer.

CartPole:

We will be using the OpenAI Gym environment CartPole where the object is to keep a pole balanced vertically on a moving cart by moving the cart left or right.



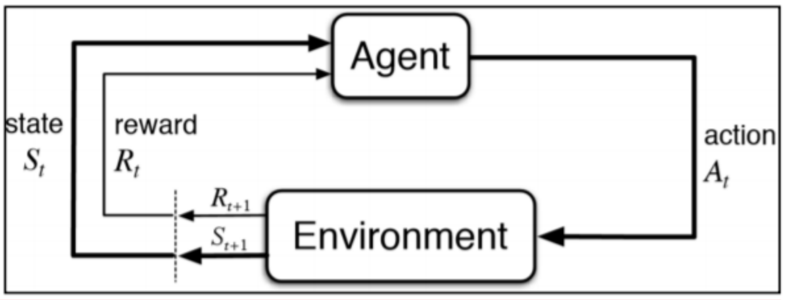
CartPole-v1

Environment:

A pole is attached by an unactuated 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 centre.

Reinforcement Learning:

Reinforcement learning places a program, called an *agent*, in a simulated environment where the agent's goal is to take some action(s) which will maximize its reward. In our CartPole example, the agent receives a reward of 1 for every step taken in which the pole remains balanced on the cart. An episode ends when the pole falls over.

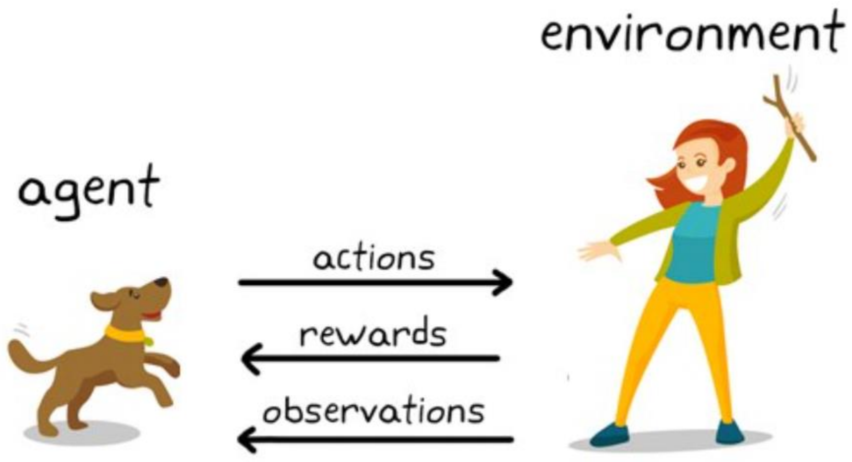


$$\boxed{S_0 \xrightarrow[R_0]{A_0} S_1 \xrightarrow[R_1]{A_1} S_2 \xrightarrow[R_2]{A_2} \ldots\ldots}$$

Goal: Learn to choose actions that maximize

$$\boxed{R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots\ldots, \quad \text{where } 0 \le \gamma < 1}$$

Autonomous agent must learn to perform a task by trial and error without any guidance from the human operator. Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards.



Plan of Work:

First task is to define the Deep Q-Network using Pytorch Sequential. Here, we have a one hidden layer with 128 units.

```
# Step 2.1: Define a Sequential network in Pytorch
self.layers = nn.Sequential(
  nn.Linear(num_inputs,128), #INPUT LAYER
  nn.ReLU(),
  nn.Linear(128,256), #HIDDEN LAYER
  nn.ReLU(),
  nn.Linear(256,num_actions), #OUTPUT LAYER
)
```

We also define a replay buffer of capacity 100000.
Replay Buffer which can store all the experiences of the agent.
During training, we will sample uniformly from the replay buffer and train our DQN.

```
replay_buffer = ReplayBuffer(100000)
num_frames = 15000
batch_size = 64
gamma = 0.99
```

'Gamma'(γ) is known as the discount factor. It is the measure of how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.

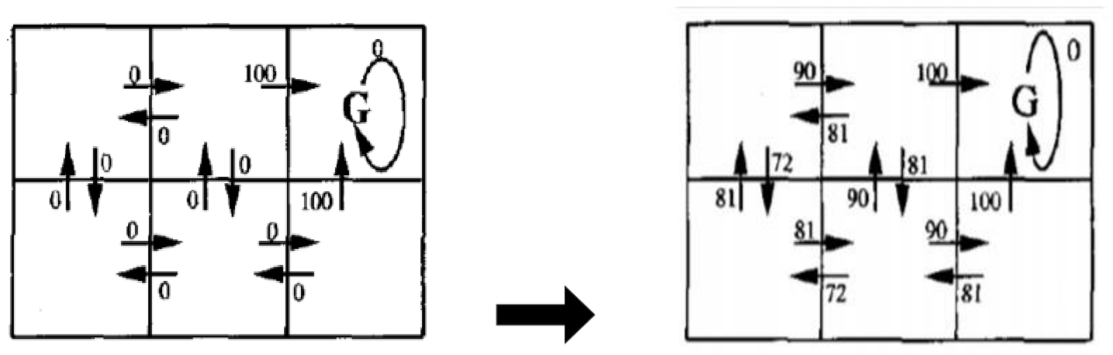
Below we have the Bellman's equation.

$$\boxed{Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a')}$$

The Q values is updated using an iterative approach. Using the following steps

```
Algorithm
• Initialize all table entries $\hat{Q}(s,a)$ to zero
• Observe the current state $s$
• Do forever
    – Select an action $a$ and execute it
    – Receive immediate reward $r$
    – Observe the new state $s'$
    – Update the table entry for $\hat{Q}(s,a)$ as follows
      $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
    – $s \leftarrow s'$
```

Below is an example of how Q value is updated after every iteration. This tells us how where to move and what steps to take in order to reach our goal in a grid word.



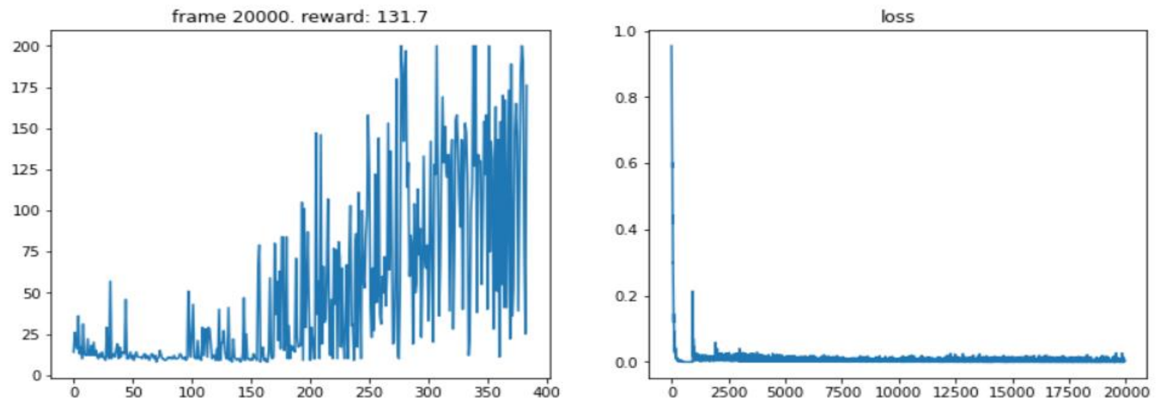
Updating Q value after every Iteration. Where ‘done’ is my variable which tells me whether the present episode is over or not.

```
    next_q_value = next_q_values.max(1)[0]
    # Step 3.2: Calculate and return the TD target using the Bellman equation
    expected_q_value = reward+gamma*next_q_value*(1-done)
return expected_q_value
```

After this we need to compute TD loss and backpropagate losses. Below is how we backpropagate in the neural networks.

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
return loss
```

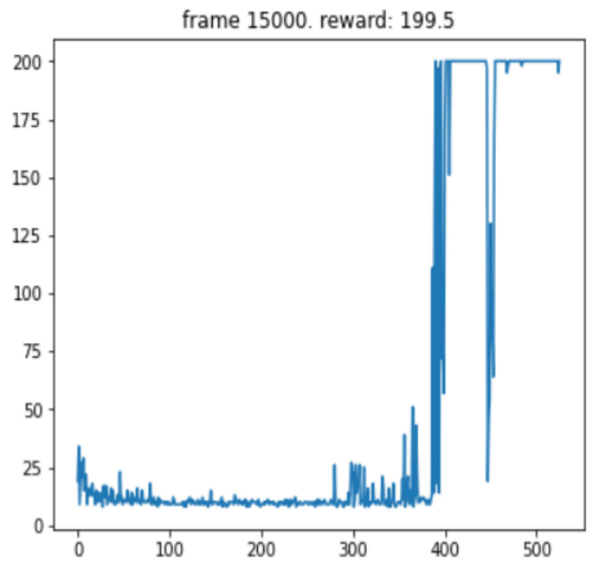
Then we train our Deep Q Networks only if we have enough examples in the replay buffer by invoking the compute td loss function. We also plot the reward and loss every 200 episodes. We are using a learning rate of 1e-4. The optimizer is Adam optimizer. And finally, copy the parameters of the DQN to the Target Q-network after every 100 episodes.

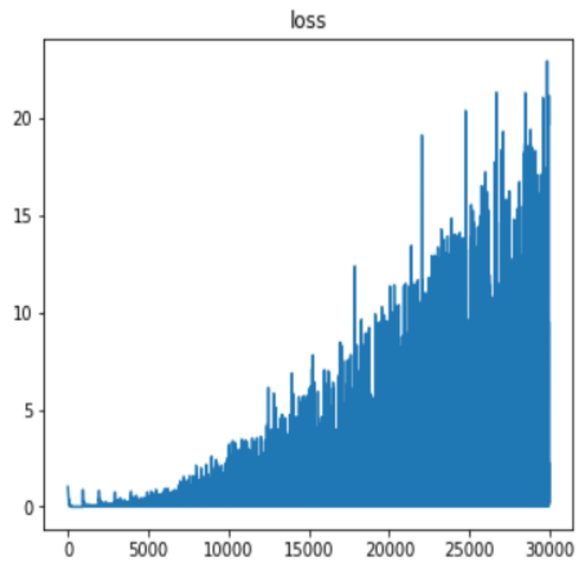


```
gamma=0.49
num_frames=2000
```

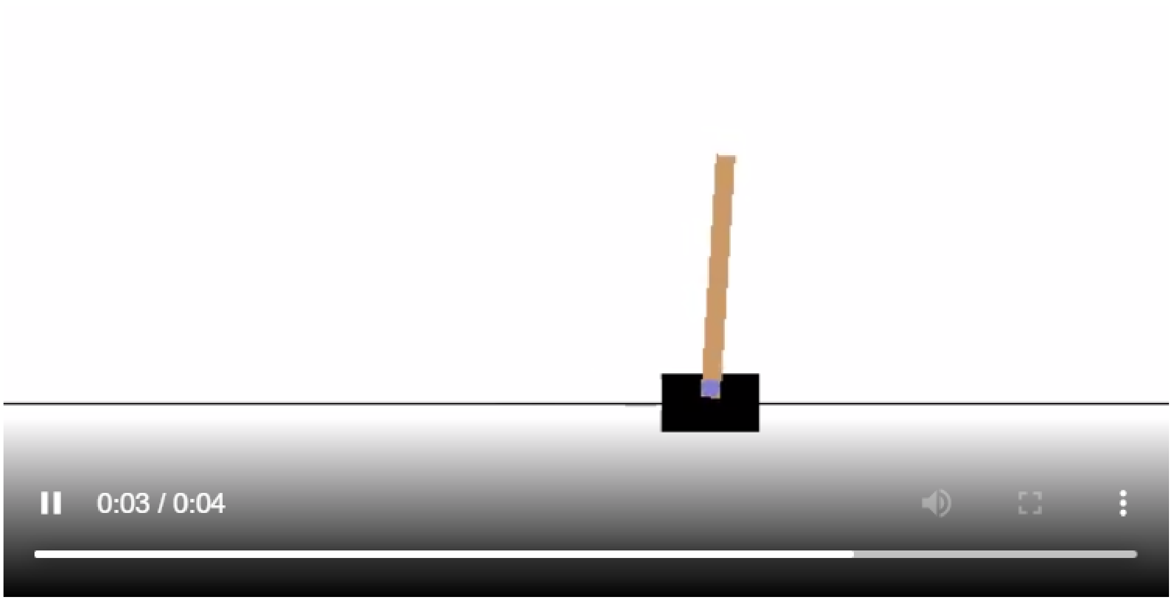
Results:

After careful observation we choose the best hyper parameters (gamma=0.99 and num_frames =1500). We train the network to get a cumulative reward of 199.5 which was desired (cumulative reward>190).





Here we can see the below image where my cartpole is in balanced position for 4 seconds before my episode ends. **Thus, environment is solved**.



---xxx---