CS5542 BIG DATA APPS AND ANALYTICS

In Class Programming – 10

Deep Q-Learning

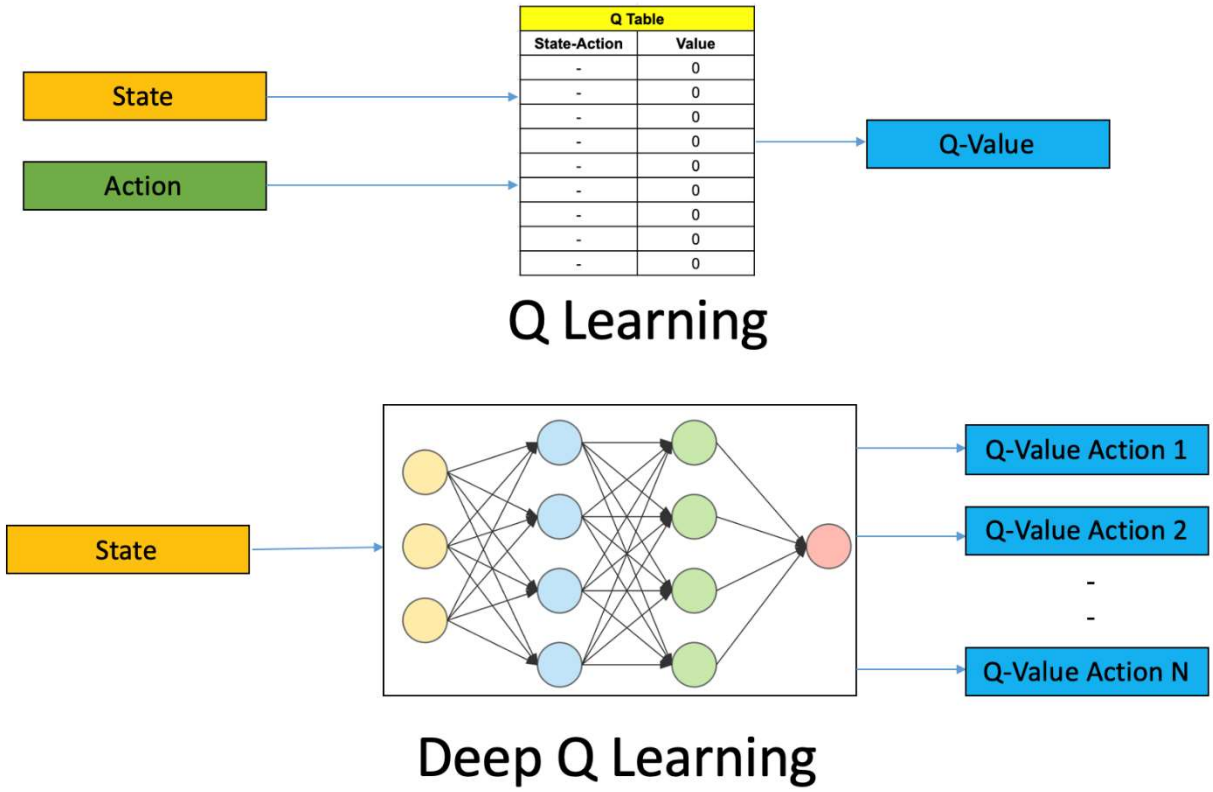
Kalyani Nikure
Kmn6bg@umkc.edu

Table of Contents

Description

Implementing Deep Q-Learning in Python using Keras & OpenAI Gym:

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning & deep Q-learning is illustrated below:



CartPole is one of the simplest environments in the OpenAI gym (a game simulator). The idea of CartPole is that there is a pole standing up on top of a cart. The goal is to balance this pole by moving the cart from side to side to keep the pole balanced upright.

Design a Deep Q learning Network (DQN), using Keras & OpenAI Gym, for cartpole game and visualize your results.

Detailed Steps Explanation

1. About CartPole Game:

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.

More information about cartpole
https://gym.openai.com/envs/CartPole-v1/

2. Installations

Do all the necessary installations like pyvirtualdisplay, piglet for visualizations and gym to get the environment from the OpenAI Gym.

```
!pip install pyvirtualdisplay
!pip install piglet
!pip install gym
```

3. Importing the libraries

```
from pyvirtualdisplay import Display

from IPython import display
import gym
import keras
import numpy as np
import random
import matplotlib.pyplot as plt
from random import shuffle

from gym import wrappers
from keras.models import Sequential, load_model
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam

from collections import deque
```

4. Creating the environment

The Gym library by OpenAI provides virtual environment that can be used to compare performances of different reinforcement learning techniques. I will be using the CartPole Environment form the OpenAI Gym Library. Let's start by creating the environment and getting some useful information about the environment.

```
env = gym.make('CartPole-v1')
```

```
env.reset()
for _ in range(1000):
  action = env.action_space.sample()
  observation, reward, done, info = env.step(action)
  env.render()
env.close()
```

5. Detailed information of the environment

There are 2 possible actions that can be performed at each time step: move the cart to the left (0) or to the right (1). There are 4 states that can be observed at each time step:

- the position of the cart
- its velocity
- the angle of the pole
- the velocity of the pole at the tip

next_state, reward, done, info = env.step(action)

As we discussed above, action can be either 0 or 1. If we pass those numbers, env, which represents the game environment, will emit the results. done is a boolean value telling whether the game ended or not. The old state information paired with action and next_state and reward is the information we need for training the agent.

```
# Environment Information
env.reset()
print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
print("High Observation Space {}".format(env.observation_space.high))
print("Low Observation Space {}".format(env.observation_space.low))
```

```
Action Space Discrete(2)
State Space Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.40
28235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
High Observation Space [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
Low Observation Space [-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

6. Hyper Parameters

There are some parameters that must be passed to a reinforcement learning agent.

- episodes - several games we want the agent to play.
- gamma - aka decay or discount rate, to calculate the future discounted reward.
- epsilon - aka exploration rate, this is the rate in which an agent randomly decides its action rather than prediction.
- epsilon_decay - we want to decrease the number of explorations as it gets good at playing games.
- epsilon_min - we want the agent to explore at least this amount.

- learning_rate - Determines how much neural net learns in each iteration.

```
ACTIONS_DIM = 2 ## Output Dimension=No. of possible actions (2)
OBSERVATIONS_DIM = 4 ## Input Dimension= No of Elements in State Tuple (4)
MAX_ITERATIONS = 200 ## Max Time Steps Per Game (Limited to 200 by Environment)
LEARNING_RATE = 0.01
NUM_EPOCHS = 50

GAMMA = 0.99 ## Future Reward Discount Factor
REPLAY_MEMORY_SIZE = 10000 ## Replay Memory Size
NUM_EPISODES = 200 ## Games Played in Training Phase
MINIBATCH_SIZE = 32 ## Number of Samples chosen randomly from Replay Memory

RANDOM_ACTION_DECAY = 0.99 ## The factor by which Random Action Probability Decreases
INITIAL_RANDOM_ACTION = 1 ## Initial Random Action Factor
Samples=[] ## A list to store Individual Game Scores
Means=[] ##A list to store Mean Score over Last 20 Games
```

7. Class and Functions

Here I have defined a class and few functions.

ReplayBuffer - A class is defined for replay memory which has a data structure of dqueue type with maximum length of REPLAY_MEMORY. SIZE (initialised above).

get_q - A function to Predict Q-Values from the Model. train - A function to Train the Model

predict - A function to predict Q values form model get_model - A function to build the Deep Q-Network update_action - A function to update the model

Model Details - We will have one input layer that receives 4 information and 3 hidden layers. But we are going to have 2 nodes in the output layer since there are two buttons (0 and 1) for the game. Memorize One of the challenges for DQN is that neural network used in the algorithm tends to forget the previous experiences as it overwrites them with new experiences. So, we need a list of previous experiences and observations to re-train the model with the previous experiences.

Replay A method that trains the neural net with experiences in the memory is called replay First, we sample some experiences from the memory and call them minibatch.

To make the agent perform well in long-term, we need to consider not only the immediate rewards but also the future rewards we are going to get. To do this, we are going to have a ‘discount rate’ or ‘gamma’. This way the agent will learn to maximize the discounted future reward based on the given state.

8. Main Program Logic

In this learning, we have 200 game limits as NUM_EPISODES to train the model and for each game we have 500-time steps as MAX_ITERATIONS.

This program holds score of last 50 games. If average of game score for last 50 games is more than 195 and iteration is more than 195, It means learning is done and model is converged.

Starting of each game, we check whether model is converged or not and then proceed. If not converged, following steps are taken.

- Reduce the Random Action Probability by Decay Factor
- Reset the Environment after Each Game

Now we will iterate over time steps for each game.

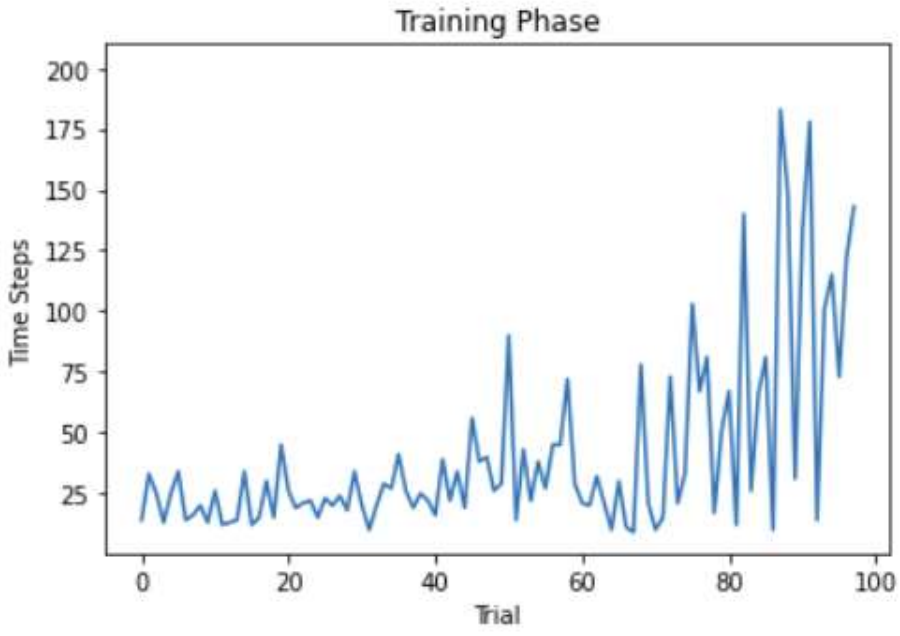
- Generate Random Action Probability
- Store Current State
- Generate random action and if it is less than Random Action Probability, we take it as action else we query the model and Get Q-Values for possible actions and select the best action using Q-Values received and observe next state.
- Check if game is over
    - Add Final Score of the Game to the Score List.
    - If Number of Games>50, Calculate Mean Over 50 Games to Check Convergence
    - Print End-Of-Game Information.
    - If, game is over at last time step, give +5 Reward for Completing the Game Successfully else Give -5 Reward for Taking Wrong Action Leading to Failure.

At the end of each time step, Observation is added in Replay Memory.

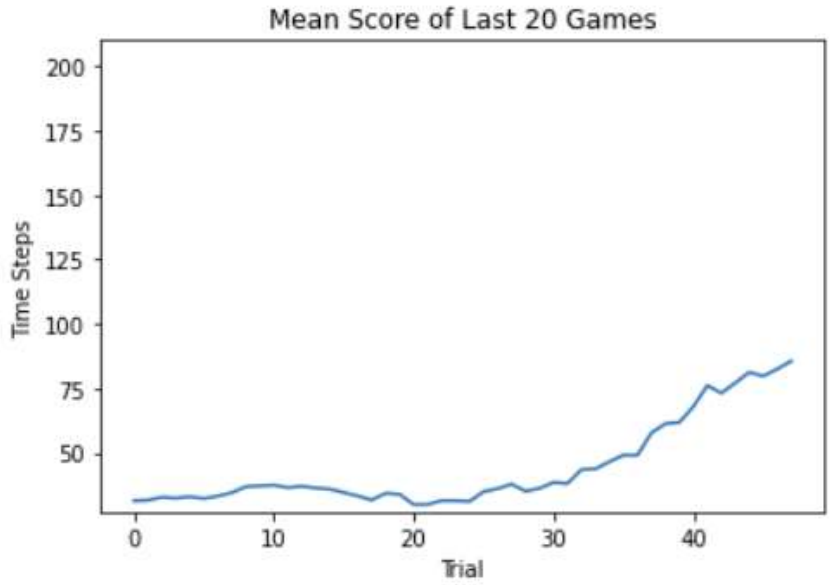
Finally Update the Deep Q-Network Model.

9. Plotting the Training models

Plot below shows the time steps taken across the 100 trained episodes -



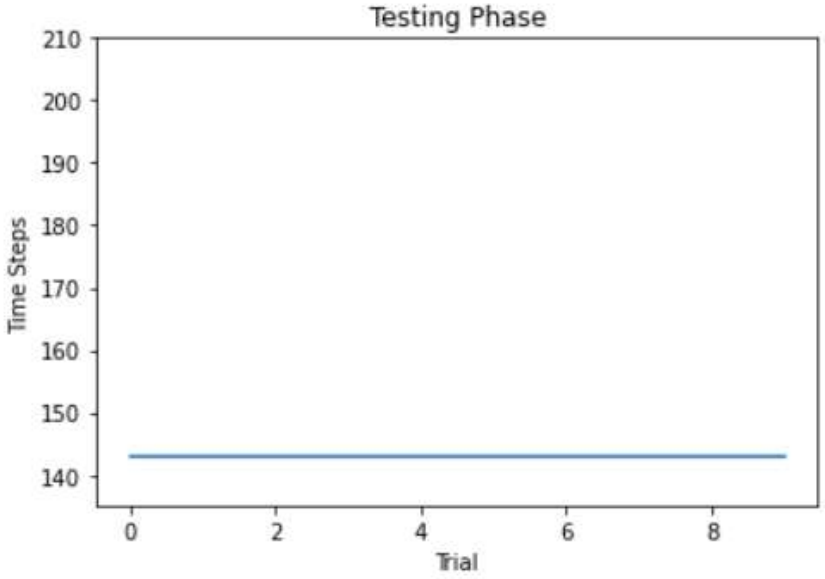
The plot below shows the mean score of last 20 CartPole games -



10. Testing the model

The resulting testing score is 143. The below plot shows the Testing phase across 10 games -

```
143.0
```



Final Observations:

- The Cartpole game becomes more stable for later episodes, so it keeps training with increase in the number of training episodes. So, there is a high possibility that my model would work even better with a greater number of episodes.
- There are multiple number of trainable hyperparameter which can also be leveraged to get a better result.
- We received a very good score of 143 on Testing Phase of the CartPole Game.

Video Link

- https://youtu.be/E774okLSnBI

Conclusion

1. Lessons Learnt

- I learnt how to train a Deep Q-Learning reinforcement model
- I understood other existing experiments present at OpenAI Gym.

2. Challenges Faced

- Running the Cartpole game on the Google Colab was challenging. So, I ran the visualization on Jupyter Notebook after doing installations related to visualization and able to visualize the CartPole Game.
- Training for 100 episodes took a lot of time. So, the considerable amount of time was gone in training the model.