Study and comparison of Q-Learning and Deep Q-Learning on CartPole Game

Chetanraj Kadam
Department of Computer Science
Syracuse University

Abstract- In this semester project I implemented two AI agents to play CartPole game using Q-Learning and Deep Q-Learning. I then carry out comparative study and analysis of these two methods and draw conclusion based on that.

I. INTRODUCTION

To demonstrate both Q-Learning and Deep Q-Learning I need to choose game task where both approaches should work and is reasonable. For Example, if we decide to run Q learning on some complex game where action depends on multiple frames then that will result in large state action space for Q learning. So I choose relatively simple CartPole game but it effectively demonstrates these two approaches.

CartPole is a game in which a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track.[1] The actions are +1 or -1 to control the system to move it left or right. The goal of game is to keep pendulum upright and prevent from falling over. The reward of +1 is given for every timestep pole is upright. The episode fails when cart moves more than 2.4 units from center of pole goes more than 15 degrees from vertical. In this task AI agent needs to interact with environment in terms of various states and based on that takes action without any prior knowledge of environment.

We can describe this task in PEAS form as follows:

- Performance Measure: Pole has to be balanced for 195 time steps for over 100 consecutive trials. Here maximizing reward will be maximizing time for pole to be upright.
- Environment: It is continuous and observable environment with cart moving along track.
- Actuators: Move to left or right by performing +1 or -1 action
- Sensors: Position of cart, velocity of cart, angle of pole and angular velocity.

One of the challenges in this kind of task is continuous environment and has to discretize to run Q-learning. For DQN it automatically takes care of discretization.

My objective is to implement Q-learning agent and DQN agent on this task and compare and analyze their performance. Also observe impact of various structural and parameter changes in model on performance.

II. METHODS

To implement this task I used OpenAI Gym game simulator environment. It is a toolkit for developing and comparing reinforcement learning algorithms.[2] It provides simulated environment for CartPole game. I just need to install the library and call appropriate API. To create at start you need call make environment function and to take action we need to call its step function with required action which simply returns observation, reward and done in this case. A sample image of running CartPole environment generated by the OpenAI Gym environment is shown in figure [1].

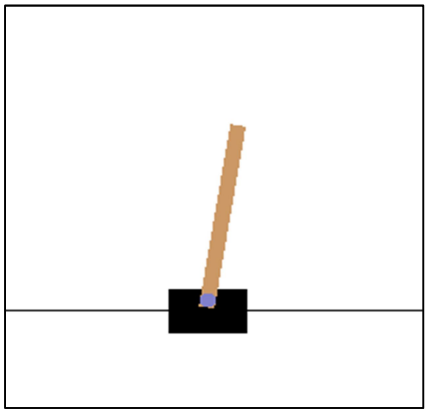

Figure .1 CartPole game in OpenAI Gym

I implemented following two algorithms:

- **Q-Learning:**

Q-learning is value function based approach in which value function tells the maximum expected future reward for each state. In Q-learning we have Q-table in which columns are actions and rows are states. Value stored in table gives us maximum expected reward for that particular state and action. In this we have to find Q-function which takes these two inputs and returns expected future reward. So this function be like-

$$Q * (s, a) = Es\ 0[r + \gamma \max a\ 0Q * (s\ 0\ , a0\ )|s, a]\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

After taking action based on this we have to update Q table with Bellman equation:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}} \quad [3] \ldots\ldots (2)$$

So in this particular I used input states as position(x), velocity(x_dot), angle(theta), angular velocity(theta_dot) same as used in gym environment. Two actions are 0 and 1 to push cart to left and right. So our task has state-space is of four dimensions as inputs and 2 dimension as output. As this will be very large I need to convert these continuous values to discrete space values. So I followed process of discretize this input space first.

Following is pseudocode of implemented approach:

```
initialize scores[ ]
for episode in 1:N:
        reset open gym environment
        discretize current_state s

        for time in 1:T:
                select action a – ( with explore_rate probability of random action or agrmmax(Q(s,a)))

                carry out action a
                observe new_state s' and reward
                discretize new_state s'
                update Q_table with s', a, action – (with learning_rate and discount rate using (2))

                current_state s = new_state s'
                r = r + reward
                if done:
                        break

        append r to rewards[]
        find mean of rewards[]

        if mean > 195:
                game solved
                break

until finish N episodes
```

The challenge with this approach was first to discretize state space and then find appropriate values for parameters like learning rate, exploration rate and discount which give best performance. I set discount rate to 1 as here goal is to keep pole upright as long possible and not to penalize agent when not reaching goal. I done some experiments with learning rate and best worked for agent when its value is relatively small. For exploration rate also set upon trying some different values and observing performance.

- **Deep Q-Learning:**

Deep Q-Learning approach combines reinforcement learning and neural networks. The Deep Q-Network (DQN) proven significant results for AI agent in playing Atari games without any prior knowledge.[4]
So I decided to use same kind of approach in playing CartPole game and check improvement in performance.

So basically in this approach neural network is used to take state as input and give output actions with corresponding Q values. So basically it works same as Q-table but most import advantage of this it works in cases where state-action space is large in which Q-learning is not possible. In addition we need not discretize state values explicitly.

For this task I used deep fully connect neural network with two hidden layers and one output layer. I did not used convolutional which was used in original architecture of DQN because in our task input state is simply represented by 4 values - position(x), velocity(x_dot), angle(theta), angular velocity(theta_dot) not by frames. Neural network architecture I used is as follows:

1. Input layer: 4 input dimension
2. Hidden layer: Fully connected layer with 24 neurons and Relu activation function
3. Hidden layer: Fully connected layer with 48 neurons and Relu activation function
4. Output: layer: 2 output nodes corresponding actions

I used Keras library to build this network. It provides 'Dense' function to add fully connected layers with required parameters. For compiling this model I used 'adam' optimizer and 'mse' loss function. Implemented neural network with Keras is uses Backpropagation algorithm.

To train neural network we need to define loss function and target value according to requirement of the task. I used following equation to find target Q:

$$y = \text{reward} + \gamma \max_a Q(s', a) \qquad \text{……..(3)}$$

So in this process I first calculate maximum target Q value for new state and after discounting add it to current reward. Keras has fit function to train model and I trained it for 1 epoch for each episode.

I am also using Replay strategy mentioned in paper [4]. It is process of retraining of neural network using mini batches selected randomly from saved memory. In remember step, current experience is saved in the form of state, action, reward, and next state and called it as memory. This is important for training DQN because it tends to forget previous experiences and overwrites with new experiences. So to avoid it we need Replay strategy.

Following is pseudocode of implemented approach:

```
build and initialize network model
initialize replay memory
initialize scores[ ]

for episode in 1:N:
        reset open gym environment

        for time in 1:T:
                select action a – (explore_rate probability of random action or prediction from network)
                carry out action a
                observe new_state s' and reward
                store experience <s, a, r, s' > in replay memory
                current_state s = new_state s'
                r = r + reward

                randomly use random mini batch of  <s, a, r, s' > from replay memory
                calculate output y for each  experience using (3) and train network
```

```
                    if done:
                            break
            append r to rewards[]
            find mean of rewards[]

            if mean > 195:
                    game solved
                    break

    until finish N episodes
```

The main challenge for this approach is to design structure of neural network and loss function to train it. I experiment with number of layers and activation function and selected best performing architecture.

The performance metrics I used are reward per episode and mean score upon 100 episodes. I also measured performance considering episodes agent takes to solve the game (i.e to achieve average 195 time steps over 100 consecutive episodes). To test implemented model and set hyperparameters appropriately I displayed rewards I am getting at end of each episode. Based on results I tried different hyperparameter values, model architectures and choose best performing ones. To experiment with agent I ran it various numbers of episodes and visualized results based on plotting graphs for episode vs. reward.

III. RESULTS

I used OpenAI Gym interface to run AI models using GPU (NVidia GTX 1080Ti) for about 1000 episodes each. For Q-Learning model it take few minutes to run and complete solution but with DQN model 1000 episodes take around 2-3 hour time even with GPU.

A. Hyperparameter Tuning

Parameters such as exploration rate, learning rate and discount are common for both algorithms. I used different values for these parameters initially and depending upon results I kept best ones.
I used decaying approach for exploration rate where it starts with high value initially to explore more and reduced over time. So initially it is set epsilon = 1.0.
For Q-Learning I used learning rate alpha = 1.0 initially and gradually reducing it. For DQN I tried using similar learning rate but it reduces performance so I kept low learning rate of 0.001.
However for both algorithms I used discount with its maximum value. That is gamma = 1 for Q and gamma = 0.95 for DQN. This is because agent here should look for long term reward as goal is to keep pole upright as long as possible.

B. Overall Performance: Q vs. DQN

I ran best models obtained after experiment and hyperparameter tuning to compare performance of Q-Learning and Deep Q-Learning.

Q-learning:

Q-learning solved game after running 249 episodes. Game solves when pole is upright for 195 time steps for over 100 consecutive episodes. For this in algorithm I took mean of most recent 100 episodes after completing each iteration of single episode and when mean is above 195 game stops and output result as solved.

I ran it multiple times to observe on average how much episodes this algorithm takes to solve. So mostly got result between 220 to 280 episodes but some in cases 1 out of 10 trials it did not solved even in 1000 episodes. So it is mostly stable approach which can guarantee to solve problem in considerable number of episodes.

Following is performance Episode vs. Rewards graph of Q-Learning agent:

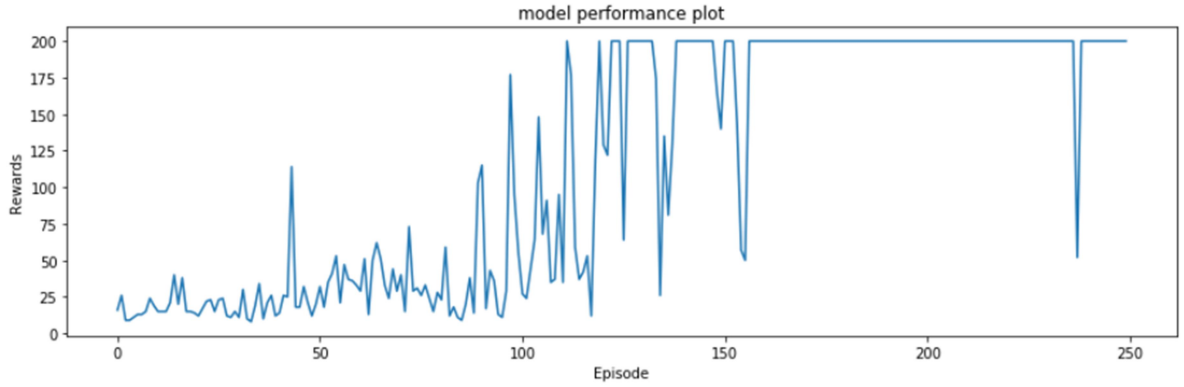

Figure 2. Q-Learning agent (1000 episodes but solved in 250)

From figure 2, we can observe that Q-learning agent learns slowly and rewards are increasing only by small amount. There are some episodes where reward goes very high and dropped immediately. But once it goes after 150 episodes it is going to be stable and gives reward in range of 190-200.

DQN-learning:

It was expected that with use of neural network performance of agent will increase. But in the case of CartPole problem it is observed from results that DQN agent is not able to solve CartPole problem in 1000 episodes. The main reason is reward gets dropped suddenly in between even when it goes above 200 episodes. The reward reaches maximum of 199 but it gets dropped somehow in between and as we are considering mean of latest 100 episodes it is not going above 195.

Following is performance Episode vs. Rewards graph of DQN-Learning agent:

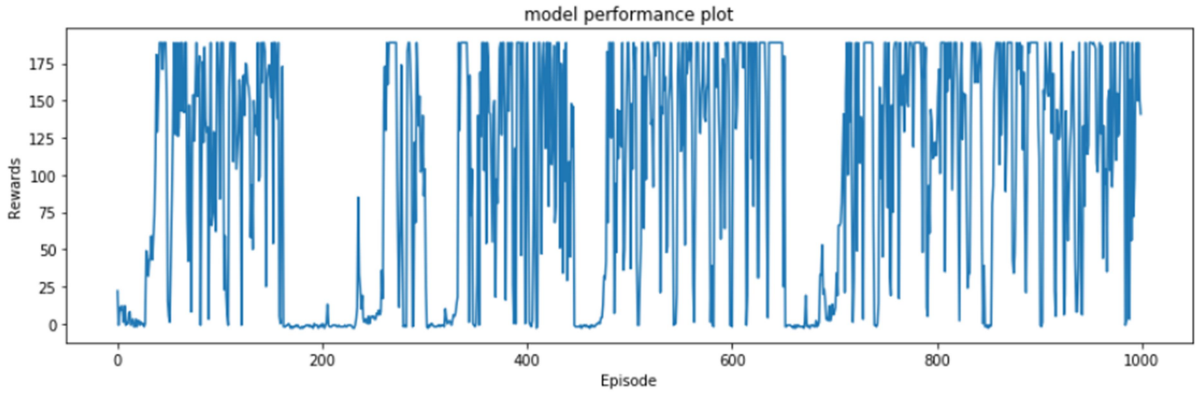

Figure 3. DQN -Learning agent (1000 Episodes)

To compare performance of both models I ran DQN just for 250 episode so we can able to observe comparison in performance. Following is performance graph of DQN model:

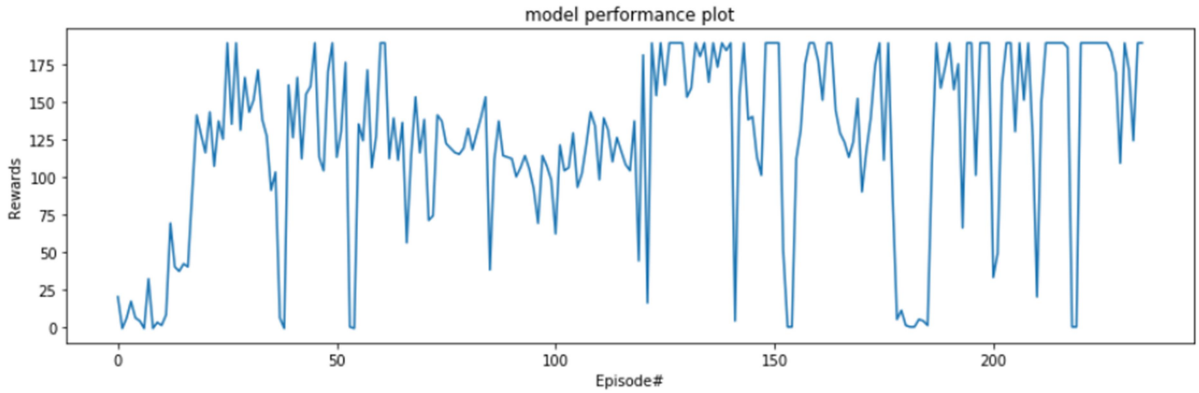

Figure 4. DQN -Learning agent (250 Episodes)

From figure 4, we observe the fact DQN actually learning very fast as compare to Q. First few rewards are small but it is suddenly increasing performance and getting high rewards. But as contrast to Q it is never stabling at high reward and as mentioned before that is main reason it is not able to solve problem in 1000 episodes.

Following are quantitative results got after running both for 1000 episodes:

	Q	**DQN**
Running time (1000 Episodes)	10 min	3 hours
Max Reward	199	199
Min Reward	7	9
Episodes taken to reach reward over 100	89	37
Mean of first 100 episodes	29.8	88.21
Mean of 100 - 200 episodes	165.56	125.37

Table1. Consolidated results of Q vs. DQN

Generally in all experiments running DQN model takes so much time than Q model. This is because of the fact that we are training neural network to give us Q value whereas in Q-model we are just storing those in table. In terms of maximum and minimum rewards both are performing equally.

As observed before DQN learns quickly so it took only 37 episodes for reward to reach 100 mark whereas comparatively slow Q-model took 89 episodes. On similar terms average of first 100 episodes for DQN is also much greater than Q. But real problem begins when going further as DQN does not show much improvement in reward.

C. DQN Variations:

To make DQN model to solve this task I tried to change architecture of neural networks. Initially I was using 24 Hidden – 24 Hidden -2 O/p layer model. But its performance was so much worse than what was expected. Mean score of first 1000 episodes was only 24.39. So I tried to increase neurons in second hidden layer and got 24-48-2 model which I described above. It gave me performance which is mentioned in above section. I experiment with increasing one more layer and used 24-48-48-2 model. Performance is so much worse than initial model. So here it is observed that increment in neurons and layers is not directly proportional to increase in performance.

I used different activation function for output layer such as sigmoid, linear and softmax. But out of these linear activation was given best performance.

IV. CONCLUSION

In this project I am able to implement both Q-learning and Deep Q-Learning AI models to play CartPole game. From results we can conclude that AI agents are capable of learning and solving task with the Reinforcement Learning. Both Q-Learning and DQN models performed well to increase rewards after certain number of episodes.

Discretizing state space for Q-learning and designing of DQN algorithm are most difficult parts of the project. In addition experimenting and training of DQN was also challenging and time consuming. One important observation I draw from this project is that performance of algorithms are highly depend on structure and environment of the tasks. In this task, surprisingly Deep Q-Learning algorithm did not performed well but it is demonstrated in several researches that DQN performed very well on Atari games than any other earlier methods. But it does not conclude that DQN is not worked at all for this CartPole task, it demonstrated to learn quickly and give high rewards faster than Q-Learning.

Because of constraint in time and computational resources, I think DQN model not trained enough to solve the task. I am also sure that if we spend considerable amount of time to on training DQN thousands of episodes with proper hyperparameter tuning and then if we run CartPole problem, it may show effective results. That is the most important future work to be done for this project. I am also interested in studying and comparing performance other reinforcement learning algorithms such as Policy Gradients and Actor-Critic algorithms.

REFERENCES

[1] Cartpole-vo: https://gym.openai.com/envs/CartPole-v0/

[2] Open AI Gym environment: https://gym.openai.com/

[3] https://en.wikipedia.org/wiki/Q-learning

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[5] R. Sutton and A. Barto, Reinforcement learning: An introduction, vol. 1. Cambridge Univ Press, 1998.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," CoRR, vol. abs/1312.5602, 2013