Ministère de l'Enseignement Supérieur Et recherche scientifique

Université de Manouba

Solving Cartpole with QN,DQN &DDQN

2021/2022

Done by :

Benali Khaoula

Lasmar Raya

Teacher :

Mrs Krichen Sabrine

Contents :

General Inroduction

In this project,our goal is to train an agent to balance a pole on the top of a moving cart using an OpenAI Gym environment.

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.

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson and it is offered by OpenAI gym with the ID : CartPole-v0.

I. Environment :

❑ Details:

- Name: CartPole-v0
- Category: Classic Control

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Velocity At Tip	-Inf	Inf

❑ Actions:

The agent is going to choose one between two discrete actions :

Num	Action
0	Push cart to the left
1	Push cart to the right

Note: How the velocity is reduced or increased is not fixed as it depends on the angle the pole is pointing. This is because the center of gravity of the pole increases the amount of energy needed to move the cart underneath it.

❑ Reward:

The reward is 1 for every step taken, including the termination step.

❑ Starting State:

All observations are assigned a uniform random value between ±0.05.

❑ Episode Termination:

1. Pole Angle is more than ±12°.
2. Cart Position is more than ±2.4 (center of the cart reaches the edge of the display).
3. Episode length is greater than 200 (500 for v1).

❑ Solved Requirements:

Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.

II. Implementation :

1. Frameworks :

- **Gym** : Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.
- **Keras** : Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.
- **Keras-RL** : keras-rl is a Reinforcement Learning library based on Keras.

2. Environment Implementation:

Rather than coding the environment from scratch, we used **OpenAI Gym**.Gym makes no assumptions about the structure of the agent (what pushes the cart left or right in this cartpole example), and is compatible with any numerical computation library, such as numpy.

We started first by loading our environment using **gym.make** function that takes the environment ID as input and returns an instance of the environment:

1. Loading the OpenAI gym environment :

```
env = gym.make('CartPole-v0')
#building our environment wich is a pole trying to balance on the top of a moving cart.
```

The next step was the instanciation of the actions, we have two discrete actions which are

0 : Pushing cart to the left

1 : Pushing cart to the right

```
action=env.action_space.sample()
#action is a sample of our environment actions which are 0 or 1 : right or left
```

The environment takes 500 step (500 cycle), always taking a random action and printing the results.

```
for t in range(500):
    env.render()
    #render is a function that displays the environment
    action=env.action_space.sample()
    #action is a sample of our environment actions which are 0 or 1 : right or left
    next_state,reward,done,info=env.step(action)
    #step is going to take one of the actions either left or right
    #done is a boolean variable that tells wether the game ended or not
    #next_state gives the possible state if we take that particular action
    print(t,next_state,reward,done,info,action)
    if done:
        break
```

The previous work was done and redone multiple times during 10 episodes of training. The pole balances but not in an efficient way and it never reaches a good score.

That’s why we needed to use deep learning to improve the decisions taken by the agent.

III. Hyperparameters :

- Episodes (Number of games the agent is going to play) = 1000
- Gamma (The future discount reward) = 0.95
- Epsilon (The exploration rate : proba of choosing an action randomly) = 1
- Epsilon_dacay (to decrease the exploration rate as the agent gets good) = 0.999
- Learning rate (How much the neural network learns at each iteration) = 0.00025
- Batch size (Memory used by the Neural Network to learn) = 64
- Optimizer : we chose RMSprop with :
    - Learning rate = 0.00025
    - Discounting factor for the history/coming gradient (rho) = 0.95
    - Epsilon (The probability of taking a random action) =0.01
- Metrics : we chose accuracy to evaluate our model.

IV. The Learning Algorithms :

1. Deep Reinforcement Learning :

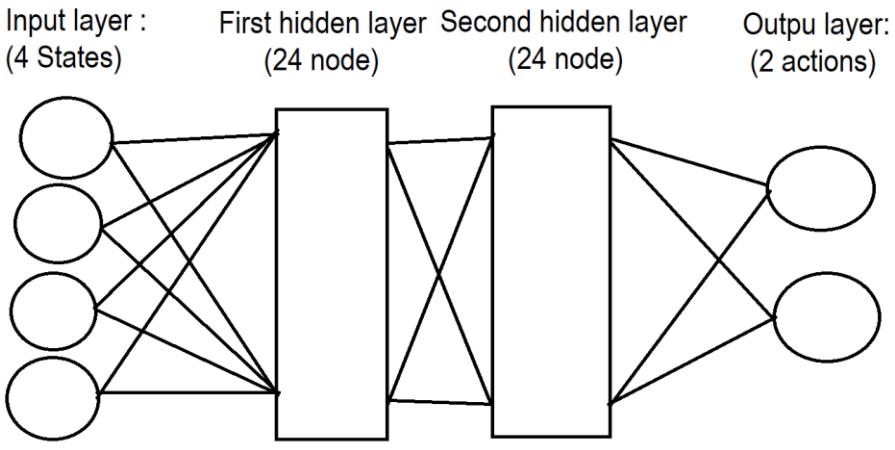
❑ Hyperparameters:

For this part our model is going to be very simple as the CartPole balancing problem is not that complicated and it doesn't require a complex neural network architecture.

We chose as hyper-parameters :

- Two hidden layers each one is composed of 24 node.
- Activation functions : "relu" for hidden layers and "linear" for output layer.

❑ The model architecture :



Our model is composed of :

- Input layer : Flatten layer composed of 4 nodes as the total number of states is 4.
- First hidden layer : Fully connected layer composed of 24 node.
- Second hidden layer : Fully connected layer composed of 24 node.
- Output layer : Fully connected layer composed of 2 nodes as the totel number of actions to choose from is 2.

❑ Results:

```
scores = dqn.test(env, nb_episodes=100, visualize=False) #let's test the performance of our model
print(np.mean(scores.history['episode_reward']))
```

```
Testing for 100 episodes ...
Episode 1: reward: 200.000, steps: 200
Episode 2: reward: 200.000, steps: 200
Episode 3: reward: 200.000, steps: 200
Episode 4: reward: 200.000, steps: 200
Episode 5: reward: 200.000, steps: 200
Episode 6: reward: 200.000, steps: 200
Episode 7: reward: 200.000, steps: 200
Episode 8: reward: 200.000, steps: 200
Episode 9: reward: 200.000, steps: 200
Episode 10: reward: 200.000, steps: 200
Episode 11: reward: 200.000, steps: 200
Episode 12: reward: 200.000, steps: 200
Episode 13: reward: 197.000, steps: 197
Episode 14: reward: 200.000, steps: 200
Episode 15: reward: 200.000, steps: 200
Episode 16: reward: 200.000, steps: 200
Episode 17: reward: 200.000, steps: 200
Episode 18: reward: 200.000, steps: 200
```

As seen,this model leads to great results the agent is receiving a value of 200 as a reward each time.

2. Double Deep Reinforcement Learning :

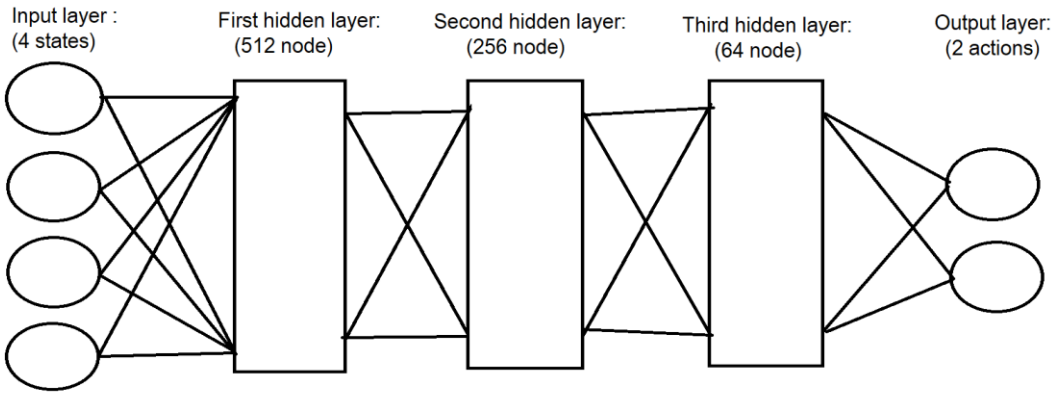
❑ Hyperparameters:

For this part we are going to work with a more complicated model in termof number of nodes in each layer.

We chose as hyper-parameters :

- Three hidden layers.
- Activation functions : “relu” for input and hidden layers and “linear” for output layer.

❑ The model architecture



Our model is composed of:

- Input layer : 4 nodes (4 states).
- First hidden layer : fully connected with 512 node.
- Second hidden layer : fully connected with 256 node.
- Third hidden layer : fully connected with 64 node.
- Output layer : 2 nodes (2 actions).

❑ Results :

```
episode: 988/1000, score: 249, e: 0.01, average: 240.3
episode: 989/1000, score: 339, e: 0.01, average: 240.4
episode: 990/1000, score: 281, e: 0.01, average: 240.5
episode: 991/1000, score: 226, e: 0.01, average: 240.5
episode: 992/1000, score: 205, e: 0.01, average: 240.4
episode: 993/1000, score: 249, e: 0.01, average: 240.4
episode: 994/1000, score: 236, e: 0.01, average: 240.4
episode: 995/1000, score: 237, e: 0.01, average: 240.4
episode: 996/1000, score: 261, e: 0.01, average: 240.5
episode: 997/1000, score: 114, e: 0.01, average: 240.3
episode: 998/1000, score: 286, e: 0.01, average: 240.4
episode: 999/1000, score: 243, e: 0.01, average: 240.4
```

As seen,this model leads to great results.After 100 cosecutive trials the agent is receiving as a reward an average value of 240 which is higher than 195.

Conclusion

Balancing a pole using Q-Learning is quite a good idea and it leads to good results,but,it’s not as good as using a deep learning model to train the agent to choose the best action each time.

However,using a duel deep learning network leads to the best results even though in this example ,the problem isn’t that complicated.