University of Manouba
National School of Computer Sciences

Report of Deep RL project

Double Deep QNetwork Implementation on the CartPole-v0 OpenAI Gym

Authors :

Khaled EL AYECH
Alaeddine JOHMANI

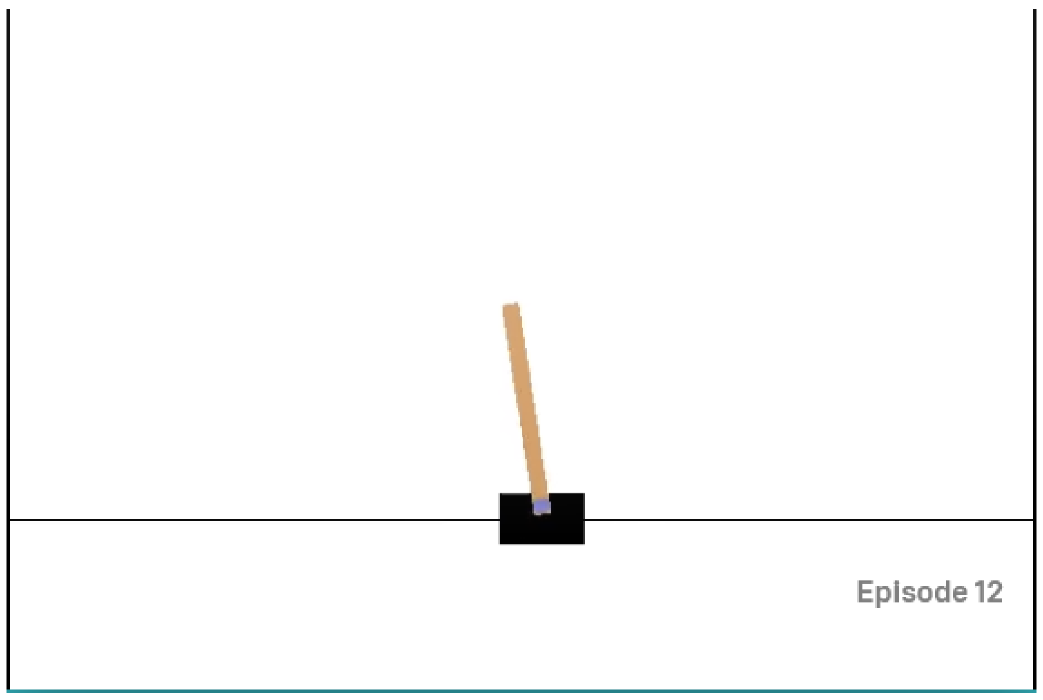
Academic Year : 2021 /2022

Table of content

1. The environment

The environment used is CartPole-v0. This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson.

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart’s velocity.



Cartpole - v0

2.The implementation

For the implementation of our solution, we divided the work into classes :

- **Class Agent :**

Which Interacts with and learns from the environment. In this class, we have 4 methods :

➢ Step

➢ Act :

Returns actions for given state as per current policy.

Params :
state (array_like): current state
eps (float): epsilon, for epsilon-greedy action selection

➢ Learn :

Update value parameters using a given batch of experience tuples.

Params
experiences (Tuple[torch.Variable]): tuple of (s, a, r, s', done) tuples
gamma (float): discount factor

➢ Soft_Update :

Soft update model parameters.

$\theta\_target = \tau*\theta\_local + (1 - \tau)*\theta\_target$

Params
local_model (PyTorch model): weights will be copied from
target_model (PyTorch model): weights will be copied to
tau (float): interpolation parameter

- **Class QNetwork :**
  This class has 2 methods, the constructor and the forward method that builds a network that maps state into action values.

- **Class ReplayBuffer:**
  which represents a Fixed-size buffer to store experience tuples.
  This class has 2 methods :

  - ➢ add : Add a new experience to memory.

  - ➢ sample : Randomly sample a batch of experiences from memory.

3. The learning algorithms

As a learning algorithm, we chose to use the Deep QNetwork algorithm.

A DQN, or Deep Q-Network, is a reinforcement learning algorithm that approximates a state-value function in a Q-Learning framework with a neural network. It is usually used in conjunction with Experience Replay, for storing the episode steps in memory for off-policy learning, where samples are drawn from the replay memory at random.

One of the improvements for the Deep Q-Network is the Double Deep Q-Network. As a matter of fact, DQN tends to overestimate action values which can be harmful to training performance and can lead to suboptimal policies especially in early stages.

The solution involves using two separate Q-value estimators, each of which is used to update the other. Using these independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator.

4. The agent hyperparameters

- ➢ **Replay buffer size :**
  BUFFER_SIZE = int(1e5)

- ➢ **Minibatch size :**
  BATCH_SIZE = 64

- ➢ **Discount factor:**
  GAMMA = 0.99

- ➢ **For soft update of target parameters:**
  TAU = 1e-3

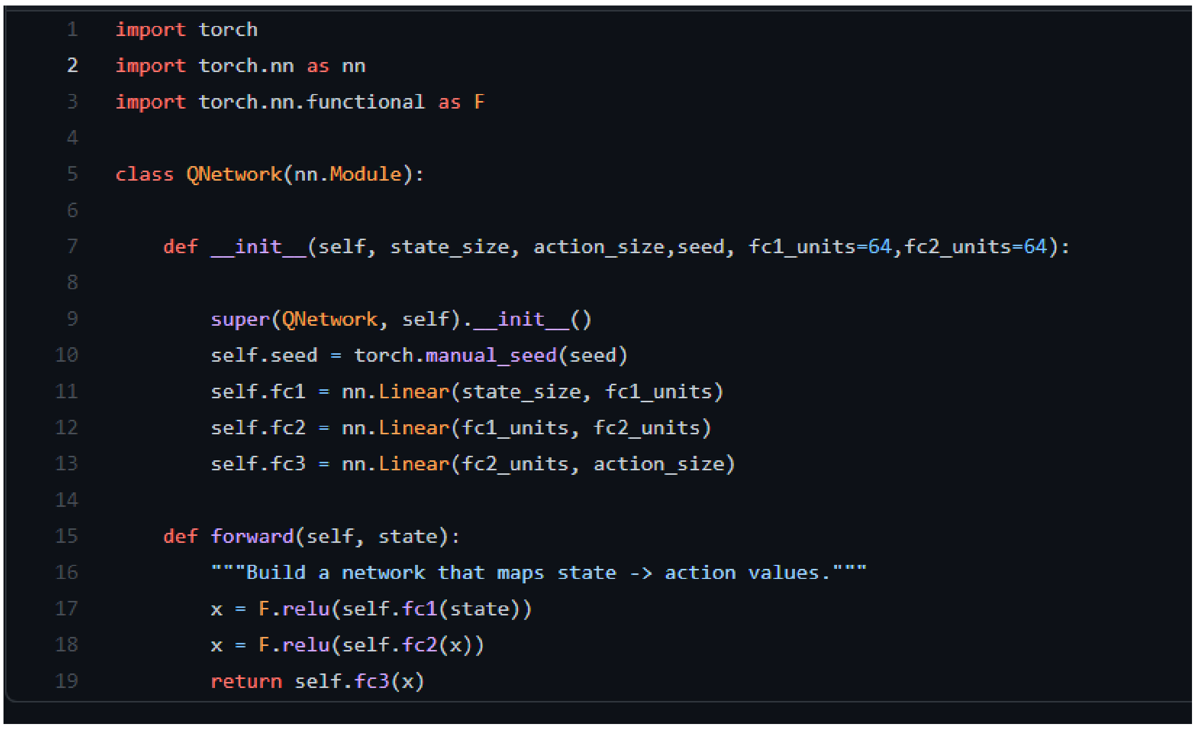
- ➢ **Learning rate :**
  LR = 5e-4

- ➢ **How often to update the network :**
  UPDATE_EVERY = 4

5. The model architecture and hyperparameters

Our model is composed of 3 fully connected layers with Relu as activation function each layer contains 64 hidden units.

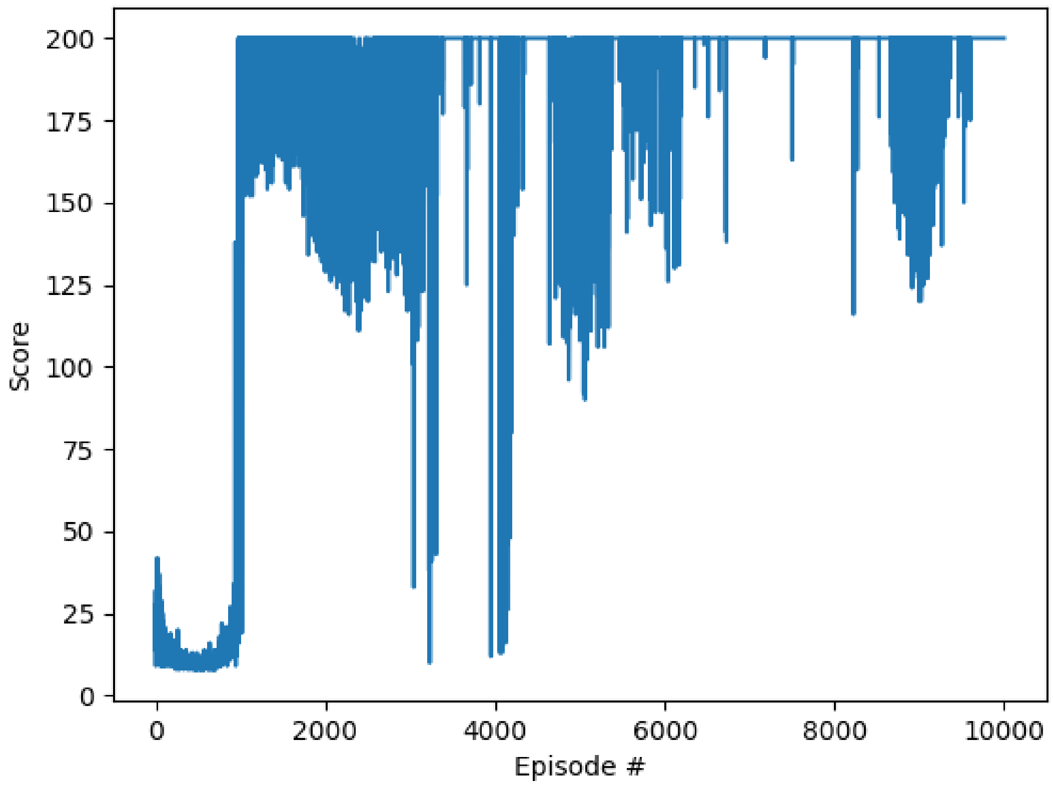
As optimizer we chose to use Adam with a learning rate = 0.001 of and MSE as a loss function.



```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class QNetwork(nn.Module):

    def __init__(self, state_size, action_size,seed, fc1_units=64,fc2_units=64):

        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

6. The detailed results

After training our agent for a number of episodes equal to 10000 episode, we have obtained the following score results plot bar :



Our solution converged in about 1000 episodes as it recorded the best reward score which is equal to 200.