ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Reinforcement Learning
Stock Day-Trading with Q-Learning and DQNs

July 18, 2024

1 Introduction to the problem

As the title of the report describes, we are called to solve a stock market day-trading problem. Each stock evolves as a 2-state Markov chain, alternating between a high (H) and a low (L) state. The exact MDP model has been provided in the previous assignment so there is no need to elaborate further on this matter.
The user starts each round with 1 euro, and at the end of it receives this amount multiplied by the percentage change of the stock the user chose to buy. Additionally, there is a transaction fee c that is applied whenever the user chooses to change from one stock to another. No transaction fee is applied if the user stays on the same stock.
For our analysis we used (as suggested) the result of Phase 1 as the ground truth, in order to compare the optimality of our new algorithms' chosen actions. We first started by examining toy scenarios and then moved on to larger-scale problems.

2 Task 1: Tabular Q-Learning for 'toy' scenarios

2.1 Comparison with Phase 1-Question 1

Problem Setup:
We first implemented the tabular Q-Learning algorithm provided in the interactive notebook delivered with this report. Then we used the same parameters for the environment as we did in Phase 1-Question 1, to examine if we can yield the same results.
Stocks: $N = 2$
Gamma (Discount Factor): $\gamma = 0$
Alpha (Learning Rate): 0.1
Epsilon (Exploration Rate): 0.1
Number of Episodes: 10000
Transaction Cost: 0.12
Horizon T: 100
Transition Probabilities:

- $p^1_{HH} = 0.9$, $p^1_{HL} = 0.1$,$p^1_{LL} = 0.9$, $p^1_{LH} = 0.1$
- $p^2_{HH} = 0.9$, $p^2_{HL} = 0.1$,$p^2_{LL} = 0.9$, $p^2_{LH} = 0.1$

Rewards:

- $r^1_H = 0.1$, $r^1_L = -0.02$
- $r^2_H = 0.05$, $r^2_L = -0.01$

We then compared the results of our algorithm with those of Phase 1 to see if we had the same results.
Optimal Policy chosen by the PI algorithm/Values:

State	Optimal Action	State Value
(1, 'H', 'H')	keep	0.1000
(1, 'H', 'L')	keep	0.1000
(1, 'L', 'H')	keep	-0.0200
(1, 'L', 'L')	keep	-0.0200
(2, 'H', 'H')	keep	0.0500
(2, 'H', 'L')	keep	-0.0100
(2, 'L', 'H')	keep	0.0500
(2, 'L', 'L')	keep	-0.0100

Table 1: Results of PI algorithm

Optimal Policy chosen by the Q-Learning algorithm/Values:

State	Optimal Action	State Value
(1, 'H', 'H')	keep	0.1000
(1, 'H', 'L')	keep	0.1000
(1, 'L', 'H')	keep	-0.0200
(1, 'L', 'L')	keep	-0.0200
(2, 'H', 'H')	keep	0.0500
(2, 'H', 'L')	keep	-0.0100
(2, 'L', 'H')	keep	0.0500
(2, 'L', 'L')	keep	-0.0100

Table 2: Results of Q-Learning algorithm

We observe that for this toy scenario, both algorithms converge to the same optimal policy and value function for each state.

2.2 Comparison with Phase 1-Question 2

Here we used the same parameters for the environment as we did in Phase 1-Question 2, to examine if we can yield the same results.
Stocks: $N = 2$
Gamma (Discount Factor): $\gamma = 0.9$
Alpha (Learning Rate): 0.1
Epsilon (Exploration Rate): Starting from 1, multiplying itself by 0.995 at the end of every episode so that it decays over time
Number of Episodes: 1000
Transaction Cost: 0.03
Horizon T: 500
Transition Probabilities:

- $p^1_{HH} = 0.9$, $p^1_{HL} = 0.1$,$p^1_{LL} = 0.9$, $p^1_{LH} = 0.1$
- $p^2_{HH} = 0.9$, $p^2_{HL} = 0.1$,$p^2_{LL} = 0.9$, $p^2_{LH} = 0.1$

Rewards:

- $r^1_H = 0.1$, $r^1_L = -0.02$
- $r^2_H = 0.05$, $r^2_L = -0.01$

We then compared the results of our algorithm with those of Phase 1 to see if we had the same results. **Optimal Policy chosen by the PI algorithm/Values**:

State	Optimal Action	State Value
(1, 'H', 'H')	keep	0.8988
(1, 'H', 'L')	keep	0.8920
(1, 'L', 'H')	switch	0.7863
(1, 'L', 'L')	keep	0.7720
(2, 'H', 'H')	switch	0.8688
(2, 'H', 'L')	switch	0.8620
(2, 'L', 'H')	keep	0.8163
(2, 'L', 'L')	keep	0.7563

Table 3: Results of PI algorithm

Optimal Policy chosen by the Q-Learning algorithm/Values:

State	Optimal Action	State Value
(1, 'H', 'H')	keep	0.7080
(1, 'H', 'L')	keep	0.6520
(1, 'L', 'H')	switch	0.3990
(1, 'L', 'L')	switch	0.2834
(2, 'H', 'H')	switch	0.6498
(2, 'H', 'L')	switch	0.5415
(2, 'L', 'H')	keep	0.5241
(2, 'L', 'L')	keep	0.3317

Table 4: Results of Q-Learning algorithm

We observe that the Q-Learning algorithm chooses almost always the same policy as the PI algorithm for each state. What is interesting is that there is a significant divergence between the State Values of the 2 algorithms. This could be possibly fine-tuned by picking proper epsilon, alpha.

3 Task 2: Tabular Q-Learning for a large scenario

In this task we tried to solve the larger scenario we tested in Phase 1-Question 3 with tabular Q-Learning. Here we used the same parameters for the environment as we did in Phase 1-Question 3, to examine if we can yield the same results. For convenience, we outputted the transition probabilities and rewards (from the Phase 1 environment) of each stock to a CSV file (stock_params.csv), so that we can initialize our environment for the tabular Q-Learning algorithm.
Stocks: $N = 8$
Gamma (Discount Factor): $\gamma = 0.99$
Alpha (Learning Rate): 0.1
Epsilon (Exploration Rate): 0.1
Number of Episodes: 15000
Transaction Cost: 0.1
Horizon T: 10000
Transition Probabilities $\&$ Rewards: Contained in file stock_params.csv We observed the policy our Q-learning algorithm chose for each state. What we saw is that for the above horizon our Q-Learning algorithm manages to achieve the same policy that the PI algorithm achieved in Phase 1, and that the Value Function for each state converges with very high accuracy (0.001 divergence). Below is a sample of what the output is for the algorithm of Phase 1 and our Q-learning algorithm. For the full results, check the files inside the folder 'Detailed Outputs of Task 2'.
Important: The enumeration of the stock on the PI algorithm ranges from 0 to 7, while on the Q-Learning algorithm it ranges from 1 to 8.
Optimal Policy chosen by the PI algorithm/Values (sample):

State	Optimal Action	State Value
”(0,('H', 'H', 'H', 'H', 'H', 'H', 'H', 'H'))”	7	6.698105180031515
”(0, ('H', 'H', 'H', 'H', 'H', 'H', 'H', 'L'))”	7	6.69377045343178
”(0, ('H', 'H', 'H', 'H', 'H', 'H', 'L', 'H'))”	7	6.698105180031515
”(0, ('H', 'H', 'H', 'H', 'H', 'H', 'L', 'L'))”	7	6.69377045343178
”(0, ('H', 'H', 'H', 'H', 'H', 'L', 'H', 'H'))”	7	6.698105180031515
”(0, ('H', 'H', 'H', 'H', 'H', 'L', 'H', 'L'))”	7	6.69377045343178
”(0, ('H', 'H', 'H', 'H', 'H', 'L', 'L', 'H'))”	7	6.698105180031515
”(0, ('H', 'H', 'H', 'H', 'H', 'L', 'L', 'L'))”	7	6.69377045343178
”(0, ('H', 'H', 'H', 'H', 'L', 'H', 'H', 'H'))”	7	6.698105180031515

Table 5: Results of PI algorithm

Optimal Policy chosen by the Q-Learning algorithm/Values (sample):

State	Optimal Action	State Value
(1, 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H')	8	6.6972
(1, 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'L')	8	6.6936
(1, 'H', 'H', 'H', 'H', 'H', 'H', 'L', 'H')	8	6.6986
(1, 'H', 'H', 'H', 'H', 'H', 'H', 'L', 'L')	8	6.6942
(1, 'H', 'H', 'H', 'H', 'H', 'L', 'H', 'H')	8	6.6986
(1, 'H', 'H', 'H', 'H', 'H', 'L', 'H', 'L')	8	6.6941
(1, 'H', 'H', 'H', 'H', 'H', 'L', 'L', 'H')	8	6.6987
(1, 'H', 'H', 'H', 'H', 'H', 'L', 'L', 'L')	8	6.6934
(1, 'H', 'H', 'H', 'H', 'L', 'H', 'H', 'H')	8	6.6979

Table 6: Results of the Q-Learning algorithm

What is also notable is that the Q-Learning algorithm ran this scenario in almost half the time (30m) in comparison to the PI algorithm.

4 Task 3: Implementing neuro-fitted Q-Learning for the above scenario (DQN)

In this task we tried to solve the above scenario with Deep Q Learning. We once again used the CSV file to setup our environment.

4.1 Architecture of our Neural Network

1.Input Layer:
The input layer receives a tensor representing the continuous yield values of the stocks. The size of the input layer is equal to the number of stocks (‘input_dim‘), which is derived from the number of stocks available.
2.Hidden Layers:
The first hidden layer: A fully connected (dense) layer with 64 neurons. It uses ReLU (Rectified Linear Unit) as the activation function.
The second hidden layer: Another fully connected layer with 64 neurons, also using ReLU (Rectified Linear _nit) as the activation function.
3.Output Layer:
The output layer is a fully connected layer with the number of neurons equal to the number of possible actions (which is also equal to the number of stocks, ‘output_dim‘). This layer does not have an activation function since we are predicting Q-values for each action.
The architecture can be summarized as follows:

- Input Layer: Size = Number of stocks (N)
- Hidden Layer 1: Fully connected, 64 neurons, ReLU activation
- Hidden Layer 2: Fully connected, 64 neurons, ReLU activation
- Output Layer: Fully connected, Number of neurons = Number of stocks (N) (no activation)

4.2 Input to the DNN

The input to the Deep Q-Network (DQN) is a tensor representing the continuous yield values of the stocks. Specifically:

- State Representation:
  - Each stock has a continuous yield value, which represents the observed return of the stock.This is a real number (continuous value).
  - For N stocks, the state is represented as a vector of length N, where each element is a continuous value corresponding to the yield of each stock.
  - This vector of continuous values is converted to a tensor and is the input to the DQN.

4.3 Output of the DNN

The output of the Deep Q-Network (DQN) is a vector of Q-values, where each Q-value corresponds to the expected future reward of taking a specific action in the given state. Specifically:

- Q-Values:
  - The DQN outputs a Q-value for each possible action. In this case, the possible actions are choosing which stock to invest in.
  - For N stocks, the output is a vector of length N, where each element represents the Q-value of investing in the corresponding stock.
  - These Q-values are used to make decisions by selecting the action (stock to invest in) with the highest Q-value, which represents the highest expected future reward.

4.4 Why This Input and Output?

- Input:
  - The input state vector represents the current market situation (continuous yield values for each stock), which is the crucial information needed to make informed investment decisions. By encoding this state as a vector of continuous values, the DQN can process and learn from the variations in stock yields.
- Output:
  - The output Q-values represent the expected future rewards for each possible action (investing in each stock). By providing a Q-value for each action, the DQN enables the agent to make decisions that maximize long-term profit based on the current state of the market.
  - This approach aligns with the principles of Q-learning, where the goal is to learn a policy that maximizes the cumulative discounted reward over time.

4.5 Relative Code

```python
class DQN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, output_dim)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

- ‘input_dim‘: Number of input features, which corresponds to the number of stocks.
- ‘output_dim‘: Number of possible actions, which also corresponds to the number of stocks.

4.6 Evaluation of our DQN

For evaluating our model, we plotted the average Q per episode and the average reward per episode (like in the Atari game DQN paper). We then compared it to the respective plots of the Q-Learning algorithm we used in Task 2 for the same scenario, to check whether we achieve similar performance.

Setup:
Stocks: $N = 8$
Gamma (Discount Factor): $\gamma = 0.99$
Learning Rate: 10^{-3}
Epsilon (Exploration Rate): Starting from 1, multiplying itself by 0.995 at the end of every episode so that it decays over time
Number of Episodes: 1000
Transaction Cost: 0.1
Horizon T: 100
Batch Size: 32
Transition Probabilities $\&$ Rewards: Contained in file stock_params.csv

Following are the result plots from running the 2 algorithms for the same environment:

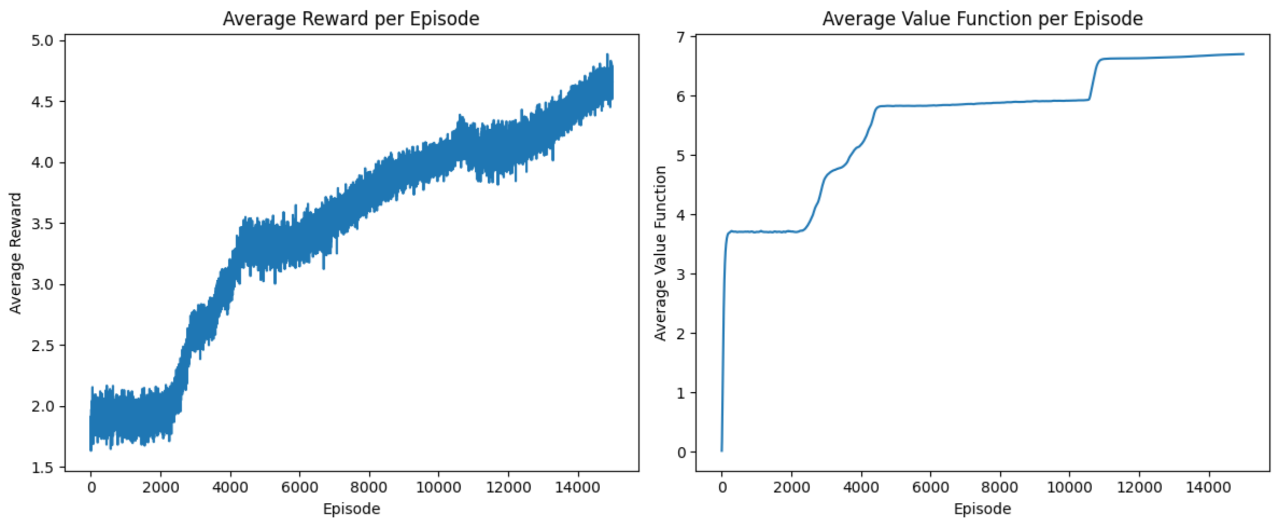

Figure 1: Plot from Task 2

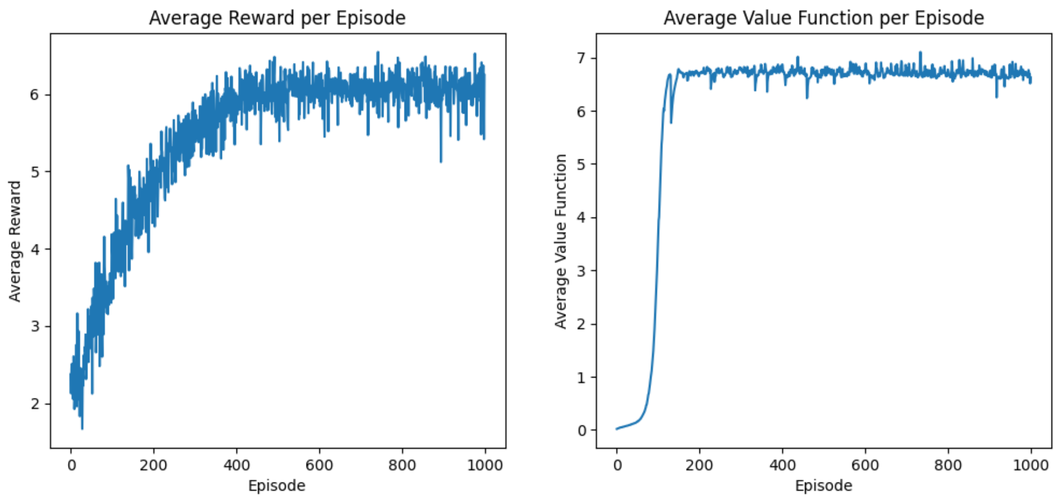

Figure 2: Plot from Task 3

We observe that our DQN converges to almost the same average reward and Q as the Q-Learning algorithm. The big difference is that our DQN managed to converge way faster than the Q-Learning algorithm (15 times smaller episode number, smaller horizon)!