Assignment 3: RL Report

Student ID:113552016 Name:陳帝嘉

1.(10%) Policy Gradient method

(1). Please read and run the sample program and try to improve the reward calculation method.

(1) 加入交易成本，以降低過度交易

```
if action != self._position.value:

            step_reward -= (self.handing_charge  + self.transaction_tex)
```

(2) 增大對於負報酬的懲罰

```
if daily_return >= 0:

             step_reward = daily_return

        else:

            step_reward = -(np.exp(np.abs(daily_return*5))-1)/5
```

(2). Please explain how you improve the reward algorithm, and how different algorithms affect the training results?

(1)效果較好，交易較穩定

(2)total return 在過程中較小回跌與波動，但是報酬較低。

2. (10%) Try to modify and compare at least three hyperparameters (neural network size, number of epochs in a batch, etc.) and explain what you observed.

(1) batch_size：如果 reward 是簡單 return，越大機器越喜歡往波動大的選擇，因此結果可能極好或極差。

(2) NN 的數目：較大的 hiden layer 可以捕捉更複雜的行為，在 traing_set 表現佳，但是在 test_set 表現不好。

多加一層也沒有好的效果。

(3) number of epochs：越大效果越好，但是訓練時間會大幅上升。

3. (15%) choose and implement one of the many RL methods such as Q Learning, Actor-Critic, PPO, DDPG, TD3, etc., and describe your implementation details.

Q-learning 的步驟：

一開始 Q 值的 table 默認 Q 值為 0，然後用 ε -greedy 更新 Q-table ，得到 Q-table。以下是 code。

```
import numpy as np

import random

from collections import defaultdict


# 超参数

ALPHA = 0.1              # 学习率

GAMMA = 0.9              # 折扣因子

EPSILON = 0.01           # ε -greedy 探索率

EPISODES = 500           # 训练回合数

STEPS = 100              # 每回合最大步数



class QLearningAgent:

    def __init__(self, action_space, observation_space):

        self.action_space = action_space

        self.observation_space = observation_space
```

```python
        self.q_table = defaultdict(float)  # 默认 Q 值为 0

    def choose_action(self, state):
        """使用 ε-greedy 策略选择动作"""
        if random.uniform(0, 1) < EPSILON:
            return self.action_space.sample()  # 随机动作
        else:
            return self._best_action(state)  # 最优动作

    def _best_action(self, state):
        """选取当前状态下的最佳动作"""
        q_values = {a: self.q_table[(state, a)] for a in range(self.action_space.n)}
        return max(q_values, key=q_values.get)  # 返回 Q 值最大的动作

    def update(self, state, action, reward, next_state):
        """更新 Q-table"""
        best_next_action = self._best_action(next_state)
        td_target = reward + GAMMA * self.q_table[(next_state, best_next_action)]
        td_error = td_target - self.q_table[(state, action)]
        self.q_table[(state, action)] += ALPHA * td_error

# 初始化环境
env = MyStockEnv(origin_df_list, window_size=10, frame_bound=(10, 1800))
```

```
# 初始化 Q-learning agent
agent = QLearningAgent(env.action_space, env.observation_space)

# 训练回合
for episode in range(EPISODES):
    state, _ = env.reset(seed=episode)  # 初始化环境和状态
    state = tuple(state.flatten())  # 将状态转为 hashable 类型

    total_reward = 0

    for step in range(STEPS):
        action = agent.choose_action(state)  # ε-greedy 选择动作
        next_state, reward, done, info = env.step(action)  # 执行动作
        next_state = tuple(next_state.flatten())  # 将状态转为 hashable 类型

        # 更新 Q-table
        agent.update(state, action, reward, next_state)

        state = next_state
        total_reward += reward
```

```
            if done:

                break


        print(f"Episode {episode + 1}/{EPISODES}: Total Reward: {total_reward:.2f}")


import pickle


with open('q_table.pkl', 'wb') as f:

    pickle.dump(agent.q_table, f)
```

4. (5%) Please specifically compare (data, graphs, etc.) the differences between the method you implemented and the Policy Gradient method, and explain their respective differences. What are the advantages and disadvantages of .

感覺 Q-learning 由於是得到一個 Q table ，比較容易收斂到一個特定的點。
每次的訓練出來的結果幾乎相同，而且似乎容易 overfitting 。在 test-set 效果並不好。

Policy Gradient 每次的收斂結果都不太相同，但是效果大多比較好。