姓名：黃展皇

學號：110621013

1. (10%) Policy Gradient 方法

請閱讀及跑過範例程式，並試著改進 reward 計算的方式。

請說明你如何改進 reward 的算法，而不同的算法又如何影響訓練結果？

在 reward 計算的部分主要改動的是這一條 code：

```
# total_reward as rewards
rewards.append(np.full(total_step, info['total_reward']))  # 設定同一個 episode 每個 action 的 reward 都是 total reward
```

在原先的策略中是考慮完一整局投資(episode)後，其得到的 total reward 就是每個 action 的 reward。而在改進 reward 算法上老師投影片提到兩種 tip，分別是 Baseline 以及 Suitable Credit (gamma)，**兩者皆有嘗試並測試，且還有兩者的結合測試**，另外 **Baseline 也有針對不同的數值進行測試(0.1、長期趨勢線 0.1634、1.0)**，自己想的方法也有兩個，分別為每個 action 只看當下 reward 的**短視近利法(single)**，以及**波段 reward(band reward)算法**，即：一段時間(波段)的利益率最佳算法應該是(賣出價格-買入價格)/持有時間，將 action 為 1 時的 reward 設為利益率(或是利益率*10，僅測試用)，action 為 0 時的 reward 設為 0。

以上 reward 算法又分別對於['Close','Open']**兩個 features** 以及 ['Shares','Amount','Open','High','Low','Close','Change','Turnover']**八個**

features 做檢驗，執行 test.py 可以得到以下：

- 原 total reward as all reward 算法：
  - 只用 2 feature："total_reward": 18.4, "total_profit": 1.1875138529091713，接近全持有僅 3 筆賣出
  - 用全 8 feature："total_reward": 29.4, "total_profit": 1.0338217658470048，接近全持有僅 17 筆賣出
- Baseline 算法：
  - 用全 8 feature，Baseline=0.1634："total_reward": 29.4, "total_profit": 1.0338217658470048
- Suitable Credit (gamma)算法：
  - 只用兩 feature，gamma=0.9：output_gamma_reward："total_reward": 18.0, "total_profit": 0.28983855519801993，頻繁交易
  - 用全 8 feature，gamma=0.9："total_reward": 9.5, "total_profit": 0.19616716656007824，頻繁交易
- Baseline + Suitable Credit (gamma)算法：
  - 用全 8 feature，gamma=0.9，Baseline=0.1："total_reward": 12.3, "total_profit": 0.20515171645320274，頻繁交易
  - 用全 8 feature，gamma=0.9，Baseline=0.1634："total_reward": 9.1, "total_profit": 0.1380516136024575，頻繁

交易

- 用全 8 feature，gamma=0.9，Baseline=1.0："total_reward": 2.2, "total_profit": 0.13012830750074414，頻繁交易

- Single 算法：
  - 只用兩 feature：output_gamma_reward："total_reward": 1.3, "total_profit": 0.9542408566646335，接近全賣出 4 持有
  - 用全 8 feature："total_reward": 2.2, "total_profit": 0.16138687348747074，頻繁交易
- band reward 算法：
  - 用全 8 feature，一般波段 reward："total_reward": 26.9, "total_profit": 0.5468846400183093，偏向持有(僅數十筆賣出)
  - 用全 8 feature，波段 reward*10："total_reward":29.3, "total_profit": 0.7808031971608166，偏向持有(僅數十筆賣出)

由以上可得知：

1. total_reward 最高者為**原 total reward as all reward 算法的 8 features** 以及 Baseline 算法的 8 features (29.4)，次高為 band reward 算法的 8 features 波段 reward*10 (29.3)。
2. total_profit 大於 1.0 者有**原 total reward as all reward 算法的只用 2 feature (1.1875)** 以及 8 features (1.0338)，以及 Baseline 算法的

(1.0338)。

3. 原 total reward as all reward 算法傾向持續持有，加入多 feature 可增加賣出頻率並大幅增加 total_reward，但小幅減少 total_profit。
4. Baseline 算法在 Baseline 設為長期趨勢線的 0.1634 之下跟原 total reward as all reward 算法一模一樣。
5. Suitable Credit (gamma)算法以及 Baseline + Suitable Credit (gamma)算法會有非常頻繁的交易，並且表現不佳。
6. Single 算法在 2 feature 情況傾向不持有，有不錯的 total_profit；8 feature 情況下頻繁交易，效率變差很多。
7. band reward 算法表現近似於原 total reward as all reward 算法，且在買賣頻率上在資料區間有數十筆波段買賣，算是在實務操作上相對合理的算法。

2. (15%) 試著修改與比較至少三項超參數（神經網路大小、一個 batch 中的回合數等），並說明你觀察到什麼。

超參數選擇更改神經網路大小、一個 batch 中的回合數以及 frame_bound 的選取區間。

* 神經網路大小部分將原先的神經網路設計分別淺化/加深並定義為 small/big NN，分別對 1.中表現較佳的三種模型(model、

model_all_feature、model_band_10times)進行不同 NN 深度的修改：

```python
def __init__(self, input_dim):
    super().__init__()
    # original
    '''
    self.fc1 = nn.Linear(input_dim, 16)
    self.fc2 = nn.Linear(16, 16)
    self.fc3 = nn.Linear(16, 2)
    '''
    # big
    self.fc1 = nn.Linear(input_dim, 64)
    self.fc2 = nn.Linear(64, 32)
    self.fc3 = nn.Linear(32, 16)
    self.fc4 = nn.Linear(16, 8)
    self.fc5 = nn.Linear(8, 4)
    self.fc6 = nn.Linear(4, 2)
    #small
    '''
    self.fc1 = nn.Linear(input_dim, 4)
    self.fc2 = nn.Linear(4, 2)
    '''
```

```python
def forward(self, state):
    # original
    '''
    hid = torch.tanh(self.fc1(state))
    hid = torch.tanh(self.fc2(hid))
    return F.softmax(self.fc3(hid), dim=-1)
    '''
    # big
    hid = torch.tanh(self.fc1(state))
    hid = torch.tanh(self.fc2(hid))
    hid = torch.tanh(self.fc3(hid))
    hid = torch.tanh(self.fc4(hid))
    hid = torch.tanh(self.fc5(hid))
    return F.softmax(self.fc6(hid), dim=-1)
    #small
    '''
    hid = torch.tanh(self.fc1(state))
    return F.softmax(self.fc2(hid), dim=-1)
    '''
```

- model_smallNN：“total_reward”: 30.7, “total_profit”: 1.2829499785446996，僅 1 筆賣出
- model_bigNN：“total_reward”: 26.6, “total_profit”: 0.9544167816572734，22 筆賣出
- model_all_feature_smallNN：“total_reward”: 16.5, “total_profit”: 0.26126994406135545，傾向多持有，頻繁交易
- model_all_feature_bigNN：“total_reward”: 29.7, “total_profit”: 1.2121626206786262，僅 6 筆賣出
- model_band_10times_smallNN：“total_reward”: 29.7, “total_profit”: 0.598246618639677，傾向多持有
- model_band_10times_bigNN：“total_reward”: 11.5, “total_profit”:

0.5324529337750625，傾向多持有

由以上可得知：

- 對於原 total reward as all reward 算法 2 feature 而言，原先 "total_reward": 18.4, "total_profit": 1.1875138529091713，total_reward 無論是 NN 變大/變小皆有提升，total_profit 則是在小一點的 NN 有較好的表現。
- 對於原 total reward as all reward 算法 8 feature 而言，原先 "total_reward": 29.4, "total_profit": 1.0338217658470048，total_reward 在小 NN 時降低很多，大 NN 時有提升，total_profit 表現類似，證明**越多 feature 需要越深 NN**。
- 對於 band reward 算法 8 feature 而言，原先"total_reward": 18.4, "total_profit": 1.1875138529091713，total_reward 無論是變大/變小皆有提升，total_profit 則是在小一點的 NN 有較好的表現。

* 一個 batch 中的回合數部分選擇改變 EPISODE_PER_BATCH 的數量，改成 1 次(變少)與 10 次(變多)，分別對 1.中表現較佳的三種模型(model、model_all_feature、model_band_10times)進行修改：

```
EPISODE_PER_BATCH = 5  # 每蒐集 5 個 episodes 更新一次 agent --> 回合數(1,5,10)
```

- model_EPISODEPERBATCH_1：“total_reward”: -5.8, “total_profit”: 0.14681082002860824，無傾向
- model_EPISODEPERBATCH_10：“total_reward”: 17.7, “total_profit”: 1.1442317050596507，僅 5 筆賣出
- model_all_feature_EPISODEPERBATCH_1：“total_reward”: 17.7, “total_profit”: 0.2558831259190814，無傾向
- model_all_feature_EPISODEPERBATCH_10：“total_reward”: 19.0, “total_profit”: 0.441220550476819，傾向持有
- model_band_10times_EPISODEPERBATCH_1：“total_reward”: 23.4, “total_profit”: 0.5406089627959313，傾向持有
- model_band_10times_EPISODEPERBATCH_10：“total_reward”: 23.7, “total_profit”: 0.7560904654410261，傾向持有

由以上可得知：

- 對於原 total reward as all reward 算法 2 feature 而言，原先 “total_reward”: 18.4, “total_profit”: 1.1875138529091713，total_reward 在 EPISODEPERBATCH_1 甚至是負的，相當不穩定，total_profit 則是差很多；EPISODEPERBATCH_10 則與原先的 EPISODEPERBATCH_5 差不多(total_reward 微降)。
- 對於原 total reward as all reward 算法 8 feature 而言，原先 “total_reward”: 29.4, “total_profit”: 1.0338217658470048，

total_reward 和 total_profit 在 EPISODEPERBATCH_1 或 EPISODEPERBATCH_10 表現都不佳。

- 對於 band reward 算法 8 feature 而言，原先"total_reward": 18.4, "total_profit": 1.1875138529091713，total_reward 和 total_profit 在 EPISODEPERBATCH_1 或 EPISODEPERBATCH_10 表現都更好。

* frame_bound 選取區間部分改變 MyStocksEnv 的 frame_bound，從(1000, 1500)改成(1500, 2000)(換選取範圍), (1000, 1100) (縮小選取範圍)，分別對 1. 中表現較佳的三種模型(model、model_all_feature、model_band_10times)進行修改：

```
改frame_bound((1000, 1500)->(1500, 2000), (1000, 1100))(換資料區間跟縮小資料區間)(避開測試的(500, 1000)以及(2000, 2500))
env = MyStocksEnv(df=STOCKS_TSMC, window_size=10, frame_bound=(1000, 1100))
```

- model_change："total_reward": 26.6, "total_profit": 0.9072735422869351，22 筆賣出
- model_minify："total_reward": 27.1, "total_profit": 0.8905167668958032，26 筆賣出
- model_all_feature_change："total_reward": 1.6, "total_profit": 0.2491160007658047，傾向持有
- model_all_feature_minify："total_reward": 20.9, "total_profit": 0.5909762568405135，傾向持有

- model_band_10times_change：“total_reward”: 23.9, “total_profit”: 0.5360988541428936，傾向持有
- model_band_10times_minify：“total_reward”: 20.8, “total_profit”: 0.5221147988667466，傾向持有

由以上可得知：

- 對於原 total reward as all reward 算法 2 feature 而言，原先 “total_reward”: 18.4, “total_profit”: 1.1875138529091713，無論是 change 還是 minify 的情況，total_reward 有顯著的提升但 total_profit 有所下降。
- 對於原 total reward as all reward 算法 8 feature 而言，原先 “total_reward”: 29.4, “total_profit”: 1.0338217658470048，無論是 change 還是 minify 的情況，total_reward 和 total_profit 表現都不佳，尤其以 change 更為嚴重。
- 對於 band reward 算法 8 feature 而言，原先“total_reward”: 18.4, “total_profit”: 1.1875138529091713，無論是 change 還是 minify 的情況，total_reward 有小幅的更好，但 total_profit 從 1.0 以上降到 0.5 左右，表現顯著下降。

3. (15%) 請同學們從 Q Learning、Actor-Critic、PPO、 DDPG、TD3 等眾多 RL 方法中擇一實作，並說明你的實作細節。

在這邊採用 stable_baselines3 的演算法套件(A2C, DQN, PPO)，並參考針對 gym_anytrading 的範例訓練，其中 feature 依然分為['Close', 'Open'] 2 feature 以及['Shares','Amount','Open','High','Low','Close','Change','Turnover'] 8 feature 分別嘗試，my_process_data 以及 MyStocksEnv 跟 PolicyGradientAgent 訓練時一樣，而訓練僅用到三行：

```
model = DQN('MlpPolicy', env, verbose=1) #A2C, DQN, PPO
model.learn(total_timesteps=100000)
model.save("DQN_MlpPolicy_100000")
```

其中模型可能是(A2C, DQN, PPO)，而 Policy 採用'MlpPolicy'，total_timesteps 皆為 100000。結果如下：

- A2C_MlpPolicy_100000："total_reward": 7.0, "total_profit": 0.1503470382827645，持有/賣出均衡
- A2C_MlpPolicy_100000_all_feature："total_reward": -20.5, "total_profit": 0.13058173352993843，持有/賣出均衡
- DQN_MlpPolicy_100000："total_reward": 32.0, "total_profit": 0.3364423991078638，傾向賣出，123 筆持有
- DQN_MlpPolicy_100000_all_feature："total_reward": 73.5, "total_profit": 0.41885424962328466，傾向賣出，107 筆持有

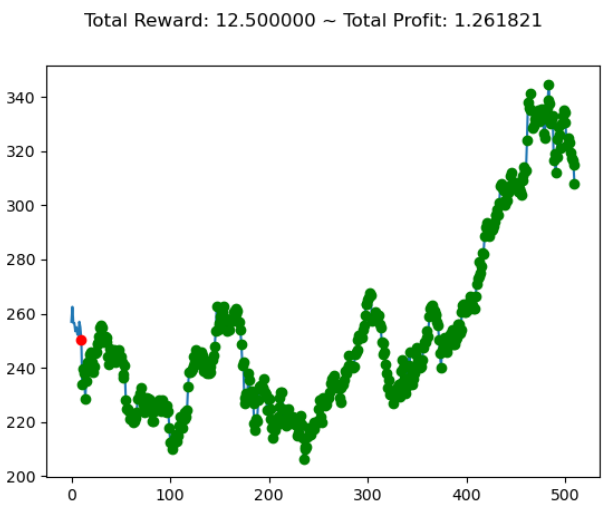
- PPO_MlpPolicy_100000："total_reward": 34.0, "total_profit": 0.3965528057191327，傾向持有，82 筆賣出
- PPO_MlpPolicy_100000_all_feature："total_reward": 3.0, "total_profit": 0.13459356189174448，持有/賣出均衡

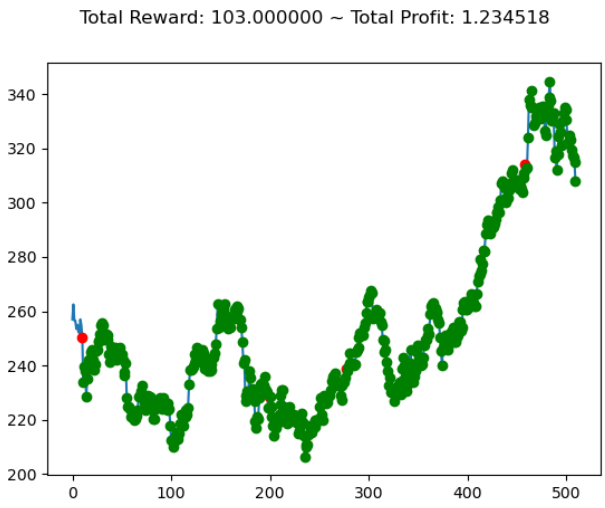
由以上可得知：

- 對於 A2C 以及 PPO 的_all_feature(8 feature)情況，total_reward 以及 total_profit 相較於 2 feature 顯著變差，甚至會有負的 total_reward，因此這兩者接下來只需討論 2 feature 情況，不過 DQN 中 total_reward 有顯著上升。
- total_reward 以及 total_profit 最高者為 DQN 8 feature(73.5, 0.4189)，其次 PPO 2 feature 也有(34.0, 0.3966)。
- 各種模型產生的交易傾向也不盡相同，例如 A2C 會有頻繁交易的情況，DQN 傾向賣出，PPO 則在頻繁交易的情況下傾向持有。

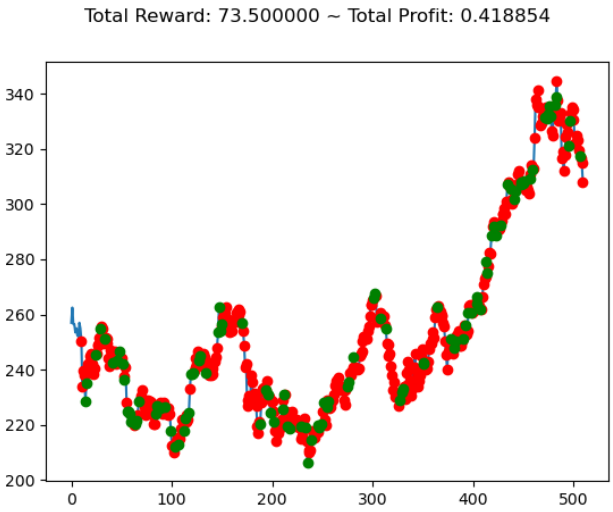
4. (10%) 請具體比較（數據、作圖等）你實作的方法與 Policy Gradient 方法有何差異，並說明其各自的優缺點為何。

最佳的 Policy Gradient 做法：最高 total_reward，同時也是最高 total_profit：2 feature 用小 NN(30.7, 1.2829)、8 feature 用大 NN(29.7, 1.2122)，下兩圖為測試資料區間的買賣情況，左側為 2 feature 用小 NN(1 筆賣出)，右側為 8 feature 用大 NN(6 筆賣出)：





而套件方法中 total_reward 以及 total_profit 最高者為 DQN 8 feature(73.5, 0.4189)，下圖為測試資料區間的買賣情況(傾向賣出，107 筆持有)：



比較 Policy Gradient 做法以及套件方法可以知道，對於 total_reward 來說套件方法 73.5>30.7，顯然對於 2000~2500 的 frame_bound 而言套件方法較好；而對於 total_profit 來說套件方法 0.4189<1.2829，顯然對於 2000~2500 的 frame_bound 而言 Policy Gradient 方法較好。

這直接讓我們陷入兩難，是應該以 total_reward 還是 total_profit 做為參考依據呢？我們發現 total_profit 本身是有收取交易的手續費的，也就是說在同樣的

reward 之下，更頻繁的交易會導致真實利潤 profit 較低。因此如果你是一位長期投資者(以年為單位)，可用 Policy Gradient 做法避免多次操作；反之若是波段餐作者(以天/禮拜/月為單位)，則可用套件方法得到更為頻繁的交易操作建議。