(2022W) COMP-5313-WDE/WB - Artificial Intelligence Computer Science

Project 1

In this Project I have used the tensor flow of version 1.15, matplotlib of 3.1, installed some other libraries mpl_finance, stable_baselines. Firstly, we imported all the required package to run the code.

Import Libraries

[25]:

```
import random
import json
import gym
from gym import spaces
import pandas as pd
import numpy as np
import pandas as pd
import datetime as dt
from stable_baselines.common.policies import MlpPolicy
from stable_baselines.common.vec_env import DummyVecEnv
from stable_baselines import PPO2
```

Required packaged to run code

[26]:

```
from stable_baselines import A2C
import matplotlib
#matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib import style
from mpl_finance import candlestick_ochl as candlestick
```

In this project I have used the dataset of Microsoft (MSFT.csv) for the stock trading.

Read Data from CSV

In [27]:

```
df = pd.read_csv('MSFT.csv')
df = df.sort_values('Date')
```

After that parameters were passed for graph and A stock trading visualization using matplotlib made to render OpenAI gym environments

graph trading

```
In [28]:
class StockTradingGraph:
    """A stock trading visualization using matplotlib made to render OpenAI gym environments"""

    def __init__(self, df, title="Microsoft"):
        self.df = df
        self.nws = np.zeros(len(df['Date']))

        # Create a figure on screen and set the title
        fig = plt.figure()
        fig.suptitle(title)

        # Create top subplot for net worth axis
        self.nw_ax = plt.subplot2grid(
            (6, 1), (0, 0), rowspan=2, colspan=1)

        # Create bottom subplot for shared price/volume axis
        self.price_ax = plt.subplot2grid(
            (6, 1), (2, 0), rowspan=8, colspan=1, sharex=self.nw_ax)

        # Create a new axis for volume which shares its x-axis with price
        self.volume_ax = self.price_ax.twinx()

        # Add padding to make graph easier to view
        plt.subplots_adjust(left=0.11, bottom=0.24,
                            right=0.90, top=0.90, wspace=0.2, hspace=0)

        # Show the graph without blocking the rest of the program
```

Setting of the initial parameter for environment

```
]:
MAX_ACCOUNT_BALANCE = 2147483647
MAX_NUM_SHARES = 2147483647
MAX_SHARE_PRICE = 5000
MAX_OPEN_POSITIONS = 5
MAX_STEPS = 20000

INITIAL_ACCOUNT_BALANCE = 100000

LOOKBACK_WINDOW_SIZE = 15
```

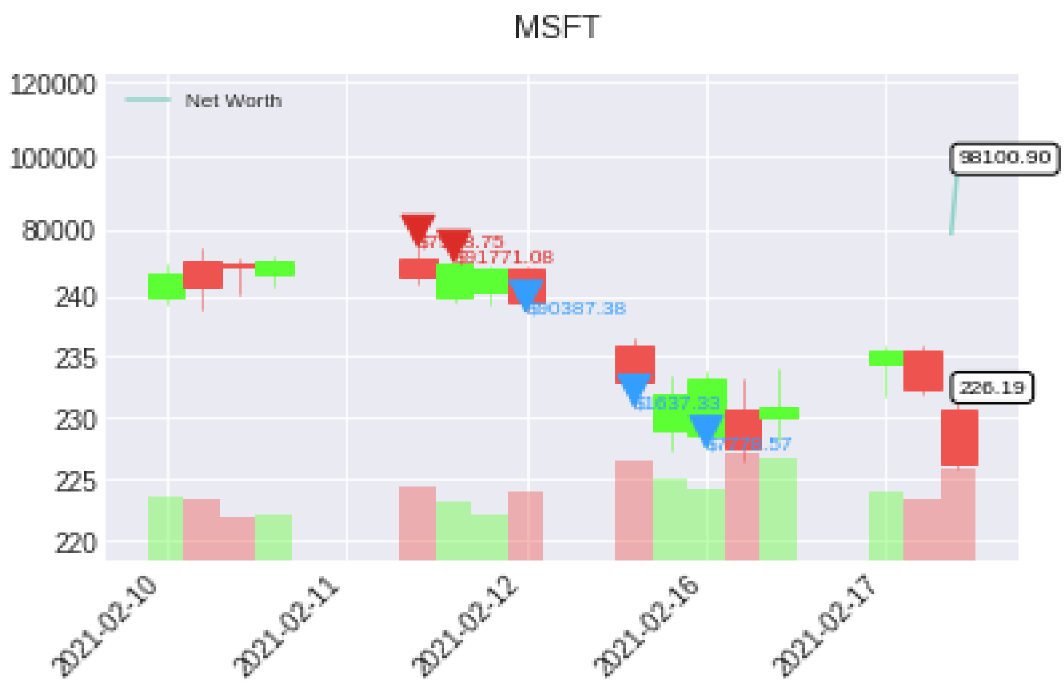
In this project I have used the A2C algorithm **A2C**, or **Advantage Actor Critic**, is a synchronous version of the A3C policy gradient method. As an alternative to the asynchronous implementation of A3C, A2C is a synchronous, deterministic implementation that waits for each actor to finish its segment of experience before updating, averaging over all of the actors. This more effectively uses GPUs due to larger batch sizes.

```
[33]: #
env = DummyVecEnv([lambda: StockTradingEnv(df)])

model = A2C(MlpPolicy, env, verbose=1)
model.learn(total_timesteps=1000)

obs = env.reset()
for i in range(200):
    action, _states = model.predict(obs)
    obs, rewards, done, info = env.step(action)
    env.render()
```

The total amount of profit which we got

```
Step: 200
Balance: 112.6257262621948
Shares held: 378 (Total sold: 2765)
Avg cost for held shares: 306.0890464167125 (Total sales value: 722749.0180940363)
Net worth: 129747.82918417621 (Max net worth: 129747.82918417621)
Profit: 29747.82918417621
```

Visualization



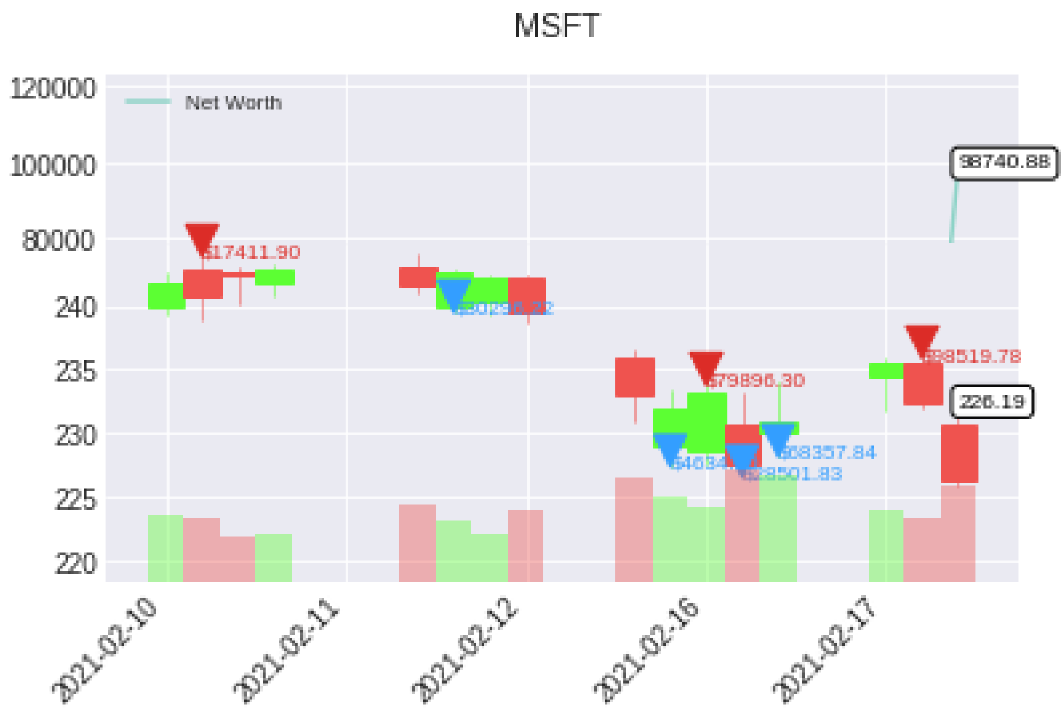
The second Algorithm which we used is PP02

```
env = DummyVecEnv([lambda: StockTradingEnv(df)])

model = PPO2(MlpPolicy, env, verbose=1)
model.learn(total_timesteps=1000)

obs = env.reset()
for i in range(200):
    action, _states = model.predict(obs)
    obs, rewards, done, info = env.step(action)
    env.render()
```

The total profit which we got is

```
Step: 200
Balance: 9.863677391942474
Shares held: 417 (Total sold: 2122)
Avg cost for held shares: 293.9111878287691 (Total sales value: 565390.0303862463)
Net worth: 142984.46787137137 (Max net worth: 142984.46787137137)
Profit: 42984.467871371366
```

The visualization is



Main Class

```
class IrisNNScratch:
    def __init__(self,x,y,hiddenLayerNeurons):
        self.input=x
        self.w1=np.random.randn(self.input.shape[1],hiddenLayerNeurons)
        self.w2=np.random.randn(hiddenLayerNeurons,3)
        self.y=y
        self.output=np.zeros(y.shape)

    def funcFForward(self):
        self.L1 = funcRelu(np.dot(self.input,self.w1))
        self.output = funcSig(np.dot(self.L1,self.w2))

    def funcBp(self):
        m=len(self.input)
        d_w2=-(1/m)*np.dot(self.L1.T,(self.y-self.output)*funcSig_derivative(self.output))
        d_w1 =-(1/m)*np.dot(self.input.T, (np.dot((self.y - self.output) * funcSig_derivative(self.output), self.w2.T)
        self.w2=self.w2 - lr*d_w2
        self.w1=self.w1 - lr*d_w1

    def predict(self,X):
        self.layert_1=funcRelu(np.dot(X,self.w1))
        return funcSig(np.dot(self.layert_1,self.w2))
```

The final accuracy is 83%

Accuracy

```
output=network.predict(testX)
Y_predict=encoOneHot.inverse_transform(output)
testY=encoOneHot.inverse_transform(testY)
accuracy=(len(Y_predict)-np.count_nonzero(testY-Y_predict))/len(Y_predict)
print("Accuracy {}".format(accuracy))
```

```
Accuracy 0.8333333333333334
```