In [1]:
```
import numpy as np
import pandas as pd
import pickle
import gym
from google.colab import files
import tensorflow as tf
from tensorflow import keras

# https://github.com/openai/gym/tree/master/gym/spaces
```

In [1]:
```
# files.upload()
```

In [2]:
```
# pickle_in = open("scalerX.pickle","rb")
# scalerX = pickle.load(pickle_in)
# pickle_in = open("scalery.pickle","rb")
# scalery = pickle.load(pickle_in)
```

In [4]:
```
Call_model = tf.keras.models.load_model('2.5Call_LR0.01.h5')
Call_data = pd.read_csv("Call_data.csv")
```

In [5]:
```
class OptionsTradingEnv(gym.Env):
    """
    An Options trading environment for OpenAI gym
    """
    # - human: render to the current display or terminal and return nothing.
    # Usually for human consumption.
    metadata = {'render.modes': ['human']}

    def __init__(self, df):
        super(OptionsTradingEnv, self).__init__()
        self.df = df
        self.days = df['START_DATE'].unique()
        self.underlying_asset_price = df['UNDERLYING'].unique()

    # private method
    def _next_observation(self):
        # Get the Options chain
        observation = self.df[self.df['START_DATE'] == self.days[self.current_st
        self.observation = observation

        return observation

    def _take_action(self, action):
        # # action in dict type with keys buy and sell
        # contracts_to_buy = action['Buy']
        # contracts_to_sell = action['Sell']
        for i in range(len(action)):
            # Assume the bought price is the ask price
            options_price = self.observation['ASK'].iloc[action[i]]
            if self.balance >= options_price:
                contract = {
```

```
                    'START_DATE': self.observation['START_DATE'].iloc[action[i]]
                    'SKRIKE': self.observation['SKRIKE'].iloc[action[i]],
                    'ASK': options_price,
                    'END_DATE': self.observation['END_DATE'].iloc[action[i]]
                }
                self.Bought_contracts.append(contract)
                self.balance -= options_price

    def step(self, action):
        # Execute one time step within the environment
        self._take_action(action)
        self.current_step += 1

        if self.current_step >= len(self.days):
            done = True
            return [], self.net_worth, done

        number_of_contracts = len(self.Bought_contracts)
        contracts_to_sell = []
        contracts_profit = 0
        if number_of_contracts > 0:
            for i in range(number_of_contracts):
                # Profit for call options
                profit = max(0, self.underlying_asset_price[self.current_step] -
                if self.Bought_contracts[i]['END_DATE'] == self.days[self.curren
                    # This is at the expiration date
                    self.balance += profit
                    # Delete the contract from the list
                    contracts_to_sell.append(i)
                elif profit - self.Bought_contracts[i]['ASK'] > 0:
                    # Exercise the contract with probability ACT_RATE
                    if np.random.binomial(n = 1, p = self.act_rate) == 1:
                        self.balance += profit
                        contracts_to_sell.append(i)
                else:
                    contracts_profit += profit

        self.net_worth = self.balance + contracts_profit
        # Delete all the exercised contracts
        self.Bought_contracts = [self.Bought_contracts[i] for i in range(number_

        done = self.net_worth <= 0
        obs = self._next_observation()

        return obs, self.net_worth, done

    def reset(self):
        # Reset the state of the environment to an initial state
        self.balance = INITIAL_ACCOUNT_BALANCE
        self.net_worth = INITIAL_ACCOUNT_BALANCE
        self.act_rate = ACT_RATE

        # Set the current step to 0
        self.current_step = 0
        self.Bought_contracts = []

        return self._next_observation()
```

```
    def render(self, mode = 'human', show = False):
        # Render the environment to the screen
        print(self.current_step)
        # print(self.Bought_contracts)
        print(self.balance)
        print('The current net worth is', self.net_worth)
```

In [6]:

```
INITIAL_ACCOUNT_BALANCE = 1000
Features = ['UNDERLYING', 'SKRIKE', 'MATURITY', 'DELTA', 'BID', 'ASK', 'IMPLIED_

profit_list = []
for ACT_RATE in np.arange(0.5, 1, 0.05):
    profit = []
    for _ in range(20):
        Env = OptionsTradingEnv(Call_data)
        cur_state = Env.reset()
        done = False
        while not done:
            X = scalerX.transform(cur_state[Features].values)
            Options_price_pred = scalery.inverse_transform(Call_model.predict(X)
            # Buy undervalued call options
            price_diff = Options_price_pred.reshape(-1) - cur_state['ASK'].value
            if sum(price_diff > 0) > 5:
                action = np.argsort(price_diff)[::-1][:5]
            else:
                action = np.argsort(price_diff)[::-1][:sum(price_diff > 0)]
            cur_state, NETWORTH, done = Env.step(action)
            # Env.render()
        profit.append(Env.net_worth - INITIAL_ACCOUNT_BALANCE)
    profit_list.append((np.mean(profit), np.std(profit)))
profit_list
```

Out[6]:

```
[(-76.15100000000008, 2.363924491179844),
 (-76.59350000000016, 1.8511571381165706),
 (-76.59100000000015, 2.7067340098354498),
 (-76.69950000000016, 2.015858315953786),
 (-77.7715000000002, 2.6050705844563664),
 (-77.89600000000021, 2.3207464316464987),
 (-77.25650000000023, 1.7559364310817267),
 (-77.82950000000021, 1.677505514148864),
 (-77.94050000000023, 1.3209295022823562),
 (-78.21600000000022, 0.8490606574326406)]
```

In [7]:

```
profit_list = []
for ACT_RATE in np.arange(0.5, 1, 0.05):
    profit = []
    for _ in range(20):
        Env = OptionsTradingEnv(Call_data)
        cur_state = Env.reset()
        done = False
        while not done:
            action = np.random.choice(np.arange(cur_state.shape[0]), size = 5, r
            cur_state, NETWORTH, done = Env.step(action)
            # Env.render()
        profit.append(Env.net_worth - INITIAL_ACCOUNT_BALANCE)
```

```
    profit_list.append((np.mean(profit), np.std(profit)))
profit_list
```

Out[7]:
```
[(-593.5954999999997, 70.17793873255324),
 (-600.2774999999998, 39.35178634763612),
 (-589.1614999999996, 58.85339822601583),
 (-594.0299999999997, 56.377465090229144),
 (-589.5544999999996, 50.5994027113957),
 (-599.4804999999997, 47.437634792957354),
 (-565.3694999999996, 47.03255335137562),
 (-612.7494999999997, 53.4625501145427),
 (-592.6919999999997, 51.0075331299212),
 (-592.3119999999997, 35.93808392221267)]
```