Stock Prediction Using q-learning Agent

Problem Statement:-

You are given the stock price till the (t-1)^th day and you need to predict whether the stock price would go up,down, or would remain sideways on the t^th day,
So the current state is the close price list till the (t-1)-day and the number of stocks we have purchased on this timestamp
Your action would be:

1. You should buy the stock(which means we predict the stock price to go up)
2. You should sell the stock(which means we predict the stock price to go down)
3. You should hold on the position (which means do nothing)

First of all there a few variables

1. MAX_TRANSACTIONS = k, maximum number of time we can do a buy after a selling
2. TOTAL_MONEY = k2 , total money that we have in cash(virtual)
3. CURRENT_STOCKS_BOUGHT = k3 , the number of stocks we have bought currently
4. CURRENT_TRANSACTION_COUNT = k4, number of buys done before selling - You can only do the buy if CURRENT_TRANSACTION_COUNT<MAX_TRANSACTIONS

Whenever you do a buy the amount with which you buy the stocks are x= TOTAL_MONEY/(MAX_TRANSACTIONS- CURRENT_TRANSACTION_COUNT) and the number of stocks bought in this buy signal = x/STOCK_PRICE at that moment.
Whenever you do sell just increase your TOTAL_MONEY by CURRENT_STOCKS_BOUGHT*STOCK_PRICE and do the CURRENT_TRANSACTION_COUNT=0Now when to do a buy or when to do the sell:

1. Do the buy whenever there is a buy signal and when CURRENT_TRANSACTION_COUNT<MAX_TRANSACTIONS
2. When there is a sell signal from DQN , just sell any stocks if you had purchased if any
3. When there is a hold signal , do nothing

Reward: It is the profit or the loss that happens in any transactions
DataSet: Use the 2009 January-2017 December data for training and then use the 2018 January-2019 December data to do the test(and report the profits made in it)
Some more details for the reward function:

1. If you are making the trade, then reward = profit/loss in that transaction.
2. If you are not holding any stock, then your reward function should penalize if there was a profit-earning opportunity and it missed it. Similarly, it should get a positive reward if there was a loss occurring, and your model did not make any trade.

Stock Prediction Using q-learning Agent

Reinforcement Learning:-

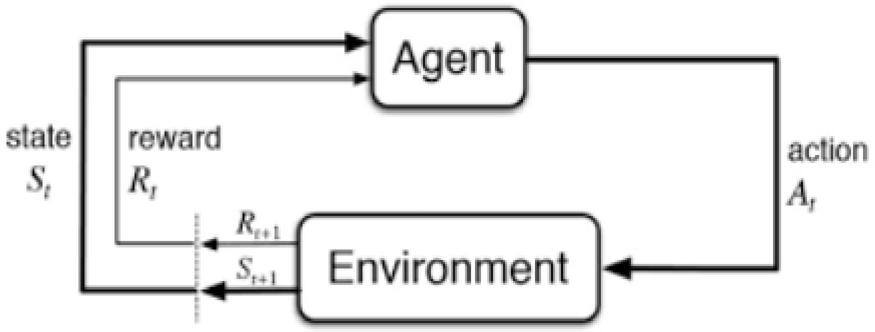
Reinforcement learning is another type of machine learning besides supervised and unsupervised learning. This is an agent-based learning system where the agent takes actions in an environment where the goal is to maximize the reward. Reinforcement learning does not require the usage of labeled data like supervised learning.
Reinforcement learning works very well with less historical data. It uses the value function and calculates it based on the policy that is decided for that action.
Reinforcement learning is modeled as a Markov Decision Process (MDP):

- An Environment E and agent states S
- A set of actions A taken by the agent
- P(s,s')=>P(st+1=s'|st=s,at=a) is the transition probability from one state s to s'
- R(s,s') – Immediate reward for any action

Reinforcement Learning Environment:-

MDP for Stock Price Prediction:

- Agent – An Agent A that works in Environment E
- Action – Buy/Sell/Hold
- States – Data values
- Rewards – Profit / Loss

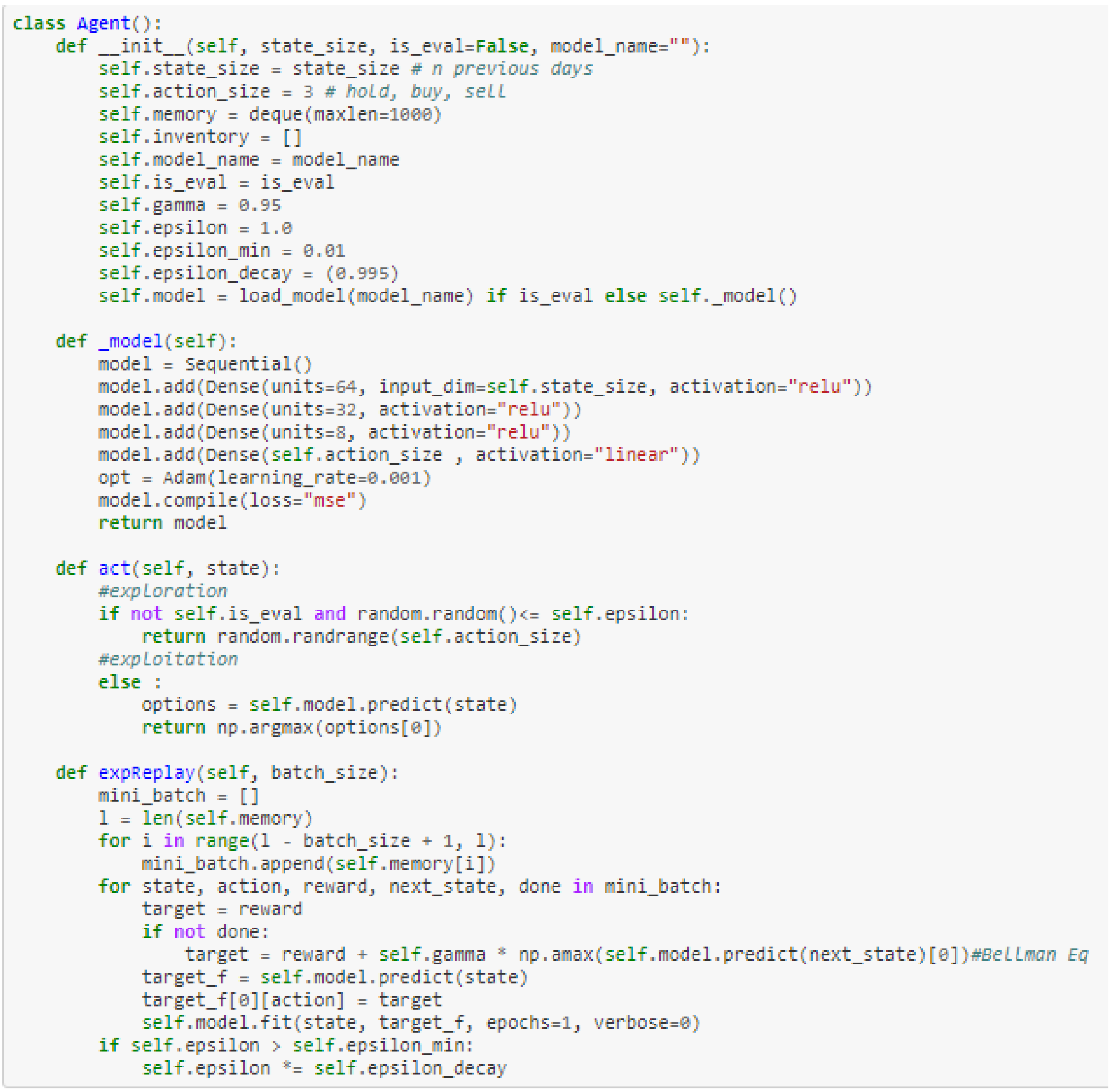


Data Extraction:-

The daily stock prices of Google are obtained from finance.yahoo.com for training and testing purposes. Stock prices between 2009 January-2017 December for training our model and stock prices between2018 January-2019 December for predicting the results(testing) with the help of our model.

Stock Prediction Using q-learning Agent

Creation of Agent:-



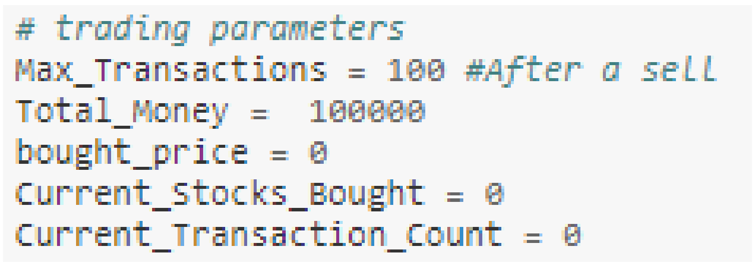
```python
class Agent():
    def __init__(self, state_size, is_eval=False, model_name=""):
        self.state_size = state_size # n previous days
        self.action_size = 3 # hold, buy, sell
        self.memory = deque(maxlen=1000)
        self.inventory = []
        self.model_name = model_name
        self.is_eval = is_eval
        self.gamma = 0.95
        self.epsilon = 1.0
        self.epsilon_min = 0.01
        self.epsilon_decay = (0.995)
        self.model = load_model(model_name) if is_eval else self._model()

    def _model(self):
        model = Sequential()
        model.add(Dense(units=64, input_dim=self.state_size, activation="relu"))
        model.add(Dense(units=32, activation="relu"))
        model.add(Dense(units=8, activation="relu"))
        model.add(Dense(self.action_size , activation="linear"))
        opt = Adam(learning_rate=0.001)
        model.compile(loss="mse")
        return model

    def act(self, state):
        #exploration
        if not self.is_eval and random.random()<= self.epsilon:
            return random.randrange(self.action_size)
        #exploitation
        else :
            options = self.model.predict(state)
            return np.argmax(options[0])

    def expReplay(self, batch_size):
        mini_batch = []
        l = len(self.memory)
        for i in range(l - batch_size + 1, l):
            mini_batch.append(self.memory[i])
        for state, action, reward, next_state, done in mini_batch:
            target = reward
            if not done:
                target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])#Bellman Eq
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f, epochs=1, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

The agent designs the layered neural network model to take action of either buy, sell, or hold. This kind of action it takes by looking at its previous prediction and also the current environment state. The act method is used to predict the next action to be taken. If the memory gets full, there is another method called expReplay designed to reset the memory.

Stock Prediction Using q-learning Agent

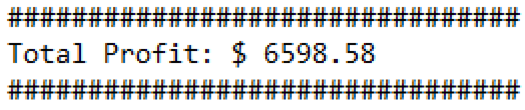
Training:-

The parameters for the trading are set manually to-



```
# trading parameters
Max_Transactions = 100 #After a sell
Total_Money =  100000
bought_price = 0
Current_Stocks_Bought = 0
Current_Transaction_Count = 0
```

Depending on the action that the model predicts, the buy/sell call adds or subtracts money. The training of the model goes by multiple episodes. The model of the final episode is saved for trading on the testing dataset. The total profit obtained at the end of final episode is **$6598.58**



```
##################################
Total Profit: $ 6598.58
##################################
```

Testing:-

The Trading parameters are re-initialized to the same value.
The model obtained after the final episode was subjected to the testing dataset.