Implementing a Reinforcement Learning Agent for Trading Stocks

Aryaman Sharma
NSUT’23, Delhi

This project involves a Deep Q-Learning Reinforcement Learning Model to estimate the optimal actions of a trader, and return the profit value incurred on a dataset.
The model uses experience replay and Double DQN with input features given by the current state of the stock, comprising 33 technical indicators, and available actions, namely buying, selling and holding a stock, while the output is the Q-value function estimating the future rewards.
The model was applied to Apple’s stocks from the US S&P 500 dataset.

Task

The target function is based on Bellman's equation, which predicts the target value for the network to be dependent on the Q-value of the next state, a discount factor (gamma) and the current action taken by the model.

The Bellman update is given as:

y = r(s,a) + gamma * Q(s’,a) ------- (1)

As per the task, two slight (independent) variations of the update were implemented.

1. The Q-value of the next state was replaced with a linearly decaying average of the Q-values of the previous N-states, including Q(s’,a).
2. Gamma was replaced with a temporal heuristic, i.e. gamma = a*e-b(t2-t1), where t2 is the timestamp of the next state and t1 is the timestamp of the current state.

Methodology

This project was partly influenced by **Quantitative Day Trading from Natural Language using Reinforcement Learning (Sawhney et al., NAACL 2021)**, in which the researchers implemented an RL agent on the US S&P 500 and China & Hong Kong dataset, including textual information from associated tweets in each of the 88 high-trade-volume stocks from the NASDAQ Exchange, and 85 China A-share stocks from the Shanghai, Shenzhen and the Hong Kong Stock Exchange respectively. Furthermore, the architecture of the network was influenced by **Mendonsa, A (2020) RL-DeepQLearning-Trading**.

Dataset preprocessing and experimental set up

The model was trained and tested on Apple’s stocks from the US S&P 500 dataset. The dataset consists of five-year price movements from 04/09/2012 to 01/09/2017, focusing on the date, open price, high price, low price, close price, adjust close price, and volume on each of the respective days. The date was set as the index for easy information retrieval. 33 technical indicators were added to the dataset from these 6 values, which were to be fed to the model. These features denoted momentum, trend, volatility and volume movements per day. A detailed list can be found in the repository.

For the model, the training data was selected as 04/09/2012 to 18/01/2017, and the testing data was from 19/01/2017 to 01/09/2017. (85-15 distribution). A window of 10 days (2 weeks) was selected. Due to time and resource constraints, the agent was trained for 7 episodes for each of the subtasks.

RL Agent

The market trading was framed as a Markov Decision Process consisting of states S, actions A, and rewards R,

where S = information available to agent (33 technical indicators), with an associated timestamp,
A = buying, holding, and selling stocks,
R = daily returns.

Model

There are various traditional methods for modelling RL agents, such as Q-learning and Deep Q Learning. These models have their roots in the Bellman Update, as follows.

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t) \, .$$

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) \, .$$

The max operator in these models uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this, we can decouple the selection from the evaluation. This is the idea behind Double Q-learning (van Hasselt, 2010). The model proposed thus transforms the target value to

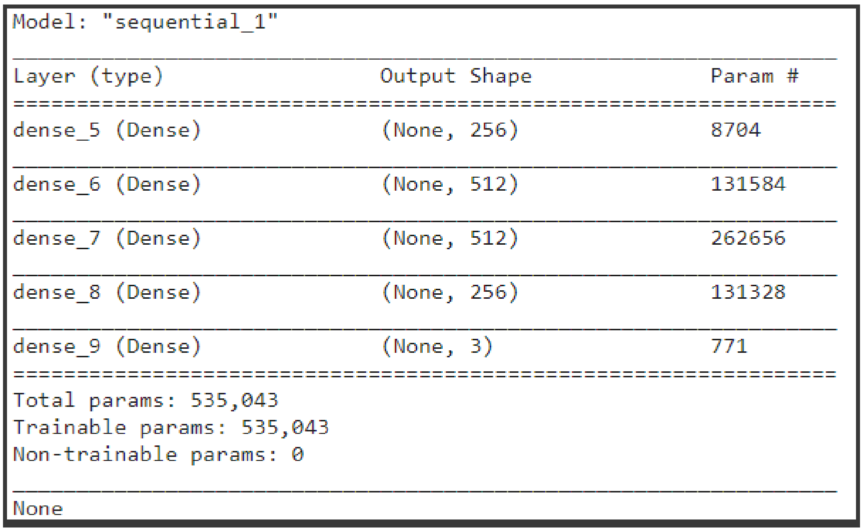
$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t) \, .$$

However, depending on the task at hand, I transformed the equation slightly.

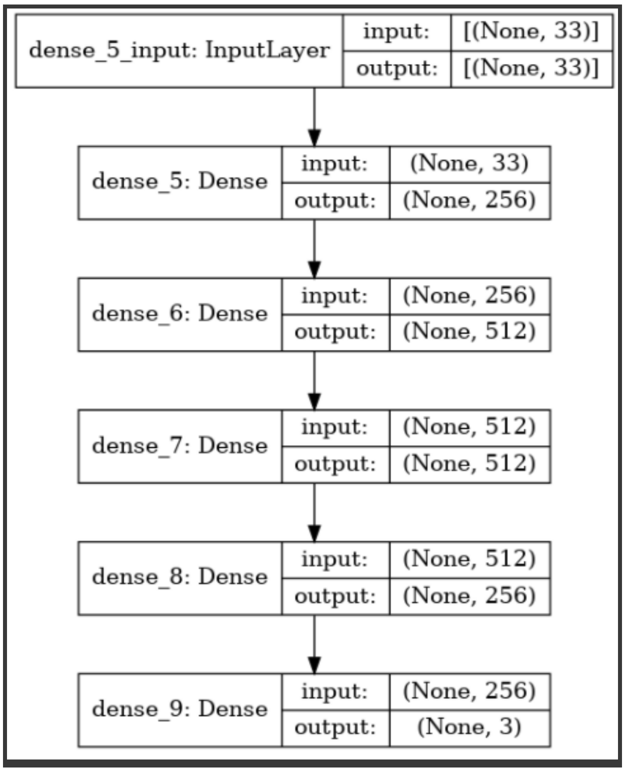
This was the basis behind my work.

1. The Agent observes the environment and take notes of the state
2. Based on that state, the Agent takes an action based on a policy.
3. The Agent therefore receives a reward from the environment.
4. The environment transitions to a new state based on the action by the Agent.
5. Repeat

The neural network was trained on the states and the list of q values of each state. Following is the model architecture :



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 256)               8704      
_________________________________________________________________
dense_6 (Dense)              (None, 512)               131584    
_________________________________________________________________
dense_7 (Dense)              (None, 512)               262656    
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 771       
=================================================================
Total params: 535,043
Trainable params: 535,043
Non-trainable params: 0
_________________________________________________________________
None
```



After every iteration, the model stored the current state and associated timestamp, next state and its timestamp, the action taken and the reward. The q- values were updated accordingly.

For task 1

The target q-value was updated by calculating the linear moving average of the past 10 states. This value was subsequently used for calculating the target.

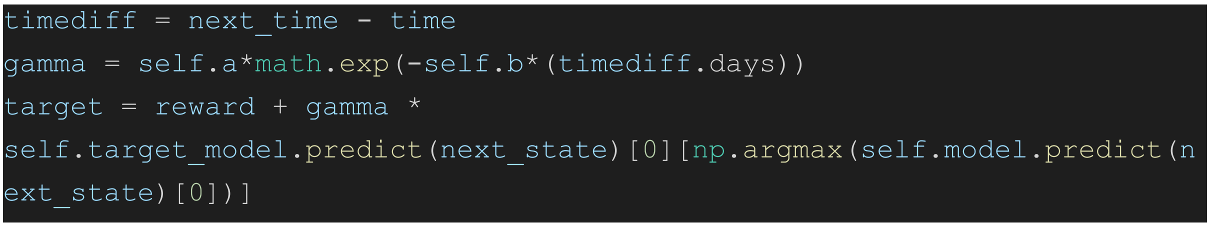
```
from source_code_task1.technical_indicators import
simple_moving_average
q_value =
self.target_model.predict(next_state)[0][np.argmax(self.model.predict(n
ext_state)[0])]
q_value = simple_moving_average(X_train, 10)
target = reward + self.gamma * q_value

q_values = self.model.predict(state)
q_values[0][action] = target
X_train.append(state[0])
y_train.append(q_values[0])
```

For task 2

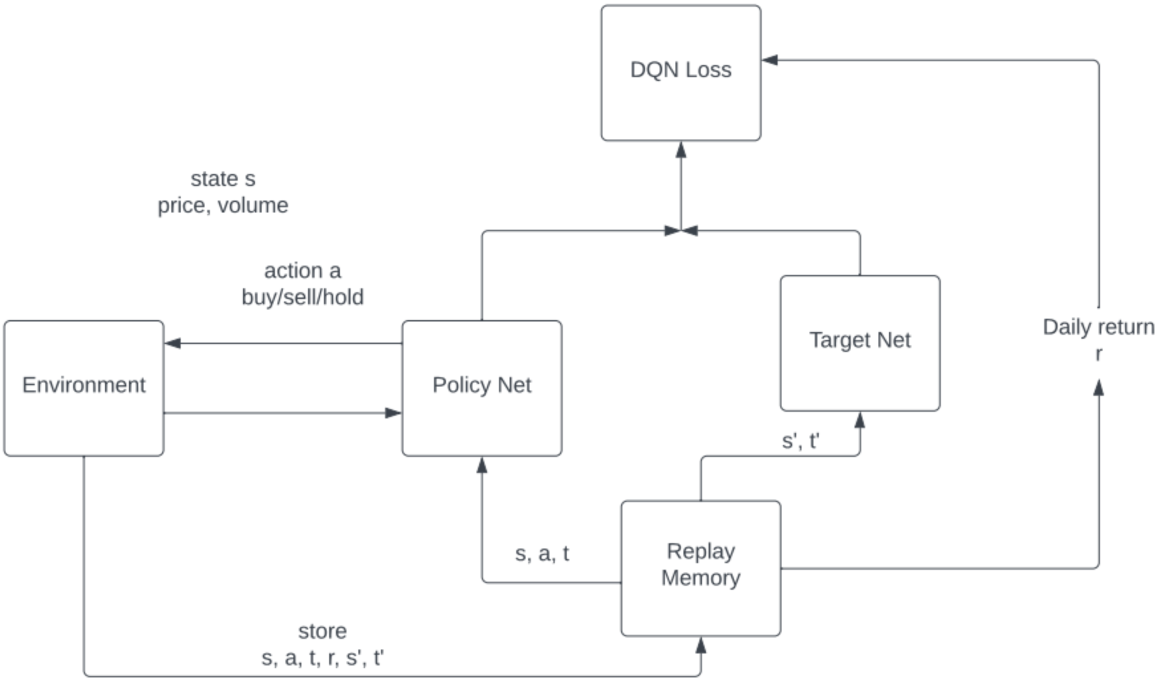
Gamma was updated with the value of the temporal heuristic :

gamma = a*e-b(t2-t1)

Here, a was chosen as 1 and b = 0.01. The value of the discount would correspondingly change with the difference between the timestamps from the remembered (stored) state.



```
timediff = next_time - time
gamma = self.a*math.exp(-self.b*(timediff.days))
target = reward + gamma *
self.target_model.predict(next_state)[0][np.argmax(self.model.predict(n
ext_state)[0])]
```



Key : s = state, s' = next state, t = timestamp (current state), t' = timestamp (next state), a = action, r = reward

Fig 1. Double Deep Q-network

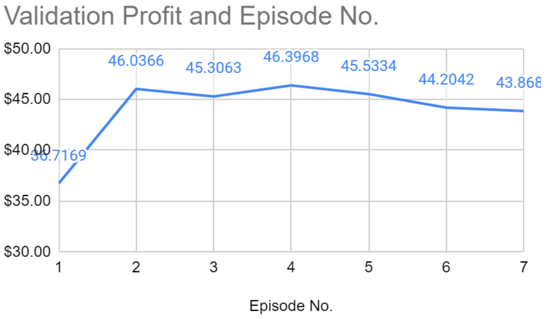
Evaluation Metrics

The model uses Sharpe Ratio for evaluation. This ratio helps investors understand the return of an investment compared to its risk.

Heuristic - (Sell) when price is above two standard deviations of simple moving average, (Buy) when price is below two standard deviations of simple moving average, or Do nothing (Hold.)

Results

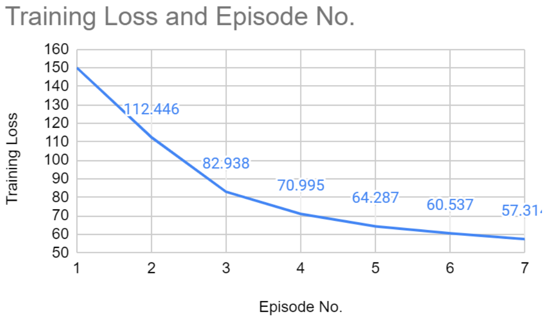
After training for 7 episodes and over 15 hours each, the training loss, training profit and validation profit was calculated.

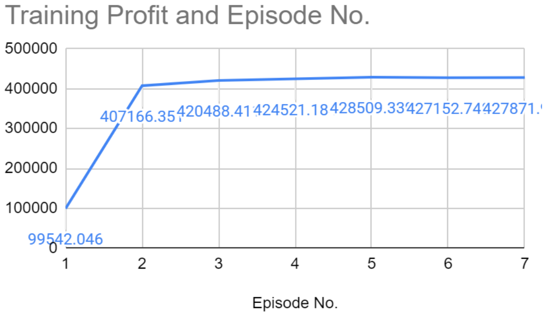


.

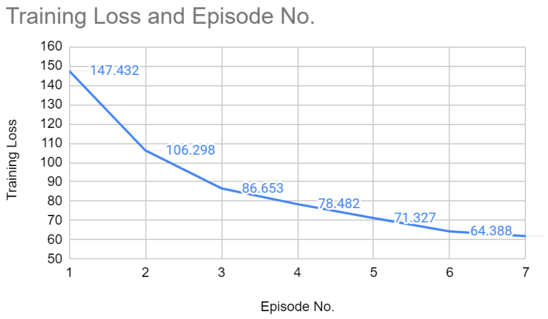
Episode No.	Training Loss	Training Profit	Validation Profit
1	150.077	99542.046	36.7169
2	112.446	407166.351	46.0366
3	82.938	420488.411	45.3063
4	70.995	424521.184	46.3968
5	64.287	428509.339	45.5334
6	60.537	427152.748	44.2042
7	57.314	427871.974	43.8687

Task 1

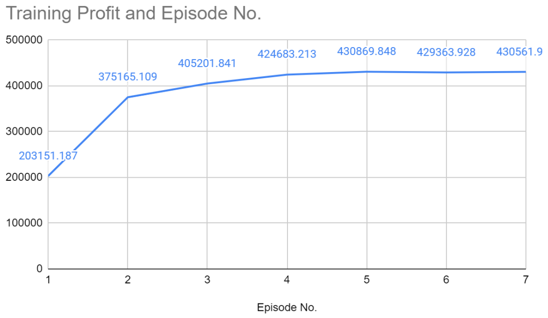




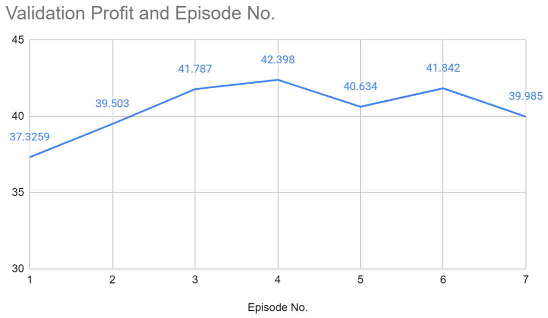
Task 2



Training Loss and Episode No.



Training Profit and Episode No.



Validation Profit and Episode No.

Episode No.	**Training Loss**	**Training Profit**	**Validation Profit**
1	147.432	203151.187	37.3259
2	106.298	375165.109	39.503
3	86.653	405201.841	41.787
4	78.482	424683.213	42.398
5	71.327	430869.848	40.634
6	64.388	429363.928	41.842
7	61.982	430561.974	39.985