Limit Order Book Analytics & Trading Strategies with Series Prediction and Reinforcement Learning

1st Kristian Litsis
MSc Data Science
University of Bristol
Bristol, United Kingdom
wg20498@bristol.ac.uk

2nd Giannis Alexandrou
MSc Data Science
University of Bristol
Bristol, United Kingdom
ou20271@bristol.ac.uk

3rd Alex Davies
MSc Data Science
University of Bristol
Bristol, United Kingdom
qj19962@bristol.ac.uk

4th Eric Tsoi
MSc Data Science
University of Bristol
Bristol, United Kingdom
hv20912@bristol.ac.uk

***Abstract*—The global financial markets are a large source of data and can be highly stochastic and volatile. Limit Order Book (LOB) exchanges have in recent years become a common form of this marketplace, and represent an ideal interface for a machine learning approach to asset trading. In this report we use a synthetic LOB dataset and present two possible approaches; series prediction with Bidirectional Long Short Term Memory (BiLSTM) networks, and learnt trading strategies with a reinforcement learning approach via Deep-Q-learning.**

Testing was performed on which data sampling rate to use for series prediction, based on the MSE of the predicted price, and we find that the optimum duration between datapoints is 1 minute, with a point-by-point MSE of 0.0097. Our Q-learning agent was similarly tested against sampling rates, but with final profit as metric. The optimum duration was found to be 10m. Following this, we performed similar testing against γ with this $\mathbf{6hr^{-1}}$ sample rate, and the optimum value was found to be $\gamma = 0.6$.

We find some moderate success with each approach. Series prediction, with our own trading strategy based on these predictions, generated a profit of 72% when buying at minimum ask and selling at maximum bid. Our Q-learning agent generated smaller profits at 5.4%, but this agent bought and sold at the mean price of transaction in the data, so this still represents a successful agent. The agent demonstrated effective trading strategies, buying at low points and selling at high, and should generate considerable profit if applied to asset prices with overall increase.

***Index Terms*—Data prepossessing, Time-series prediction, Recurrent Neural Network, Reinforcement learning, Q-learning Spark streaming, Long Short-term Memory**

Github link: https://github.com/lkrist01/Team26.git

I. Introduction

In order to perform data analytics for level2 data using Long short-term memory, the dataset will be put under statistical analysis to identify the distribution of the data. The dataset is then put through preprocessing techniques to clean the given data. The same clean dataset is then used in both BiLSTM and Deep-Q-Learning implementations to ensure fair comparison. This paper will be break down the different aspects of the project into the following topics: Related work (review of work that has inspired our project), Challenges (problems that we encountered and needed to overcome), Data preprocessing steps (the preliminary processing of the data to prepare for processing), Feature Engineering and Selection (the chosen features to be used when modelling based on client request), spark streaming (Processing tool used for the dataset files), Methods Attempted (the different steps and methodology used throughout the project), Model Evaluation and Visualization of Profit (Evaluating the methodology of which was implemented and showing the visual analysation that we had done), Future work (New implementations that could improve efficiency) and conclusion (overall summary of what we found when doing this project).

II. Related Work

Previous studies had inspired process throughout the project, such as the LSTM implementation in Model-based Reinforcement Learning for Predictions and Control for Limit Order Books [1]. The authors create a profitable electronic trading agent using RL that buys and sells orders in the market. The models they create is built using previous observational data and using the RL agent, it learns the trading policy by interacting with environment model. Another study was Using LSTM in Stock prediction and Quantitative Trading [2].Three deep learning models are created which are RNN with LSTM model, RNN with LSTM and attention and RNN with Stacked-LSTM. In addition to this they add a traditional time series model with Auto regressive integrated moving average model.By using LSTM they are able to create predictions of stock prices on the next day to the volatile and what is deemed to be unpredictable market. By using LSTM they show that it out-performs other models in terms of prediction error and has a higher return in their trading strategy over other models. Another interesting paper on RL [3] also discussed the use of Q-learning as a critic based method to learn an optimal value-function for specific problem. It explores how RL is used where traditional supervised learning may not be able to, including the financial problem of FX trading. There is further demonstration on the use of LSTM-RNN used in stock prediction [4] as it shows that by using LSTM, they are able to predict 10 company's closing stock price from the regression from numerical and textual information. This is done by having it memorise the past time steps from the architecture.

III. Challenges

Throughout the project we encountered several challenges that we must overcome in order for our project to be a success. The first being that the data from the Limit Order Book file was so large that it took too long and having small memory meant that it was difficult to run when preparsing the data. The solution to this was to find an appropriate method in order to complete the preprocessing efficiently and for this we used Spark Streaming processing with Pyspark and a physical machine with access to larger memory. With this being a new topic to all of us in the group a lot of research into understanding the algorithms by selecting the best hyper parameters and features for LSTM models or defining the agent, environment and states of the problem for the RL model had to be done in order to complete the project.

IV. Data Preprocessing

The data is provided in psuedo-json format, below is one datapoint:

```
["time",2.144,["bid",[[192,4],[155,3],[87,5]]],["ask",[[800,3]]]
```

Bids and asks are recorded every 0.04s, resulting in 1500 datapoints a minute, and 720k per 8 hour trading day.
The first stage of preprocessing is to translate this data into a more standard json format. The nature of the initial data format lacks colons and often features empty arrays. This means that standard Python libraries cannot parse the files, so the lines of the files are treated as long strings, and altered using regular expressions. Following this "translation" it was possible to extract the time, bid and ask values, which were then written to a .json file. Below is a datapoint in this format:

```
{
"time": 3.056,
"bid": {"0": [192, 4], "1": [180, 2], "2": [155, 3], "3": [87, 5]},
"ask": {"0": [197, 3], "1": [198, 5]}
}
```

After this translation data could be explored. Good practise with timeseries data is to find a running mean and deviation, within a given window, and plot histograms of the individual datapoints distances from this running mean. Below we include Figure 1, demonstrating this process.

The plotted data in the histograms clearly follows a normal distribution, so it is reasonable to assume that outliers will be $|\mu_{\text{running}} - x| > 2\sigma_{\text{running}}$. There are two options for dealing with these outliers.. Firstly, individual ask and bid prices can be removed from dataponts, or alternatively these bids and ask prices can be changed to the running mean at that point. The first option is used, as both options involve effectively removing the data, but the second will still be skewed by the outliers.

Another issue in preprocessing is the size of the data. Initially files are from 600-700Mb, which presents difficulties in efficient usage, as memory becomes a constraint. This can be partially amended during preprocessing by sampling only some fraction of the bids and asks at a given frequency - for example using only one datapoint from each second - in order to reduce eventual file size without large loss of information. This is helped by how commonly bids and asks are identical in sequential timesteps, meaning that this sampling preserves the large majority of the bid and ask price movement. Indeed, it was considered that a method of reducing file size would be to remove sequential lines that were identical, but it was decided that in time series processing for neural nets it would be better to have data from regular time intervals, allowing time to be omitted as an input component.

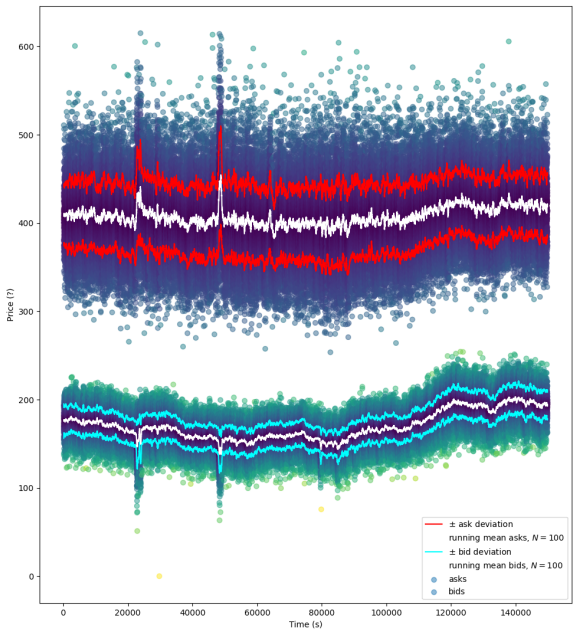

Fig. 1. Mean average and standard deviation of Ask/Bids

V. Feature Engineering & Selection

Given that this project aims to develop a machine learning tool, and the client's preference for a deep learning approach, thought must be given to what features should be passed as input to the eventual model. Thankfully the data is not, in essence, complicated; it is composed only of bid and asks prices with no missing sections.

A. Selection of Min/Max of bids and asks

The limit order book datasets are presented as a list of orders of prices, and for both BiLSTM and DQN, we make transactions in the form of bids and asks for that time. We wanted to generate profit by buying and selling this asset at the best price to potential traders, so we decided to focus on finding the minimum Ask price where we would buy and the maximum bid price where we would sell to a trader. Selecting only these features made the processing faster, and every time we had the best possible price for that specific time.

B. Extracting new features

We are using the LSTM model to approach this problem as a time-series prediction, and we wanted to add more features as inputs to get a better result. Thus we extracted new features like averages of bids/asks, the volume of the transactions and the number of requests of bids and asks until that time.

- Mean average of price

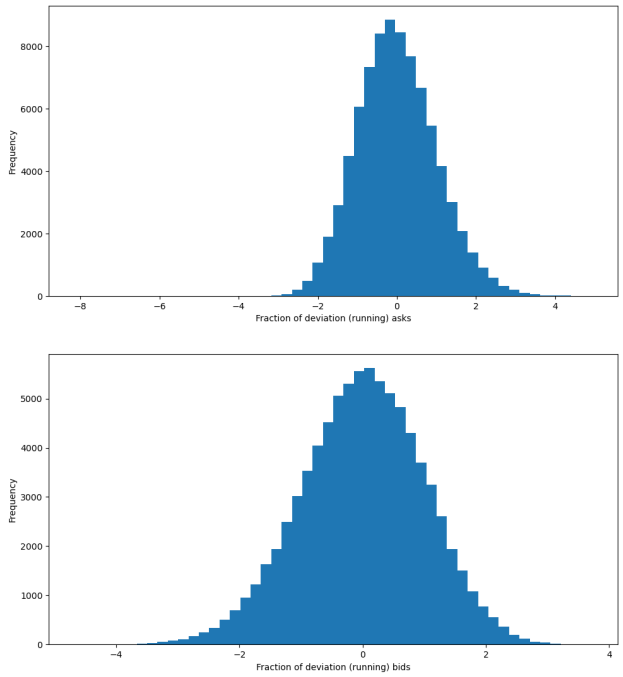

Fig. 2. A histogram of the datapoint distances from the running mean bids and ask prices, expressed as a fraction of the running deviation at that point

For every list of bids and ask that we had in our dataset, we created a new column that would return the mean average of the price. These features are essential for trading algorithms and can be used to calculate rolling-moving averages to detect trends in data.

- Volume of transactions
  We were provided datasets that contained the transactions made on a particular day for every second, so we decided to create a new feature that would count the total Volume of the asset, which is the Volume of transactions that were sold and bought until that time. Using Volume can help to make trading decisions because they somehow introduce a short-term prediction of the price.

- Number of requests
  The third feature that we extracted was the number of requests that were made for the asset at a specific time. We counted all number of asks and bids in order to detect if there is a trend in that particular time-window. These features would be used by the Recurrent neural network and will impact the short-term memory of the model.

- Mean price per transaction
  The fourth feature that we extracted was the mean price per transactions that calcuated the average price for all transactions made until that time period.

VI. Spark Streaming

After managing to extract features from a particular file in our dataset, we needed to figure out how to process all the dataset files. We needed to use a mechanism for processing all the data files and extracting the aggregated results. Spark Streaming is a tool that is an essential part of our purpose of cleaning and creating new features in a faster way. With the help of the AWS EMR service, we constructed a cluster of 3 nodes, and each node consists of 4 vCore and 16 GB of memory (m5.xlarge instance). The records from all data files are collecting as an RDD object which applies actions in parallel and stored in memory across the cluster.

Hence, with the help of the EMR cluster, we adopted the Apache Spark processing with its interface in Python, called PySpark. Through the running of script in PySpark kernel, we converted all the LOB txt files to json format, extracted the summary statistics per day to remove the outliers in ask and bid values, and aggregated all data to extract the new features per minute and second. We also aggregated the transaction data for each day deriving new features. In the end, we joined all the data per day and generated a summarised dataset representing all the data.

VII. Methods Attempted

A. Reinforcement Learning Model Implementation

An intuitive method here is to implement a reinforcement learning agent which learns a trading strategy. The "problem" of making a profit with stock trades in this format lends itself to this due to its simplicity. A reinforcement learning agent, in simple terms, outputs an action given some input data. Here these actions are, at each timestep:

- 1. Buy some volume of the asset at a given ask price
- 2. Sell some part of its existing inventory at a given bid price
- 3. Do not act

Which action it takes can is learnt through training in a simulation of the market, using our data, with profit/losses acting as reward/cost. The agent, ideally, learns to maximise profit through experience.

Q-learning is a reinforcement learning approach developed by Chris Watkins in 1989 [8], with a subsequent proof of convergence published in 1992 [9]. It was initially developed without deep learning or neural networks, instead attempting to optimize a goal metric by exploring an action space. Q-learning takes its name from this metric Q, which is written as:

$$Q(s,a) = r(s_t,a) + \gamma \max(Q(s_{t+1},a)) \tag{1}$$

Here r represents a reward, as a function of a, the action taken in the state at timestep s_t. The constant γ represents a discount on future rewards - ie the extent to which the algorithm will favour immediate rewards. A γ value of 1 leads to large consideration of future rewards, and $\gamma = 0$ leads to consideration only of rewards at the immediate time. This

algorithm is recursive, as any Q_{t+1} depends on γQ_{t+2}, so the full Q-value is:

$$Q(s_t, a_t) = \sum_{0}^{n \Rightarrow \infty} \gamma^n Q_{t+n}(s_{t+n}, a_{t+n}) \tag{2}$$

This, in a deterministic system, has been shown to converge given proper selection of γ, but in stochastic systems often does not. Furthermore, in systems with a very large number of states (such as our data) the problem of the full calculation of Q at each step becomes intractable.

A recent development, pioneered by DeepMind (later acquired by Google) [10], evades the problem of large computational overhead by instead approximating the Q value using a neural network. This is termed "Deep Q-Learning" or DQN, and we have implemented this as our reinforcement learning agent.

The neural network is fairly simple. It takes a window of stock prices as input, fixed at 90 states (one for each timestep), as a vector, and outputs an action to take (detailed above) in the form of a 3-length vector, the highest value of which represents the action to take. In order to aid simplicity of implementation, given the limited scope of this project, the network itself consists only of fully connected Dense hidden layers with *relu* activation, and is implemented using Tensorflow and Keras. The output layer is also densely connected, but has a *softmax* activation, which ensures that the sum of the values representing each possible value is 1. The three hidden layers consist of 128, 256 and 128 neurons and are included in the same order.

B. BiLSTM Model Implementation

We decided to approach the problem as a Time series prediction algorithm. The datasets that were provided contained information of bids and asks in level 2 market data. We used a bidirectional Long short-term memory implementation for predicting prices in future. We used LSTM because we were required to use a long trading strategy and obtain information long term. Thus, this approach would make it possible to store information on the prices over a long time because of its unique architecture. We used Bidirectional nodes because it would help reduce the problem of vanishing gradients that is present in the RNN models as mentioned by Hochreiter and Schmidhuber [5]. They try to reduce the gradient in order not to make changes insignificant over the long term. This is done by incorporating gradient with gate functions. The LSTM network has three gates: input, forget and update gate [5].

Below we represent the gates and their equations. The LSTM cell is described as:

$$c_t = c_{t-1} \odot f_t \odot c'_t \odot i_t$$

f_t is the forget cell those work is to delete any information form the previous cell and is calculated as:

$$f_t = \sigma(W_f * [h_{t-1}, X_t])$$

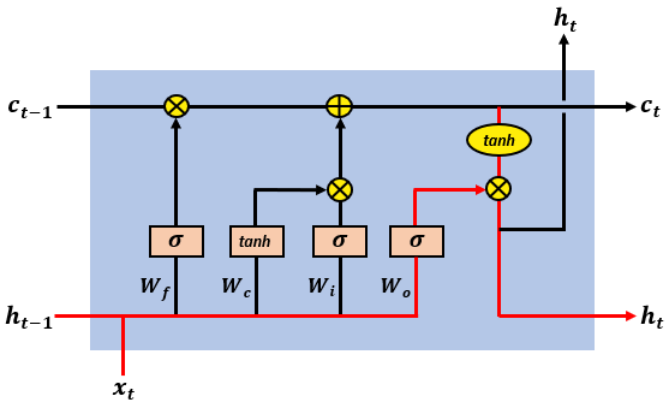

Fig. 3. LSTM gates

The input gate selects new information to add to the cell based on the current input and previous hidden state and is calculated as:

$$i_t = \sigma(W_i * [h_{t-1}, x_t])$$

And last the output gate which selects information from the cell to combine with the current input and previous hidden state and it returns the output vector.

$$o_t = \sigma(W_o * [h_{t-1}, x_t])$$
$$h_t = o_t * \tanh c_t$$

All the parameters in the network are corrected through backpropagation using negative gradient descent.

$$\frac{\delta E}{\delta W} = \sum_{t=1}^{T} \frac{\delta E_t}{\delta W}$$

The construction of the model that we implemented can be seen in the figure below.

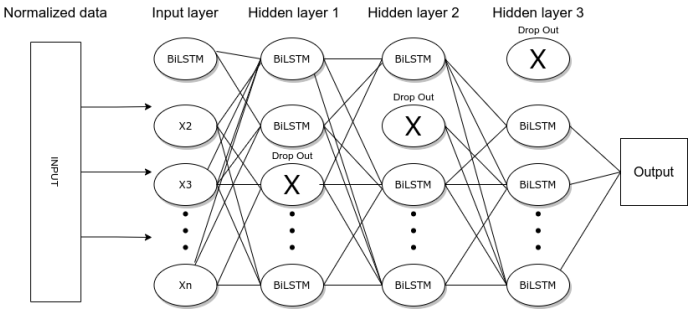

Fig. 4. BiLSTM Model

The model was implemented using the LSTM layer provided by Keras. [6] As we can notice from the diagram above, we used 4 Bidirectional hidden layers. In each hidden layer, we used 50 units in order to generate a complex function that could extract useful information about the data. This was an optimal number that resulted after some experimentations. In

the input layer, we used the nine features that we featured engineered. We added additional dropout layers, which set randomly (0.2) the inputs of some units. This technique was made to reduce the time required for training and also to prevent the overfitting problem that would lead to better generalization of the algorithm. In the end, we predicted the price of the bid and ask in test data and created a simple strategy to measure the profit made by our predictions.

VIII. Model Evaluation & Visualization of profit

A. Long Short-Term Memory(LSTM) model evaluation

For the evaluation of the model, we performed different experiments to measure the performance and then be able to choose the best model and strategy for better profit. Firstly, we evaluated the model for different time-windowed datasets. We aggregated the data into 1,5,20,30,60 minutes spans, and we measured the mean square error (MSE) between predicted and actual subsequent values in both training and testing. Below we represent the model performance in training for ten epochs.

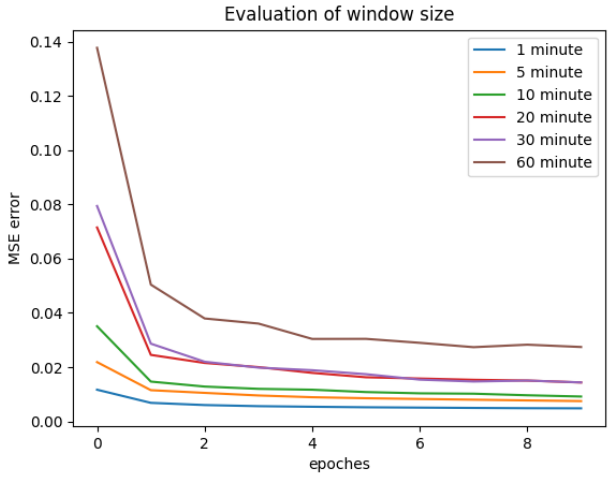

Fig. 5. Training Mean Square error in LSTM model

We can see clearly that the MSE in the 1-minute window has a minor error at around 0.004 compared with other methods. We compared the performance with a linear regression baseline, and we saw that the LSTM model outperformed the error rate(Baseline MSE error=3.1 vs LSTM=0.004). We also wanted to measure the error in the test data after the training was finished. As we can see in the plot below, the MSE testing error continues to be lower in the 1 minute. This happens because more data are used for training the model, thus making possible for the system to learn more complex function in the hidden layers.

Window size minutes	Test MSE error
1	**0.0097**
5	0.0165
10	0.0217
20	0.0237
30	0.0355
60	0.08675

We decided to choose the 1-minute data span, and we predicted the prices of bids and asks in test data. We generated a plot that shows the actual vs predicted prices for both asks and bids. As we can see, the predictions are very accurate, which means that the model was able to predict the fluctuation of prices in unseen data successfully.

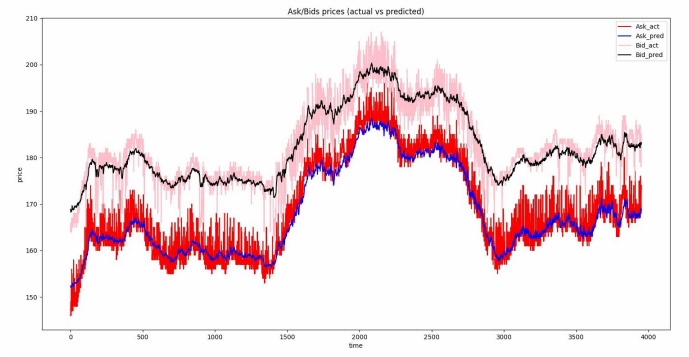

Fig. 6. Bid/Ask price predictions in test data

For the calculation of the profit, we created two simple trading strategy. The first strategy was constructed very simple in a greedy way. Because we saw that the price increased every hour, we decided to buy the price at the beginning of every 1 hour and sell it at the last minute of the hour. With this strategy, we achieve around 2000$ at the end of the test data by making around 150 trades.

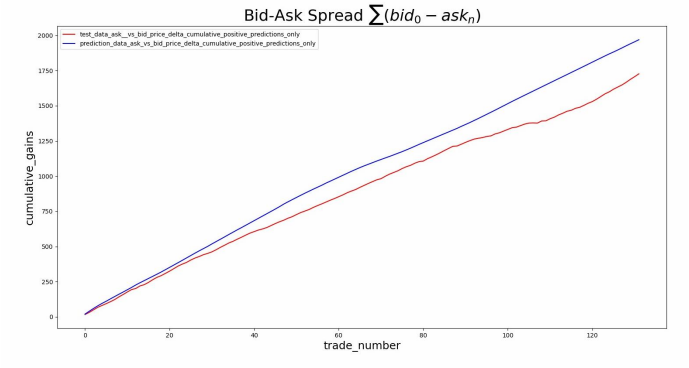

Fig. 7. Bid-Ask Spread (Window trading strategy)

The second strategy was a little more economical and strategic by using the mid-point price (bid+ask/2). We made use of two rolling window averages of price for short=10minute and long=1 hour for monitoring the trends. The algorithm was simple. We detected when the rolling averages windows matched and compared the price trends. If the price trend were down, then the model would give a buy signal and contrary when the trend was upward, the model would give a sell signal. At the end of the test data, the model would generate a profit of around 14K $ with an initial balance of 10k $.

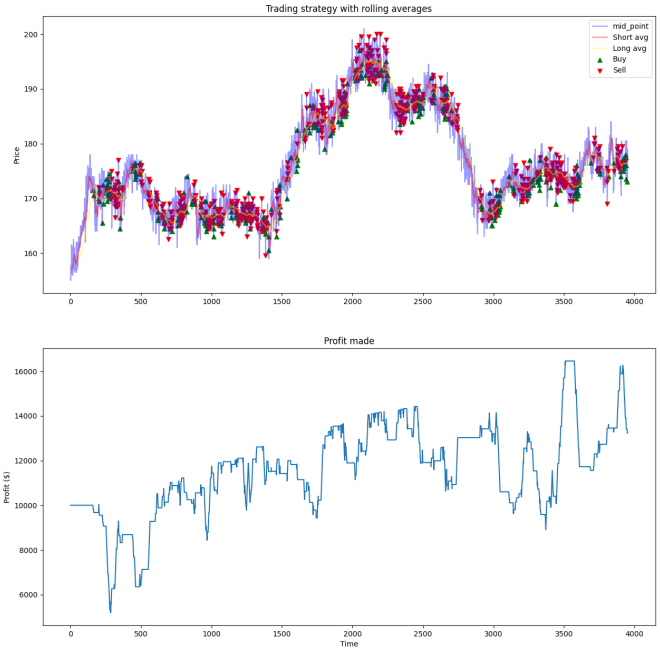

Fig. 8. Buy/Sell trading strategy with rolling averages)

Overall we can say that the second strategy using rolling average generated more profit at around 72% additional capital.

B. Q-Learning Evaluation

We also evaluate our DQN implementation over the same sampling frequencies and data. Buy and sell prices were set to be equal to the mean price of transaction, which was also the data fed to the DQN. This both aided in simplicity and represents a realistic model. In reality, transactions can only occur where bid and ask prices overlap. The size of this overlap region, called the "spread", is generally small, and narrowed further by broker and transaction fees, the large majority of asset exchanges occur very near the mean price.

For this implementation, certain values are selected for the variables involved, and new values are introduced specifically for training. These are as follows:

- $\gamma = 0.9 * (1 - \frac{\delta t}{2 \times \max(\delta t)})$ - This is fairly high, prioritising future gain over immediate, given the large number of datapoints at high sampling rates, scaled by the maximum timestep (minimum sample rate). This should ensure consistent weighting of immediate vs future profit for each sample rate
- $\alpha = 0.001$ - The learning rate, not previously discussed for the sake of brevity, is fixed at this fairly low value, again due to the large number of datapoints/states
- $\epsilon_n = k^n \epsilon_0$ - Here n represents the training epoch. ϵ is used as a "demonstration" factor. When the agent is asked to act, there is a probability ϵ that instead a random action will be taken. This acts to provide initial information to the agent about the results of actions at different states, and decays by a factor k after each epoch. The value of k is discussed below.
- $N_{\text{epochs}} = 10 \times \text{timestep}$ - The number of training epochs, scaled by the timestep to account for the disparity in data exposure at different sample rates.
- $k = 0.999 * \frac{\delta t}{\max(\delta t)}$ - The decay factor of ϵ. This gets smaller as the number of epochs decreases, so that exposure to the random choices from ϵ is equal for each sampling rate.

Presented in Figure 9 are the results from this testing. The performance metric used, defined in the figure caption, accounts for the nature of the stock price. Essentially it is meant to show the efficacy of the trading strategy, in a manner which the agent's profit might not, due to the consistently falling transaction price in our data. It is best interpreted qualitatively; > 0 and an agent is performing better than the stock, < 0 and it is performing worse.

The best performing agent is with a timestep of 10m, corresponding to a sampling rate of 6hr^{-1}. This is not a definite identification of the ideal sampling rate, due to the large number of hyperparameters that must be considered, in particular the weighting of future profits γ. To explore further we perform similar testing, with a fixed timestep of 10m and varying γ. (see Figure 10)

We find no trend between γ and final agent profit. The most profitable agent was at $\gamma = 0.6$, thought again this should not be taken as a definitive best value for γ, due to the stochastic nature of the algorithm and its neural network. Agents at all γ values show some trading strategy, characterised by unique increases/decreases in funds (see funds vs time plot in Figure 10), with significantly unprofitable agents consistently running out of funds early in the simulation. These unsuccesful agents can still be seen to sell at "determined" points, though they quickly buy back the assets they have just sold, and again return to zero funds.

The most successful agent at $\gamma = 0.6$ shows larger, more rapid changes in funds, and we analyse this agent's strategy further using Figure 11. This shows an interesting and somewhat intuitive pattern - the agent has learnt to buy stock at locally low prices, characterised by drops in its funds, and then sell at locally high prices. Also present are plateaus in its funds, appearing at time periods where prices are still dropping, perhaps demonstrating that it waits for the lowest price to buy more assets. At $t \simeq 1300$m it empties all of its funds into buying assets, running out of funds at $t \simeq 1400$m, then sells them all back as the asset price begins to increase. In short, the agent appears to have learnt the simple trading behaviour of buying the dip, selling the peak.

Worth noting is that the agents here are restricted to buying and selling a volume of one asset, whereas in reality bids and asks specify any quantity of assets to be traded. The agents were implemented this way in the interest of simplicity, as well as aiding training speed by reducing the range of possible actions. This may explain why the agent has periods of buying and selling every timestep, whereas the ideal could be to buy/sell a large quantity of assets at local maxima and minima

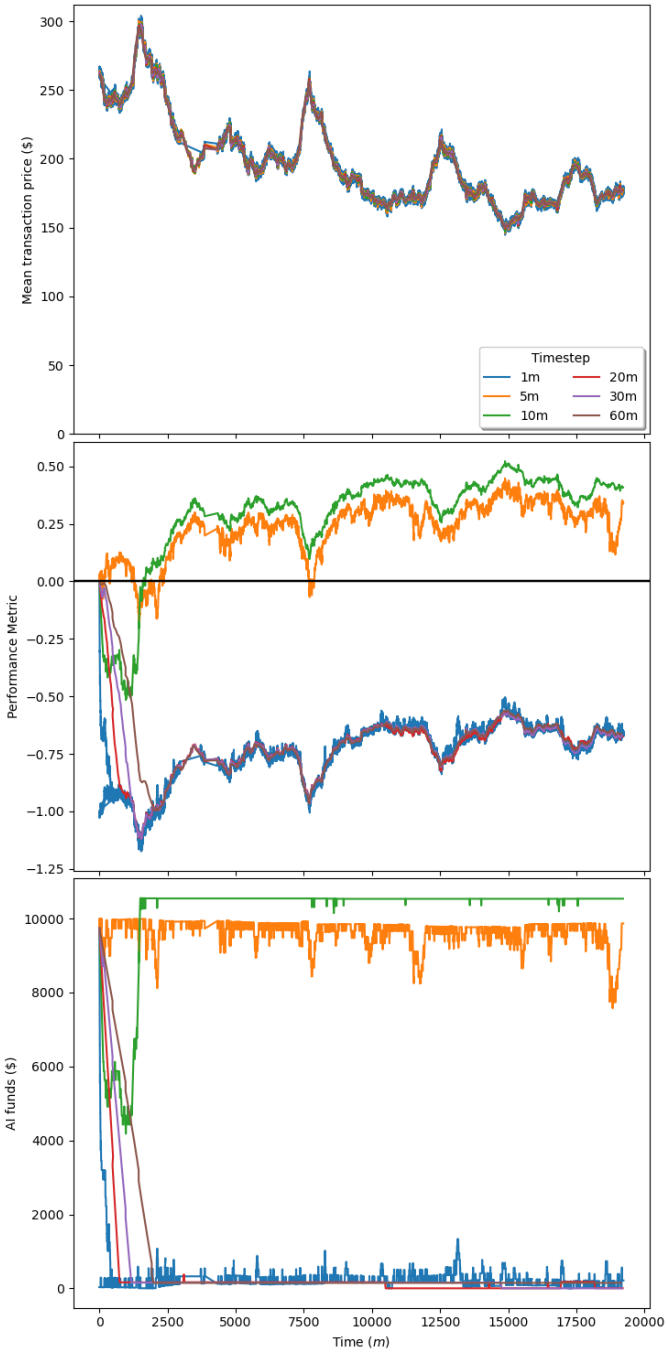

Fig. 9. Agent performance at the various sampling frequencies. Top to bottom: Mean transaction price vs time, Performance metric vs time (Performance metric = AI funds/Starting funds - Price/Initial price), AI funds vs time

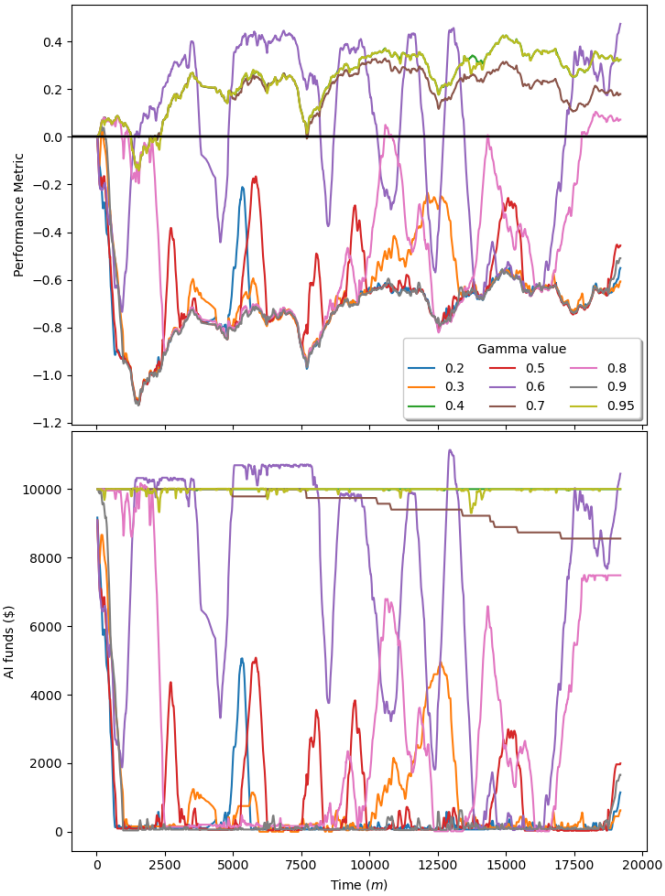

Fig. 10. Agent performance with varying γ. Top: Performance metric vs Time, Bottom: Agent funds vs Time

instead. For example, the agent here consistently sells assets as soon as the price begins to increase, to ensure that it has the opportunity to sell as many of its assets as possible before the price begins to fall again.

This proves to be a successful strategy. Even in this data, with a consistently falling stock price, the agent makes a profit over its initial funds multiple times, with a final profit of $540 on initial funds of $10000, representing a %5.4 increase. We are confident that with more favourable data the agent would make a considerable profit.

IX. Future Work

The financial market trading inquiry can extend into a wide range of further works in the future. Our study focuses mainly on looking for profits or losses in the future and making crucial trading decisions using either series prediction with a BiLSTM network or a DQN agent. However, there are also plenty of tasks that are essential for the system's better performance.
First of all, what we have considered as the next important step in our system is managing the dataset as real-time data. Processing our data in real-time by importing them in Spark Streaming as batch files can help our system to function as an online trading model. A combination between Apache Spark and Kinesis/Kafka can serve this duty. What we have done so far for this task is to use the AWS Kafka service (due to the AWS usage in our project) to construct a Data Stream for our data to collect them as streaming data, then to create Data Analytics for processing and cleaning the streaming data, and the implementation of another Data Stream that transfers the batch files to our Spark Streaming model.
Another notable thing that we have granted as future work

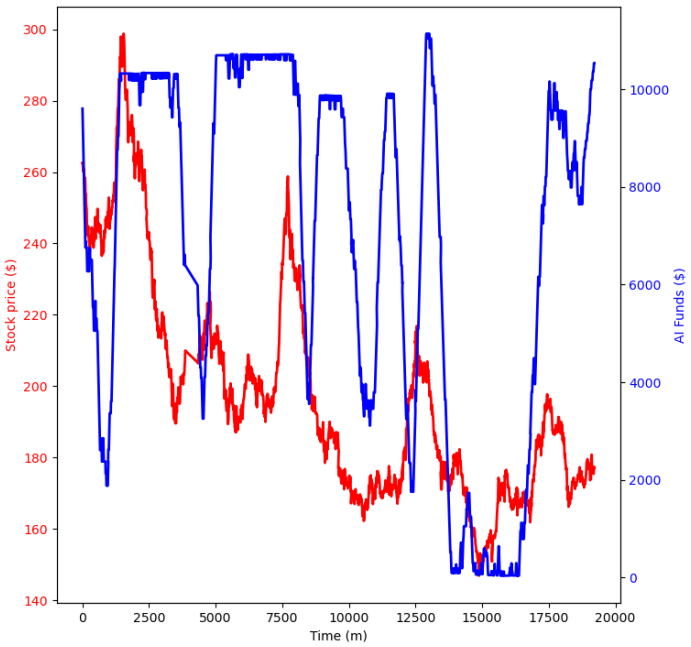

Fig. 11. Stock price and AI Funds vs Time for the best agent found, $\gamma = 0.6, \delta t = 10$m

is evaluating our model with real-trading data. As we have witnessed, the provided dataset that we have practiced in our model has been distributed in a probabilistic way which helps us to achieve high profits in our evaluation. However, the financial markets are not deterministic, comprised of noise and inconsistent trends that we need to consider.

Following the approach of Deep Reinforcement Learning, a next solid step is to continue investigating other Reinforcement Learning approaches, such as the A3C approach, the main ongoing research interest that HSBC is exploring for finding good trading strategies. Unfortunately, the limited time of the mini-project prevented us from following this suggestion. An interesting proposition is to combine our implemented approaches by extending the DQN network using BiLSTM layers. This would aid the agent's ability to predict future prices and hence give it more accurate predictions for Q-values, and given the demonstrated accuracy of our BiLSTM series prediction network, this would likely also prove successful.

X. Conclusion

This research paper aimed to identify effective machine learning algorithms that could implement a successful automated trading agent. We showed that gathering level 2 from the Limited Order Book had several challenges because of the large amount of data, and we demonstrated that it could be solved using spark technology. Furthermore, we noticed that to train our model successfully; the data should pass several preprocessing and cleaning steps that significantly reduced the size and errors. Finally, we compared two main implementations, the BiLSTM and Q-learning, and we showed how we could use different trading strategies from baseline to more sophisticated and generate higher profits.

We trained our BiLSTM model with around 80% of the data, and we let it predict the remaining 20%, which was around four days of data in the future. Our results revealed that the BiLSTM model was able to make accurate predictions with error rates of 0.009 in test data.

Overall both implementations were successful when using minimum ask and maximum bid prices. The BiLSTM predictor, with a previously defined trading strategy, achieved a 40% profit over the test set. The DQN agent rapidly learnt that, given minimum ask $<$ maximum bid, any successive buy and sell action gave a net gain, so produced a highly unrealistic profit of 1000%. Switching to using the mean price of transaction as both ask and bid price yielded more realistic results. The BiLSTM implentation made a small loss of 1%, but the Q-learning agent was more successful, with a profit of 5.4%.

Besides, it showed us that with the given testing data, the Reinforcement learning agents performed better as it was able to find the better opportunities for trade. We should also mention that we made as simple as possible the implementation by using only the price of the asset and not including the real-life challenges like commissions, delays in orders and quantities of the assets. It was let as part of future work, where we could combine both techniques (LSTM and Q-learning) that we implemented and introduce these challenges as part of training to make it more realistic.

References

[1] Zhichao Zou, Zihao Qu, "Using LSTM in Stock Prediction and Quantitative Trading", CS230:Deep Learning, 2020
[2] Horan Wei, Yuanbo Wang, Lidia Mangu, Keith Decker, "Model-based Reinforcement Learning for Predictions and Control for Limit Order Books",aaai,2019
[3] Yuqin Dai, Chris Wang, Iris Wang, Yilun Xu, "Reinforcement Learning for FX trading", Stanford University
[4] Ryo Akita, Akira Yoshihara, Takashi Matsubara, Kuniaki Uegara, "Deep learning for stock prediction using numerical and textual information", IEEE/ACIS, 2016
[5] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
[6] Chollet, F., & others. (2015). Keras. https://keras.io.
[7] João Carapuço, Rui Neves, and Nuno Horta,"Reinforcement learning applied to Forex trading," Applied Soft Computing, 2018, pp. 783–794.
[8] C. Watkins, King's College, "Learning from Delayed Rewards", PhD Thesis, 1989
[9] C. Watkins, P. Dayan, "Q-learning", Machine Learning, 1992, pp. 279-292
[10] Google Inc., V. Mnih, K. Kavukcuoglu, Patent for "Methods and Apparatus for Reinforcement Learning", US Patent, 2015, Pub. No. US 2015/0100530 A1