Supervised Deep Learning for Optimized Trade Execution

Hua Wanying, Long Zijie, Wang Kunzhen

April 20, 2019

1 Introduction

Optimized Trade Execution is one of the best-studied problems in the field of quantitative finance. In this problem, the goal is to buy(sell) a given number, V, of a specific stock within the given time horizon, H, with the minimum total cost(maximum total gain). If there is remaining inventory approaching the end of the time horizon, the agent will be forced to place a market order.

Note the equivalence of both sides (the buy and sell side) of the problem, we will therefore solve the problem only for the sell side. That is, the goal of our model would be to maximize the total selling cost of the V shares of a stock given the time horizon H. Moreover, we also note that the problem is additive with respect to different stocks. That is, a problem to buy V_1 shares of Stock A and sell V_2 shares of Stock B within a given time horizon H has a solution equivalent to the addition of optimal solutions given by solving the problem for Stock A and Stock B individually.

Our contribution to the study of the problem is to provide an alternative approach to addressing the problem, referencing the reinforcement learning model in [1]. The model we build is based on supervised deep learning. Implementation details, results as well as relevant justifications for the choices and assumptions made for building the model are also provided in this report.

2 Literature Review

3 Model

In this project, we assume that the optimal execution strategy can be expressed as a pure function of the following 6 variables: t the remaining time before the end of the time horizon, i the remaining inventory to sell, the price level, price trend, limit order book volume mismatch as well as the bid-ask spread at the decision point. Following the convention in [1], we group the 6 input variables into two categories, i.e., the **private variables** consisting of t and i that is specific to the Optimized Trade Execution problem, and the **market variables** consisting of the rest of the four. Output of the model is represented by ***action***, the price at which to place a limit order. The model can be expressed mathematically as

$$action = f(t, i, price\ level, price\ trend, vol\ mismatch, bid\text{-}ask\ spread),$$

where f is an unknown function to be learned.

To estimate the function f, we develop a supervised deep learning model (thereafter referred to as *the model*) as described below. The model is implemented with *Tensorflow* and *Tensorflow*

Keras provided by Google Brain, using *Python*. Implementation of the model can be found in the file *Model.py*.

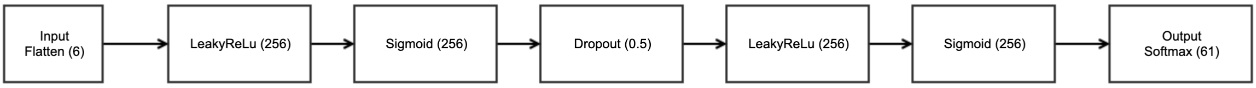

Figure 1: The Supervised Deep Learning Model

- **Input Layer** The input layer consists of simply the 6 parameters of the function f. Detailed definitions, rationales and extractions of these variables are provided in Section 4.2 and 4.3.
- **Hidden Layers** The model is composed of 5 fully-connected hidden layers with 256 neurons each. Activation functions for each layer is, correspondingly, *leakyReLu*, *sigmoid*, *dropout* with a rate of 0.5, *leakyReLu*, *sigmoid*. These activations are chosen after taking into consideration the nature of the problems. For example, noting the sparse activation characteristic of the *leakyReLu* activation and that the outputs are discrete, we chose *leakyReLu* to denoise the training process. Another advantage of the *leakyReLu* is its computational efficiency and ability to avoid dead neurons. The *sigmoid* activation is chosen for its ability to capture non-linear relationships. A *Dropout* layer is chosen in the middle to denoise and speed up the descent.
- **Output Layer** The output layer represents the predicted action given the input. The output variable, ***action***, is discrete for computational efficiency. Moreover, having a discretized output is important to avoid overfitting. Refer to Section 4.3 for details on how ***action*** is discretized.

4 Data Preparation

Describe and justify the value of V, H, T, I chosen for the research. Please refrain from saying that it's what the paper has chosen. We should have our own rationale.

4.1 Data Description

Describe the dataset and how we split it into training set vs testing set.

4.2 Market Variables

Describe the following:

- How market variables are extracted from the dataset. **Please include the exact formula**.
- Rational of why those market variables are chosen.
- Point to the exact python file for reference.

4.3 Private Variables

Describe the following:

- How private variables are extracted from the dataset. Please include details on **dynamic programming (esp. formula), execution simulation, variable discretization** and considerations.
- Point to the exact python file for reference.

5 Model Training

For model training, we've experimented with quite a few model constructions, from which we settled at the following configurations.

Firstly the loss function is determined to be the **sparse categorical crossentropy** loss provided by *Tensorflow Keras*. The loss function is one of the standard choices in multi-categorization models, measuring the categorical crossentropy.

For optimization algorithm, we choose the widely used **Adam Optimizer** [2]. It employs an adaptive learning rate and has a relatively efficient computational cost, making use of both the first and second moments of the gradients. Key update routine adopted by Adam is listed below.

$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1})\text{(Get gradients w.r.t. stochastic objective at time t)}\\
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t\text{(Update biased first moment estimate)}\\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2\text{(Update biased second moment estimate)}\\
\hat{m_t} &\leftarrow \frac{m_t}{1-\beta_1^t}\text{(Compute bias corrected first moment estimate)}\\
\hat{v_t} &\leftarrow \frac{v_t}{1-\beta_2^t}\text{(Compute bias corrected second moment estimate)}\\
\theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m_t}}{\sqrt{\hat{v_t}}+\epsilon}\text{(Update parameters)}
\end{aligned}
$$

The model is trained with the aforementioned configurations for 2000 iterations, at which point we note that the cost remains relatively stable and the accuracy stops improving. Therefore, we stop the training process at 2000 iterations.

6 Results

We implement two cateogiries of metrics to evaluate the model, namely,

1. **Accuracy**-based
2. **Total Strategy Execution Cost**-based

In this section, we define each of the two types of metrics and present their results correspondingly.

6.1 Accuracy

This is referred to as the accuracy metric, as defined by

$$accuracy = \frac{count(prediction == label)}{count(predictions)}.$$

Ultimately it measures how often the prediction matches the label provided, for both in-sample and out-of-sample data. Fortunately we have a default implementation provided by *Tensorflow Keras*.

We calculate the percentage in terms of decision points instead of training episodes. That is, assume that we have K training episodes in total with D decision points each and out of the $K \times D$ predictions we have P predictions hitting the optimal decision, the percentage calculated would be $\frac{P}{K \times D}$. On average, we are able to achieve 54% in-sample accuracy and 53% out-of-sample accuracy.

6.2 Total Strategy Execution Cost

Total Strategy Execution Cost is defined as the total cost if the client were to completely follow the model's suggested actions at each decition point. Assume that we are allowed to make decision every T time unit. That is, during the total time horizon of the problem definition, we are only allowed to make decisions at $t \in \{kT : 0 \leq k \leq \frac{H}{T}\}$. Here for simplicity, we assume that $\frac{H}{T}$ is whole. Let o_t represent the set of market variables at time $H - t$, $c_{im}(t, i, o_t)$ and $n_{im}(t, i, o_t)$ represents the immediate execution cost and immediate execution volume in time interval $[H - t, H - t + T)$ with remaining inventory i and market variable o_t at time $H - t$. Then the **Total Strategy Execution Cost** is defined recursively as

$$c(V, H) = c_{im}(H, V, o_H) + c(V - n_{im}(H, V, o_H), H - T).$$

Reusing the *ExecutionEngine.py* class, we can easily calculate $c_{im}(t, i, o_t)$ and $n_{im}(t, i, o_t)$ given t, i, o_t, thus $c(V, H)$.

There are two evaluation metrics based on the **Total Strategy Execution Cost**: percentage by which the model underperforms the optimal solution in terms of the cost and the percentage by which the model outperforms the mid-spread submit and leave strategy. Results for both metrics are presented below correspondingly.

6.2.1 Model VS Optimal

The optimal strategy is the strategy one can ever achive assuming all market information for the whole time horizon H is known at the beginning. That is, the market becomes deterministic. However, reality is that the market is stochastic, therefore the strategy one can achieve would be at most as good as the optimal strategy.

Out-of-sample results comparing the model with the optimal strategy are shown in Figure 2. Assuming the optimal cost is c_{opt} and the model cost is c_m, then the y-axis is expressed as

$$opt\% = \frac{c_m - c_{opt}}{c_{opt}}.$$

Recall that the problem is for selling, therefore $c_m \leq c_{opt}$ and hence $opt\% \leq 0$. On average, the model is able to achieve $\bar{opt}\% = -0.1179\%$ for out-of-sample data. The number shows that on average, we are only 0.1179% worse than the optimal strategy.

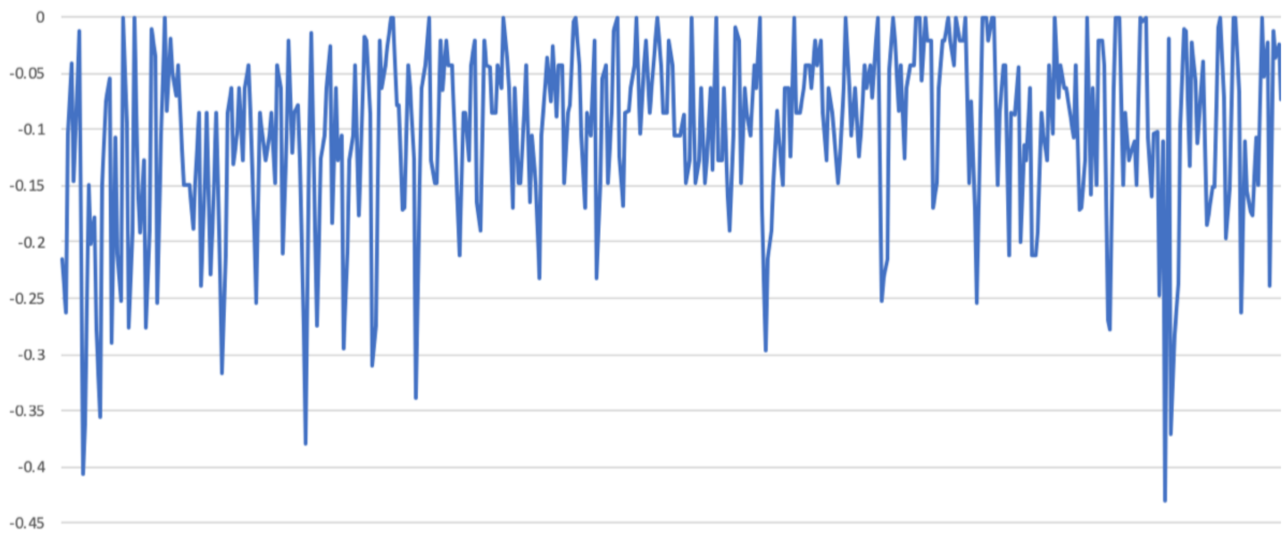

Figure 2: Percentage by which the model underperforms the optimal (%)

6.2.2 Model VS Mid-spread S&L

The mid-spread submit & leave strategy is one that at the begining of the time horizon H, we submit a limit order at the mid-spread price of the order book. At the end of the time horizon H, all remaining inventory are submitted as market order and assume to be executed at market price.

Out-of-sample results comparing the model to the Mid-spread submit & leave strategy are shown in Figure 3. Similar to the optimal case, assuming the mid-spread strategy cost is c_{mid}, then the y-axis is

$$mid\% = \frac{c_m - c_{mid}}{c_{mid}}.$$

On average, the model is able to achieve $mid\% = 0.54\%$.

Although this number seems small, it's actually quite significant, given that the mid-spread submit & leave strategy is on average only 0.7% worse than the optimal. Moreover, given our short time horizon H of 120 seconds and long decition interval of 30 seconds each, we are only given 4 decision points for each training episode. Therefore, if we were to follow the mid-spread price, it could be very hard to achieve the goal of selling all the V shares within the horizon. Indeed, from Figure 3 we can observe two extreme points at which our model is significantly worse than the mid-spread strategy. After tracing back the execution in the episode, we realize that in those two periods if we were to follow the mid-spread strategy, we wouldn't be able to sell all the V shares within H.

7 Remarks & Future Work

Despite the satisfying performance of the model, there are still plenty of room for future improvement. We have yet to implement the improvements due to time constraint of the project. However, we would like to note them down here as remarks.

- **Loss Function** For the current model training, the loss function is chosen as sparse categorical crossentropy to measure the categorical crossentropy. However, note that the

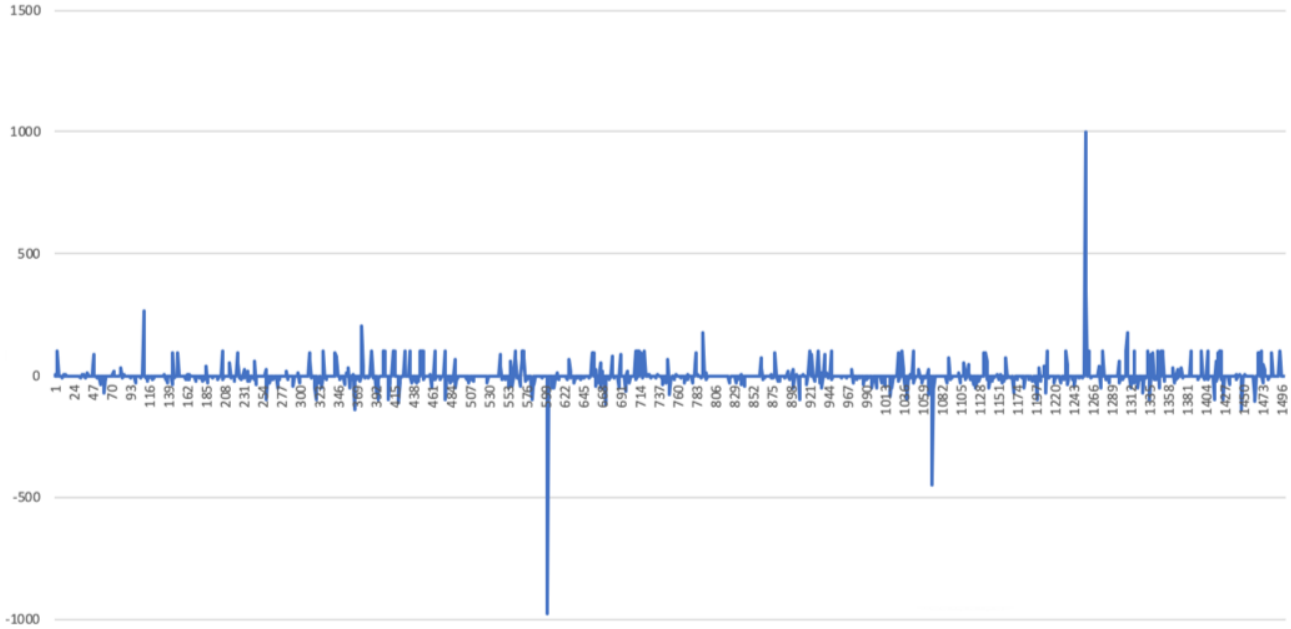

Figure 3: Percentage by which the model outperforms the mid-spread submit and leave (%)

label provided for the supervised training is the optimal strategy, having such categorical crossentropy losses might over-penalize the some of the predictions that could have done almost as good as the optimal solution in terms of total strategy execution cost, though having completely different decisions at each decision point. Therefore, a readily available alternative loss function would be the **total strategy execution cost** as defined in Section 6.

We have implemented the loss function as per described in Section 6, but have yet to integrate it to the training process because the implementation requires extra inputs than those provided in the function signature required by *Tensorflow Keras*. To actually integrate the loss function, further understanding of the *Tensorflow Keras* framework is required.

- **Model Inputs** Current choices of the model inputs, especially the market variables, are somewhat arbitrary. We believe that further analysis is necessary to justify that the market variables chosen are sufficient to predict the actions. For example, a principle component analysis (PCA) should be performed for selection of the market variables.

- **Market Impact** Throughout our research, we have been assuming that there is no market impact for the limit order we placed. That is, we placing a new limit order to the market will not change the subsequent order book status except for an extra message book entry. However, this is definitely not true, especially when the volume becomes significant compared to the market volume. A model with such market impact factored in [3] must be employed for real-life application.

8 Conclusion

In this project, inspired by [1], we develop a supervised deep learning model for the optimized trade execution problem. Sophisticated market simulation and dynamic programming algorithms are implemented for prepartion of the model input to make it computationally more efficient. Construction and training of the neural network is also fine-tuned with efficiency and accuracy in mind. With *Tensorflow* and *Tensorflow Keras*, we are able to build and train the model with limited effort, and indeed the results of the model are shown to be encouraging. Last but not least, we conclude the project with a few important remarks and after-thoughts. It has been a fruitful journey.

References

[1] Yuriy Nevmyvaka, Yi Feng, Michael Kearns. *Reinforcement Learning for Optimized Trade Execution*. Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006.

[2] Diederik P.Kingma, Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization*. ICLR, 2015.

[3] Robert Kissel, Morton Glantz. *Optimal Trading Strategies: Quantitative Approaches for Managing Market Impact and Trading Risk*. AMACOM, 2003.

[4] Almgren, R., Chriss, N.. *Optimal Execution of Portfolio Transactions*. Journal of Risk, 2002.

[5] Bertsimas, D., A, Lo, A.. *Optimal Control of Execution Costs*. Journal of Financial Markets 1, 1-50, 1998.