Supervised Deep Learning for Optimized Trade Execution

Hua Wanying, Long Zijie, Wang Kunzhen

April 19, 2019

1 Introduction

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*.

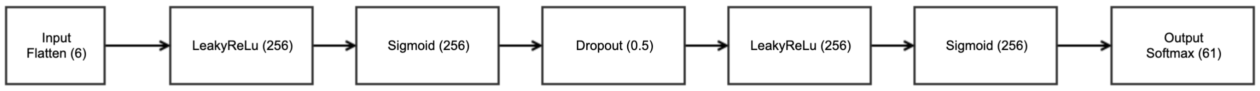

Figure 1: The Supervised Deep Learning Model

- **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

This section describes in depth the dataset our research bases on and how we extract the six aforementioned model inputs from the dataset.

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

To train the model, the loss function is first 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 employes 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.

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

For evaluation metrics, the ***accuracy*** metric provided by *Tensorflow Keras* is chosen:

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

Ultimately it measures how often the prediction matches the label provided.

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

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 metric is chosen as ***accuracy*** to measure how often the prediction matches the label.

8 Conclusion

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.