# Optiver: Trading at the Close submission

Bálint Varga (R5MUH0), Vilmos Balázs Varga (PKWSUX), József Velkey (VWWAQY)

***Abstract*— This paper explores our attempt at Optiver's "Trading at the Close" competition, where teams are challenged to develop deep learning models capable of predicting the closing price movements for Nasdaq listed stocks using the provided data. By combining classical mathematical tools like linear regression, and the advantages of modern artificial intelligence technology, we aim to develop a tool which can accurately forecast a stocks movement based on its performance during trading hours. The dataset consists of entries from the Nasdaq order and trading book, which we preprocess and shape to create an effective input for the training of our neural network. By capturing the patterns in the seemingly random data, our model can predict future values of stocks, which are then evaluate on the mean absolute error between the predicted return, and the observed target. Despite our models effective learning and prediction, we found that it is not accurate enough for real world applications.**

## I. Introduction

Our motivation to enter this competition was due to our goal to gain further understanding of time series prediction deep learning models. When choosing our assignment we found other interesting tasks, however we agreed that some were too challenging for us, while others we had no interest in, because we already had experiences with them in the past.

While we already had a basic grasp on the theory and methods of linear regression models, we were uninformed on stock trading, and we lacked the necessary understanding on stock analysis. Due to the absence of this familiarity, we had a difficult start, only made easier by the very helpful explanations of the values provided by both the hosts of the competition, and the other contestants.

Our goal was to acquire the provided data, process and shape it to make it suitable as an input for our model, then train said model to predict the target value. To get an understanding of our model's accuracy, we evaluated it using the mean absolute error (MAE) of the predicted values of the network, and the target values provided in the data. We used this because the goal of the competition is to get the lowest possible MAE value. Its formula is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - x_i\right|$$

Figure 1: Formula for mean absolute error

To achieve the best possible value, we sought to optimize the models hyperparameters after the first successful run with tools specifically developed for this purpose.

## II. Explaining the data

The dataset can be accessed by joining the competition on Kaggle. It contains 481 days' worth of data from the Nasdaq Stock Market, and each day contains information on 200 stocks. The value of these stocks is recorded in 55 steps, throughout the last 10 minutes of the day's trading session, and the goal is to predict their movements in the next 60 seconds.

Each trading day on the stock exchange ends with a closing auction, where orders are collected over a pre-determined timeframe, and then matched at a single price determined by the buy and sell demand expressed by auction participants. The provided data contains values both from the order book, and the auction book of the trading session.

The order book has buy (bid) and sell (ask) orders from traders at different prices, with the bid side wanting to buy a number of shares at a certain cost, while the ask side wants to sell some at their desired price. Closing price is the price at which the maximum number of shares can be matched, which is known as *matched size*. In continuous trading the transactions are immediate, for example if someone puts in a bid for 10 shares at 3 cost each, and later someone wants to sell 5 shares at 3, the reference price would be 3 and matched size would be 5, because 5 stocks were sold at the cost of 3.

The auction book behaves differently, because orders are not immediately matched, but collected through a certain amount of time and matched at the end. Both buyers and sellers put in their orders, and at the end of the auction the orders are matched. *Far price* is the hypothetical value where the most bids can be matched if the auction were to end at that moment, while *uncross price* is the actual value at the closing of the auction. *Imbalance* indicates the number of unmatched shares, if there are unsold shares the imbalance is in the ask direction, and if someone wants to buy but there aren't any the imbalance is in the bid direction. [1]

## III. Data Preparation:

Among the dataset, there were NaN values that presented challenges for subsequent training. Specifically, the features imbalance_size, reference_price, matched_size, ask_price, and wap each had 132 NaN values. This constitutes a very small fraction compared to the total number of data points, which amounts to 5,237,980, so we completely ignored these data points. The primary concern, however, lies with the far_price and near_price features, both exhibiting approximately 55% NaN values.

Various approaches could have been considered in

addressing this issue. One option was to omit these features entirely, resulting in a substantial loss.

Alternatively, these two features could have been excluded from the training process, focusing solely on the remaining features.

However, our preference was to utilize all available data points and features. Consequently, we opted to predict missing values for far_price and near_price based on the remaining data. It's worth noting that this decision may not be optimal, given that near and far prices typically manifest in the last 5 minutes, resulting in their absence (So, these values are not missing; they were intentionally omitted.). We decided to use XGBoost, which gave us fast results.

Furthermore, we made the decision to exclude the row_id from the training data, as it does not convey substantive information.

## IV. Training

The code uses TensorFlow and Keras to create a neural network model and performs hyperparameter tuning using the keras_tuner library's Hyperband algorithm.

**Model Architecture Definition (build_model function):**

The build_model function is defined to construct a neural network model.

The model consists of a flatten input layer, a variable number of dense (fully connected) hidden layers with ReLU activation, dropout layers for regularization, and a linear activation output layer (for regression).

The number of hidden layers (num_layers), the number of units in each hidden layer (units_i), and the dropout rate for each hidden layer (dropout_i) are hyperparameters to be tuned.

The optimizer choice (adam, sgd, rmsprop) is also a hyperparameter to be tuned.

**Early Stopping:**

Early stopping is employed using the EarlyStopping callback. It monitors the validation loss and stops training if there is no improvement after 5 epochs (patience=5). The best weights are restored (restore_best_weights=True).

**Hyperparameter Tuning:**

The Hyperband tuner is used for hyperparameter tuning. It performs a search over the hyperparameter space defined in the build_model function.

The search is conducted for a maximum of 10 epochs (max_epochs=10) with a factor of 3 for resource allocation.

The objective to optimize is the validation mean absolute error ('val_mean_absolute_error').

The results and configurations are stored in the 'tuning_directory' under the project name 'project'.

**Tuner Search:**

The tuner's search method is called with training data (X_train, y_train), validation data (X_val, y_val), and the early stopping callback.

The number of training epochs during the search is set to 2 (epochs=2), and the search is guided by the validation data.

In summary, this script defines a neural network model, uses the Hyperband algorithm to search for optimal hyperparameters, and performs the search over a limited number of epochs, utilizing early stopping for efficiency. The goal is to find the hyperparameter configuration that minimizes the validation mean absolute error.

## V. Decisions with the training

**Neural Network Architecture:**

Utilizing a neural network with multiple hidden layers enables the model to grasp deeper relationships within the data. Financial markets, renowned for their complex dynamics, pose a challenge that a deep neural network can address by learning intricate patterns inherent in order book and closing auction data. Traditional non-deep learning methods may struggle in the financial market domain (to be honest, deep neural networks don't consistently excel in real-life scenarios either, but for this particular task, it is good enough)

**Hyperparameter Tuning:**

The hyperparameter tuning is crucial because finding the right combination of hyperparameters is often crucial for achieving good model performance. The dynamics of the stock market, especially during the critical final ten minutes, might require fine-tuning of parameters such as the number of layers, units in each layer, dropout rates, and the choice of optimizer.

**Early Stopping:**

The stock market can be highly unpredictable, and early stopping helps prevent overfitting and ensures that the model generalizes well to new, unseen data. It also saves computational resources and time.

**Choice of Optimizers:**

The tuner explores different optimizers such as 'adam', 'sgd', and 'rmsprop'. The choice of optimizer can significantly impact how well the model learns from the data. Different optimizers have different strengths, and tuning this hyperparameter allows the model to adapt to the specific characteristics of the financial data.

**Use of Dropout Layers:**

Financial data can be noisy, and dropout helps prevent the model from becoming too sensitive to the noise in the training data.

## VI. Testing and evaluation

Testing primarily involves evaluating the trained model's performance on a separate test dataset that the model has not seen during training.
We divided our dataset into training, validation, and testing sets, maintaining an industry-standard approach to prevent data leakage and overfitting. The division was done with an 80-10-10 split, using **train_test_split** with 80% for training, 10% for validation during the hyperparameter tuning phase, and the remaining 10% reserved for testing.

So the model is trained on one set of data (**X_train**, **y_train**), the hyperparameters and stopping criteria are tuned on another (the validation set, **X_val**, **y_val**), and the final evaluation is done on a third, completely unseen set of data (**X_test**, **y_test**).

To gauge the predictive accuracy of our model, we employed the Mean Absolute Error (MAE). The MAE is a measure of how close the model's predictions are to the actual outcomes. If there's a pre-saved model (**best_model.h5**), the code checks whether the newly trained model performs better on the test set in terms of MAE. If the new model is better, it replaces the old one; otherwise, the old model is retained.

Our model's performance on the test set yielded an MAE of 6.396504988239148, implying that the predictions were, on average, approximately 6.4.

The presence of missing values in key features like **far_price** and **near_price** posed a significant challenge. We addressed this by predicting these missing values using XGBoost.

## VII. Summary

This paper outlines efforts in Optiver's stock prediction competition, using deep learning. The dataset comprises Nasdaq order/trading book entries, with a focus on predicting closing price movements. Challenges include NaN values, hyperparameter tuning with TensorFlow/Keras. The neural network architecture incorporates dropout layers and early stopping.

## VIII. REFERENCE(S)

[1] Optiver, Trading at the Close Introduction, kaggle.com, 2023.12.12
https://www.kaggle.com/code/tomforbes/optiver-trading-at-the-close-introduction