MTH767: Neural Networks and Deep Learning
2023/24

Queen Mary University of London

School of Mathematical Sciences

Mini project report
Trading At The Close

Daniil Vorontsov (ID: 230807431)

Examiner: M. Hanada

Contents

1 Introduction and problem statement

The present project is dedicated to working with dataset from **Trading At The Close** competition, hosted by Optiver on Kaggle in 2023[1]. The competition was concerned with development of a model capable of predicting short term movements during the 10 minute closing auction period.

1.1 Trading At The Close

Each trading day on the Nasdaq Stock Exchange concludes with the **Nasdaq Closing Cross auction**, which sets the official closing prices for listed securities. These prices are crucial indicators for investors, analysts, and other market participants in assessing the performance of individual securities and the overall market. Nearly 10% of Nasdaq's average daily volume occurs during the closing auction. This auction provides accurate price and size discovery, establishing benchmark prices for index funds and other investment strategies.

In the final ten minutes of the trading session, market makers such as Optiver integrate traditional order book trading with price auction data. This ability to consolidate information from both sources is essential for offering the best prices to all market participants.

1.2 Dataset overview and problem statement

In this competition, target is short term price movements during the 10 minute auction period. All the training data is contained within a single csv file[2].

- **stock_id** - A unique identifier for the stock. Not all stock IDs exist in every time bucket.
- **date_id** - A unique identifier for the date.
- **imbalance_size** - The amount unmatched at the current reference price (in USD).
- **imbalance_buy_sell_flag** - An indicator reflecting the direction of auction imbalance (buy-side imbalance: 1, sell-side imbalance: -1, no imbalance: 0).
- **reference_price** - The price at which paired shares are maximized, the imbalance is minimized and the distance from the bid-ask midpoint is minimized, in that order. Can also be thought of as being equal to the near price bounded between the best bid and ask price.
- **matched_size** - The amount that can be matched at the current reference price (in USD).
- **far_price** - The crossing price that will maximize the number of shares matched based on auction interest only. This calculation excludes continuous market orders.
- **near_price** - The crossing price that will maximize the number of shares matched based auction and continuous market orders.
- **[bid/ask]_price** - Price of the most competitive buy/sell level in the non-auction book.
- **[bid/ask]_size** - The dollar notional amount on the most competitive buy/sell level in the non-auction book.
- **WAP** - The weighted average price in the non-auction book.

$$\frac{BidPrice * AskSize + AskPrice * BidSize}{BidSize + AskSize}$$

- **seconds_in_bucket** - The number of seconds elapsed since the beginning of the day's closing auction, always starting from 0.
- **target** - The 60 second future move in the wap of the stock, less the 60 second future move of the synthetic index.

The synthetic index is a custom weighted index of Nasdaq-listed stocks constructed by Optiver for this competition. The unit of the target is basis points, which is a common unit of measurement in financial markets. 1 basis point price move is equivalent to a 0.01% price move. Formula for **target**:

$$Target_t = (\frac{StockWAP_{t+60}}{StockWAP_t} - \frac{IndexWAP_{t+60}}{IndexWAP_t}) * 10000$$

Where t is the time at the current observation. All size related columns are in USD terms. All price related columns are converted to a price move relative to the stock **WAP** (weighted average price) at the beginning of the auction period.

1.3 Architecture

The main building block for the model in this project will be Long Short-Term Memory unit (**LSTM**)[3]. LSTM is a type of recurrent neural network (RNN) that expects the input in the form of a sequence of features. The ability to memorize long term and short term data patterns makes LSTM a powerful tool for time series predictions.

Initial approach included adding convolutional layer (CNN) to LSTM[4]. CNNs are powerful for learning local patterns in data, while LSTMs are effective at capturing long-term dependencies in sequential data. Combination of these two types of networks is widely used in video recognition tasks and can be used to improve time series predictions. Output of the CNN can be used as the input to the LSTM which allows the LSTM to learn features from the input data that have been learned by the CNN.

Despite outlined advantages of adding CNN to network, training process proved to be highly complex and typically resulted in overfitting model. This prompted me to pursue less complex models, trained on sole target with lags as it's features.

Second architecture used as improvement was a bidirectional LSTM (**BLSTM**)[5]. BLSTM combines two LSTMs: one processes the input sequence in the forward direction (from the first token to the last), and the other processes it in the backward direction (from the last token to the first). This allows the network to have information from both the past and the future when making predictions.

Another tool to improve performance of the model is to create encoder-decoder (autoencoder) based on LSTM[6]. The LSTM network can be organized into an architecture called the Encoder-Decoder LSTM that allows the model to be used to both support variable length input sequences and to predict or output variable length output sequences. This architecture is the basis for many advances in complex sequence prediction problems such as speech recognition and text translation.

Overall, this project is be targeted at following architectures:

- LSTM
- BLSTM
- LSTM autoencoder
- BLSTM autoencoder

1.4 LSTM unit

A vanilla LSTM unit has following structure:

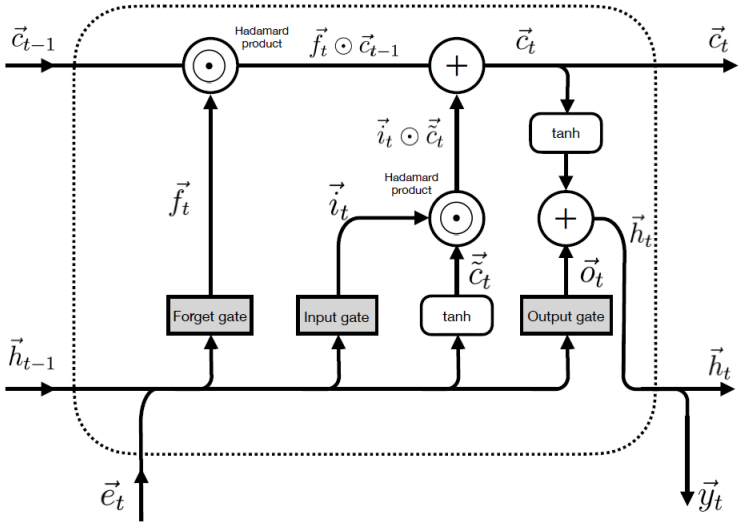


Figure 1: Long Short-Term Memory architecture

- The **input gate** $\vec{i}_t$ decides how much of the new input x_t and the previous hidden state $\vec{h}_{t-1}$ should influence the cell state.
- The **forget gate** $\vec{f}_t$ determines how much of the previous cell state $\vec{c}_{t-1}$ should be kept.
- The **cell gate** g_t creates new candidate values $\vec{\tilde{c}}_t$ for the cell state.
- The cell state $\vec{c}_t$ is updated by blending the retained cell state and the new candidate values.
- The **output gate** $\vec{o}_t$ controls the output of the cell state to the hidden state.
- The hidden state $\vec{h}_t$ is updated to reflect the current cell state modulated by the output gate.

$$
\begin{aligned}
\vec{f}_t &= \sigma(W_f \cdot [\vec{h}_{t-1}, \vec{e}_t] + \vec{b}_f) \\
\vec{i}_t &= \sigma(W_i \cdot [\vec{h}_{t-1}, \vec{e}_t] + \vec{b}_i) \\
\vec{\tilde{c}}_t &= \tanh(W_c \cdot [\vec{h}_{t-1}, \vec{e}_t] + \vec{b}_c) \\
\vec{c}_t &= f_t \odot \vec{c}_{t-1} + \vec{i}_t \odot \tilde{C}_t \\
\vec{o}_t &= \sigma(W_o \cdot [\vec{h}_{t-1}, \vec{e}_t] + \vec{b}_o) \\
\vec{h}_t &= \vec{o}_t \odot \tanh(\vec{c}_t)
\end{aligned}
$$

Here σ is the sigmoid function, and $\odot$ is the Hadamard product.

2 Training

Mean Absolute Error (MAE) was used as loss function in this project:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i and $\hat{y}_i$ are actual and predicted values correspondingly.

For regular LSTM two structures were tested:

- 25 stacked vanilla LSTM units
- two groups of LSTM layers (first Layer learning temporal dependencies in the input sequence and outputing a sequence and a second LSTM Layer which further process the sequence and outputs a single vector capturing the sequence's temporal dependencies.

2.1 Training strategy

As mentioned above, only target values were used for training and validation. The target values were extracted from original dataset, grouped by stocks (**N_STOCKS**=200) and lagged by given number (**N_LAGS**=20). In the notebook this is done with **windowed_dataset** and **build_features** functions.

The function **build_features** takes a DataFrame containing time series data for various stocks and transforms it into a pivoted format suitable for time series analysis and modeling. The function ensures comprehensive coverage by including all combinations of stock IDs, date IDs, and time intervals, and then restructures the data into a pivot table format.

Each model begins with an input layer with the shape=(**N_LAGS**, **N_STOCKS**). The output is a vector of predicted future movements of **N_STOCKS**.

Windowed_dataset function takes a time series data, converts it into a TensorFlow Dataset, and transforms it into overlapping windows. Each window contains a sequence of data points for the model to learn from (features) and the next data point as the target (label). The dataset is then batched and prefetched for efficient training.

Models were trained in Google Colab, accelerated by GPU using mixed precision strategy provided by Keras[7]. First 390 days were used for training and days 390-480 were used for validation. Adam optimiser provided by Tensorflow framework was used for optimal training as well as early stopping technique to prevent overfitting of a model.

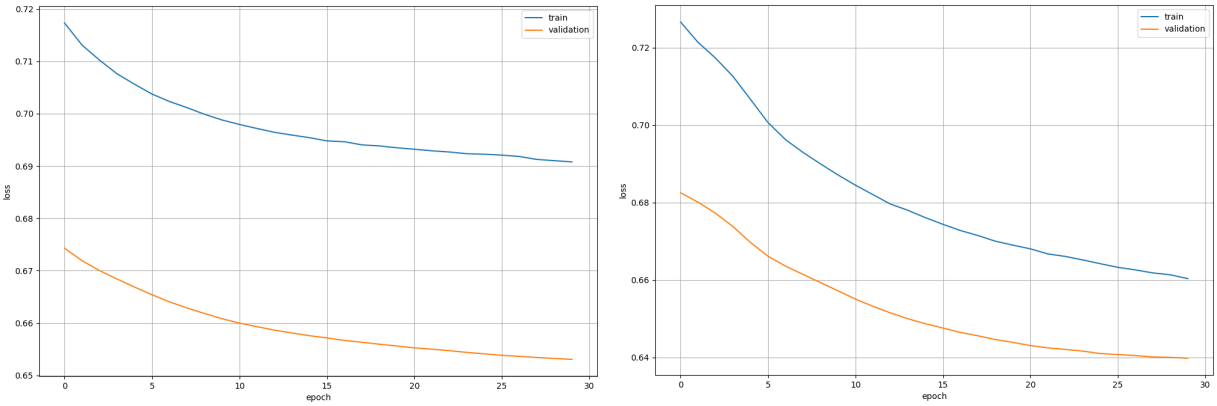
The idea and preprocessing functions (build_features and windowed_dataset) were taken from published solution[8] for competition. Original notebook included only LSTM AutoEncoder model. The contribution consisted of extending the list of models to compare performance among different architectures based on LSTM unit.

3 Results

The number of epochs was set to 30 as higher values resulted in disconnection from GPUs because of Colab restrictions on running time.

Following section provides plots for training and validation losses per batch for all models listed previously.

3.1 LSTM



(a) Regular LSTM (b) LSTM with temporal dependencies

Figure 2: LSTM training results

Evident from the figure, layer with sequence output followed by regular LSTM outperformed staked vanilla LSTM units. The curve of former one loss is steeply decreasing and concludes in the lower values than the latter one.

3.2 Bidirectional LSTM

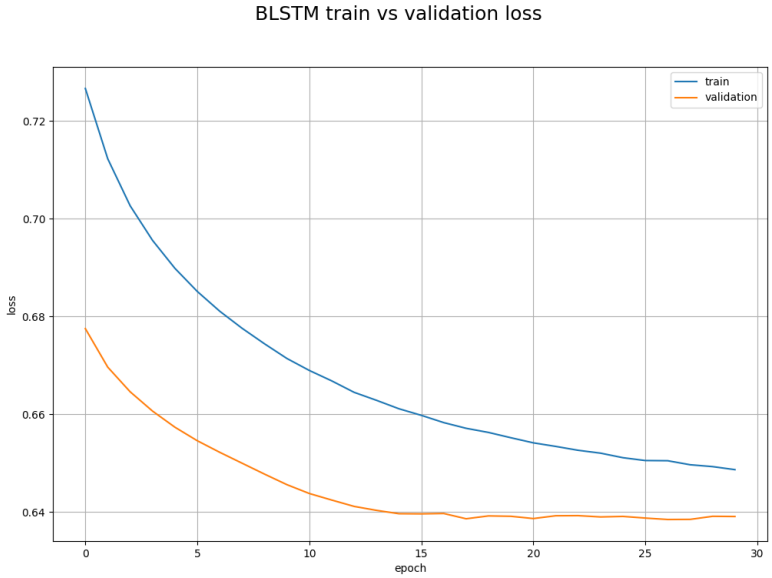


Figure 3: BLSTM training results

Bidirectional architecture provided even better results than unidirectional models. Noticeably, ratio of convergence of training and validation losses is better than on the previous plots. One can conclude that structure provides best result among less complex structures.

3.3 LSTM with autoencoder

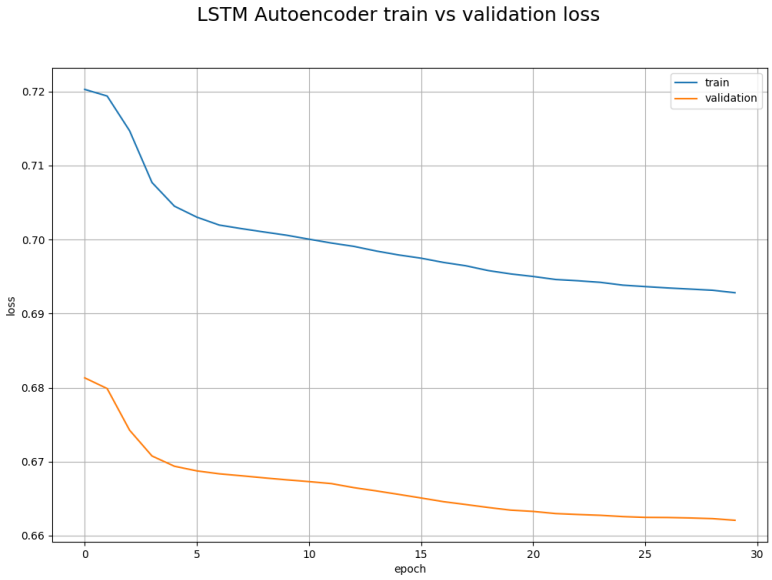


Figure 4: LSTM autoencoder training results

LSTM with autoencoder differs from previous models by sharp fall during first 5 epochs and slow decrease afterwards. This behaviour can be explained by nature of autoencoding training. The model learns to reconstruct the input sequences and because the network starts with random weights initially, there's a lot of room for improvement.

3.4 BLSTM with autoencoder

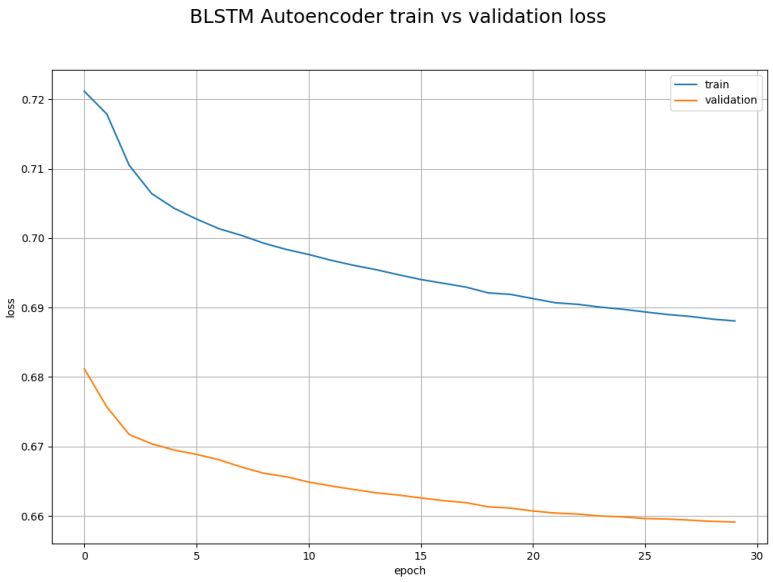


Figure 5: BLSTM autoencoder training results

As we can see from the figure, BLTSM autencoder exhibits similar behaviour to LTSM autencoder. The difference is the slope of curve after the fall is steeper. Both the train and validation learning curves reach better values than before.

4 Conclusions

Present project explored efficiency of different LSTM architectures for predicting short range price movements. In total 5 models were trained and validated on a dataset of more than 5 million values for high frequent financial time series.

The comparative analysis of training and validation losses revealed insights into the effectiveness of different configurations. Notably, the model incorporating a layer with sequence output followed by regular LSTM units demonstrated superior performance compared to stacked vanilla LSTM units. This observation suggests the importance of carefully designing the model architecture to achieve optimal results.

Furthermore, the bidirectional LSTM architecture exhibited even better results, indicating its effectiveness in capturing temporal dependencies in high frequency time series. The improved convergence of training to validation losses signifies the robustness of this architecture compared to simpler structures.

The introduction of LSTM with autoencoder demonstrated a distinct training behavior characterized by a sharp decline in loss during the initial epochs, followed by a slower decrease. This behavior aligns with the nature of autoencoding training, where the model learns to reconstruct input sequences.

The comparison between LSTM and BLSTM autoencoders highlighted similar behaviors, with BLSTM autoencoder exhibiting a steeper slope in loss reduction post-initial decline. Due to computational constraints, limiting the number of epochs to 30, autoencoders did not show better performance relative to other models. Generally, loss should generally decrease over time, and in the long run attention mechanism eventually outperform regular architectures.

In summary, the results underscore the importance of architectural choices in LSTM models for stock price prediction. Future research could explore additional modifications of RNN such as Gated recurrent unit (GRU)[9] or other architectures based on pure attention mechanism such as transformers[10].

5 References

[1] `https://www.kaggle.com/competitions/optiver-trading-at-the-close`.

[2] `https://www.kaggle.com/competitions/optiver-trading-at-the-close/data`.

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

[4] Jeffrey Donahue et al. "Long-term recurrent convolutional networks for visual recognition and description". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2625–2634.

[5] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[6] Pankaj Malhotra et al. "LSTM-based encoder-decoder for multi-sensor anomaly detection". In: *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM. 2016, pp. 107–115.

[7] `https://keras.io/api/mixed_precision/`.

[8] `https://www.kaggle.com/code/jorgearreserodriguez/lstm-with-simplified-feature-engineering`.

[9] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[10] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.