Realized Volatility Prediction

A concluded kaggle competition

Team members: Martin Molina, Yuan Zhang

Deep Learning Bootcamp

The Erdos Institute, Summer 2025

The order book

A real time list of buy and sell orders for a stock, organized by price level. Shows the available volume to buy or sell at each price and the balance of supply and demand.

The WAP (weighted average price) is calculated with the best bid and ask data. Then the RV (Realized Volatility) is the sum of all the squares of log return of the consecutive of WAP’s.

Volatility is associated with how much the stock prices fluctuate.

Goal:

Predict next 10-minute realized volatility from the current 10 minutes of book order data and trade data.

The Dataset

- ~30000 level 2 book order book data, each identified by a stock ID and a time ID. The pairwise combination of stock and time ID is the row ID.
- Each time series represents one 10-minute window for a specific stock
- Sampling times within each window (“seconds in bucket”) are irregular
- 112 different stocks in total
- ~ 3 million order book records in total
- Trade data

ETL pipeline

ETL pipeline: Data Extraction and Transforming

- Retrieving raw data from kaggle.
- Recovering time id order (credit of this part goes to this kaggle forum and the top solution).
- Creating timeseries data (by separating each time id into smaller intervals), and other data (e.g. current RV) from raw book data.
- Creating trade data from raw trade data.

ETL pipeline: Data Loading

- Creating the train and test split by making the latest 10 percent of time id as test set.
- Using the custom RVdataset pytorch Dataset subclass to initialize a pytorch Datasets for pytorch dataloader.
- Loading the values into the custom training loop reg_training_loop_rmspe by passing the train and test dataloaders as parameters.

Neural Networks

Custom layers

- Frozen convolution layer to create derivative like time series features.
- Positional embedding layer using cross-attention.
- Encoder based on self attention.
- Decoder using self and cross-attention with options for masking and shifting the ground-truth target.

Remark: The attention based layers structure is built according to the legendary paper *Attention is all you need*.

Proposed models

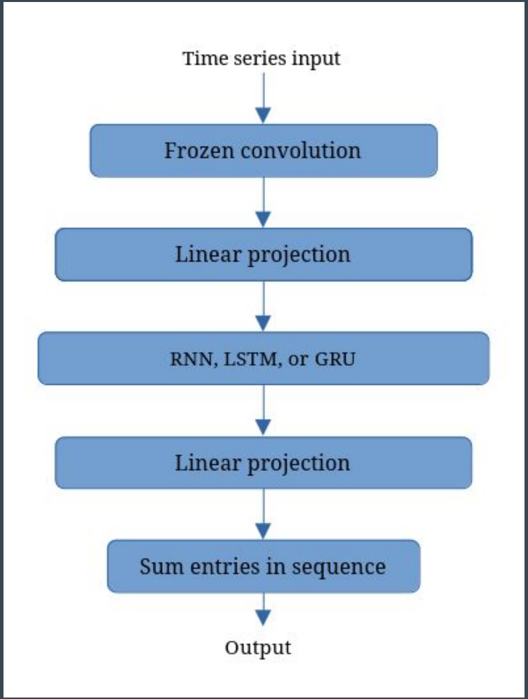
Timeseries based models

- RNN (Recurrent Neural network) based mods (rnn, lstm, gru)
- Encoder only transformer
- Encoder-decoder teacher forcing transformer model

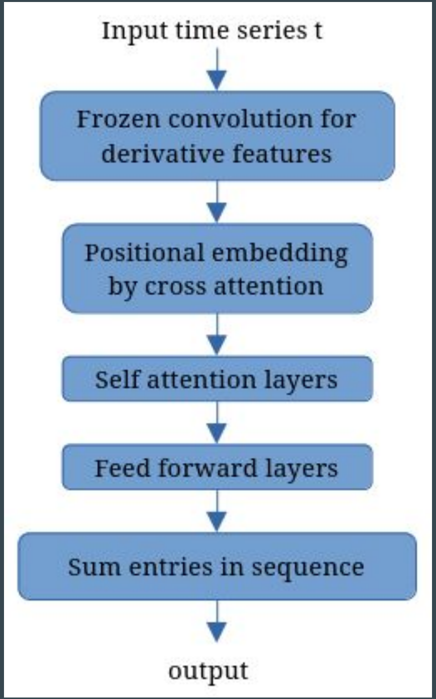
Adjustment model

- Produce adjustment value according to tabular parameters, and categorical embeddings. Then combine the adjustment value with the output of a base model (trained as a submodel).

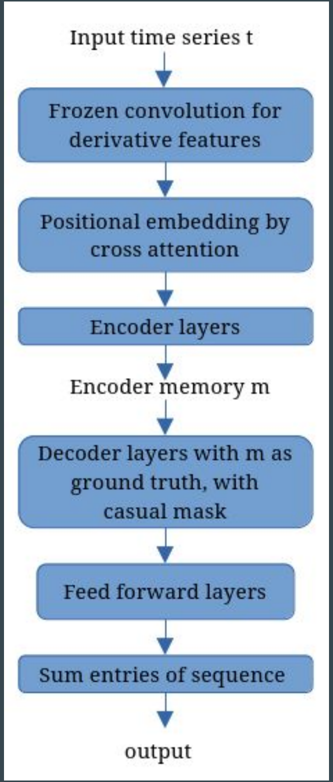
Time series models



RNN bases model: With best loss (rnn: 0.2348, lstm: 0.2301, gru: 0.2308)

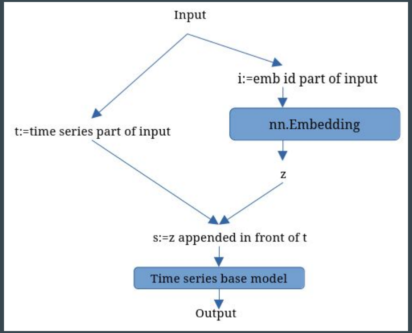


Encoder only transformer: With a best loss 0.2332

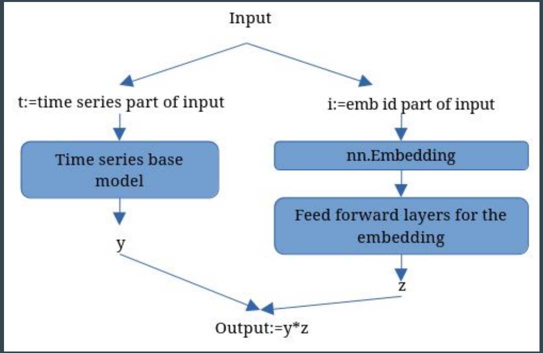

Input time series t
Frozen convolution for derivative features
Positional embedding by cross attention
Encoder layers
Encoder memory m
Decoder layers with m as ground truth, with casual mask
Feed forward layers
Sum entries of sequence
output

Encoder and casual masked decoder teacher forcing transformer: With a best loss 0.2296

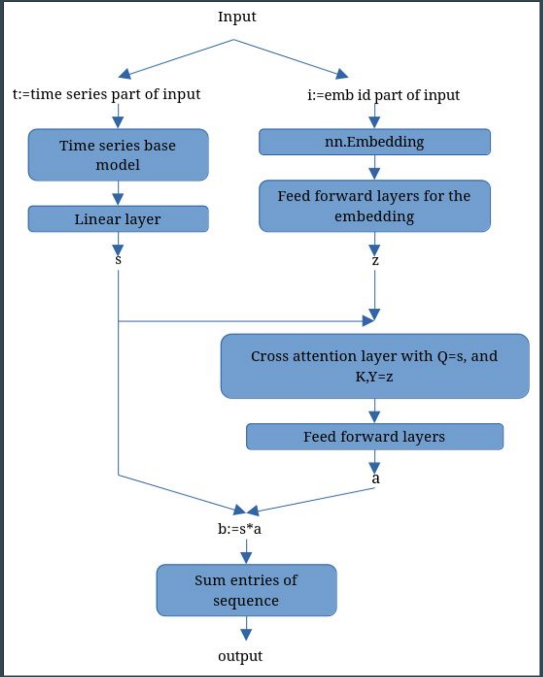
Adjustment models



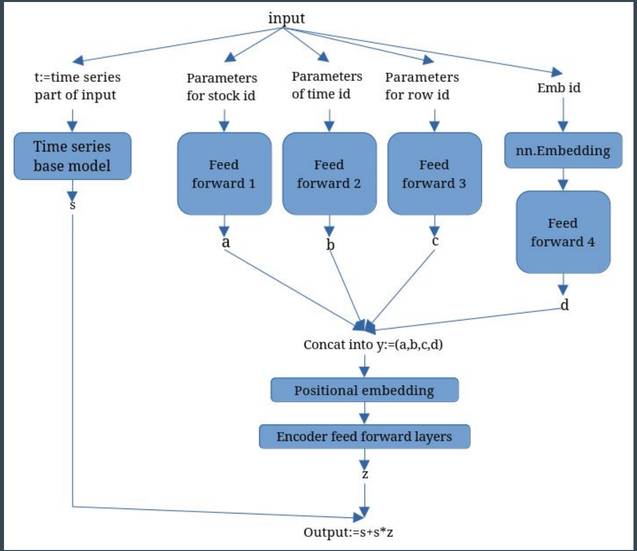
Pre-append model: Best loss 0.2233 with base model GRU or LSTM



Multiplication adjustment model: Best loss 0.2228 with base model GRU



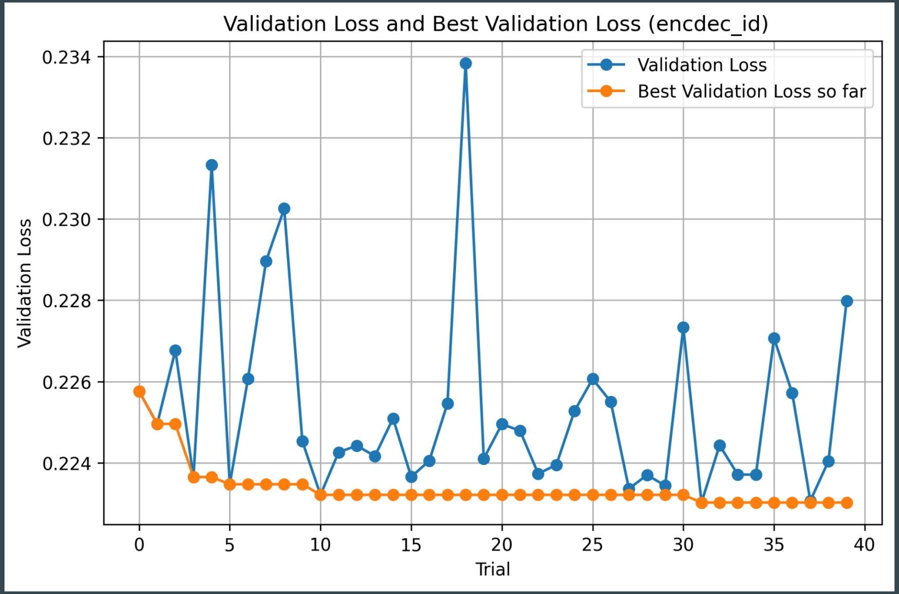
Cross attention adjustment model: Best loss 0.2249 with base model GRU



Combined adjustment model: Best loss with 0.2153 with base model GRU

Fine Tuning

Fine-tuned two encoder models (with and without ID embedding) using Optuna with carefully selected hyperparameter ranges and early stopping for underperforming trials.



Take away

RNN-based

Pro: Light weight with pretty good performance - Best base model lstm with loss **0.2301.**

Con: Not as modular as Transformers, so might not be as easy to combine them and create improved models.

Transformers

Pro: Best base model - The teacher forcing model is the best base model with loss **0.2296** . Transformers are very modular, so easy to play around with.

Con: Heavy to train. The improvement might not be worth the cost.

Adjustment models

Pro: It greatly improved our models. The best model has loss 0.2153.

Further work is needed, as too many parameters might cause noise and induce bad regularization.

Thank you!