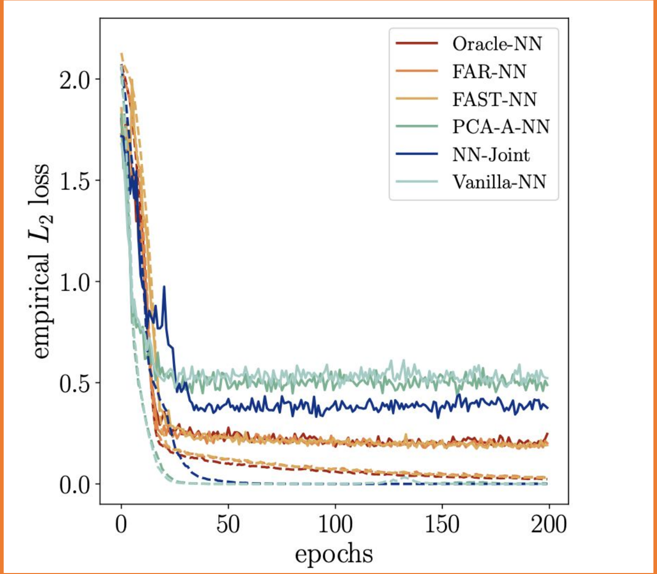
Group 10 STA 4365 Final Presentation

Andres Machado, Erick Rodriguez, Franco Vidal, Simon Hernandez

Table of Contents



Preview of the FAST Transformer Model

Optiver - Trading at the close

- Kaggle competition in which competitors were challenged with developing models to predict the closing price of 200 stocks from the NASDAQ, specifically within the last 10 minutes of the trading day.
- The target variable is defined by the following expression:

$$Target = \left(\frac{StockWAP_{t+60}}{StockWAP_{t}} - \frac{IndexWAP_{t+60}}{IndexWAP_{t}} \right) \times 10000$$

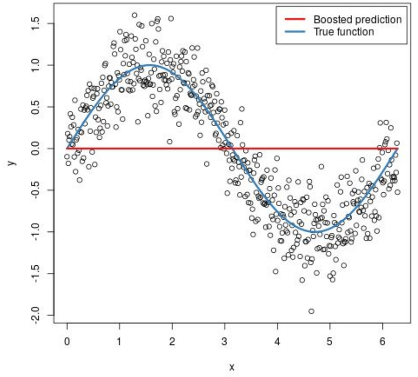
- How much better or worse does the stock perform compared to the overall market index in the next 60 seconds (scaled)?

Features

- Stock_id
  - Identifier for the stock
- Date_id
  - Identifier for the date
- Imbalance_size
  - Amount unmatched at current price
- Imbalance_buy_sell_flag
  - Reflects the direction of auction imbalance
- Reference_price
  - Price at which paired shares are maximized
- Matched_size
  - Amount that can be matched at current reference price
- Far_price
  - Price that will maximize number of shares matched based on auction interest
- Near_price
  - Crossing price that will maximize number of shares based on auction and continuous market orders
- Ask_price
  - Price of the most competitive buy/sell level in the non-auction book
- Ask_size
  - Dollar notional amount on the most competitive buy/sell level in the non-auction book
- WAP
  - Weighted average price
- Seconds_in_bucket
  - Number of seconds elapsed since the beginning of the days closing auction

GIF Source: https://uc-r.github.io/gbm_regression

XGBoost: Intro



Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors.

Gradient Boosting is the framework XGBoost is built on. Gradient boosting learners train themselves on residual errors of the previous predictors. It aims to optimize a loss function (that is differentiable) with gradient descent, that's where the "gradient" in gradient boosting comes into play.

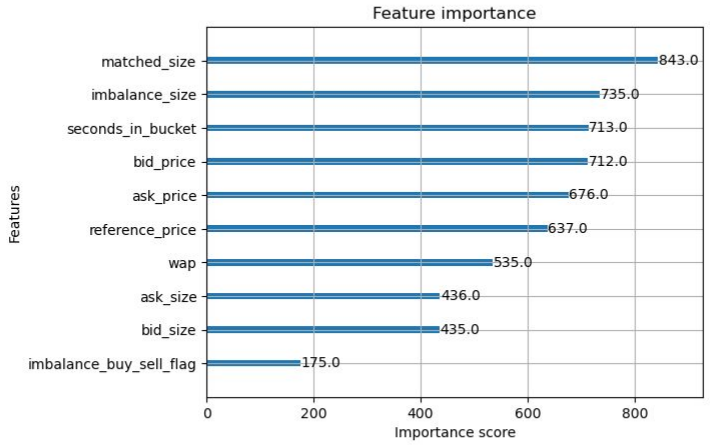
XGBoost (short for Extreme Gradient Boosting) is an advanced implementation of the ever so popular gradient boosting algorithm. It has become infamous in ML competitions and tasks all over the world because of its great performance.

Advantages over traditional Gradient Boosting: Hyperparameters, parallel processing, handling missing data, and much more.

XGBoost: Base Fitting

Baseline model:

- Using **XGBRegressor** function with default parameters.
- 80/20 Train-Test split.
- 10 standardized features.
- MAE: 6.2840



XGBoost Bonus: PCA Analysis

Out of curiosity, we decided to fit an XGBoost model using PCA. So, we transformed our design matrix (X) into a matrix with **3** principal components. We added up the explained variance ratio for all three PC's and they explained 69.2615% of the total variance of the data.

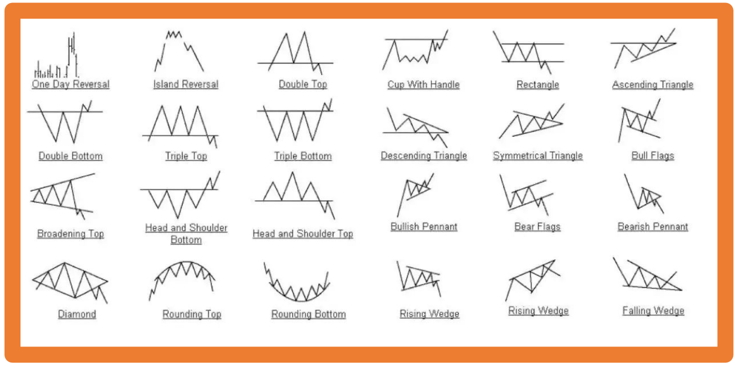
In fact, the **MAE** for that model was **6.4069** (*a difference of 0.1229*).

Reminder: Each principal component in PCA is a weighted linear combination of the original features, where the loadings (weights) indicate how much each feature contributes. The magnitude of a loading reflects the strength of a feature's influence on that component, regardless of direction.

We sorted the absolute value of loadings for the first 3 principal components and it painted the picture clearly. The first PC captured mostly the pricing aspect features (bid/ask price, reference price, etc.), the second PC captured mainly the sizing aspect features (bid/ask size, matched size, etc.), and the third PC captured sizing as well as timing (seconds_in_bucket) as well.

Note: This is before feature engineering, with addition of new features results could be different.

CNN - Motivation



Consider the best-known application of a CNN - pattern recognition in images

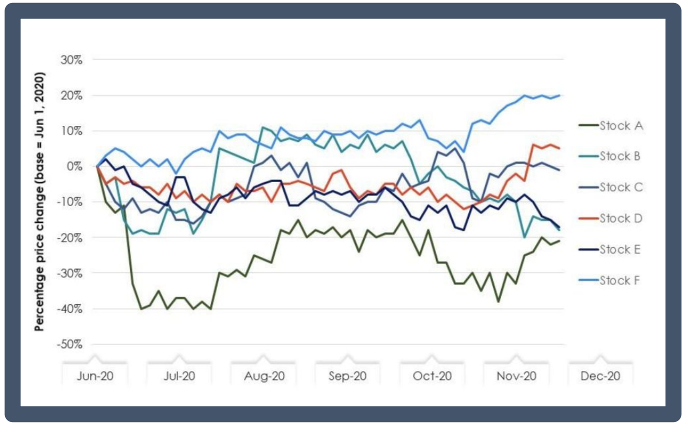
- Shared edges allow for the network to learn trends that, say, a fully connected network cannot
- Convolutional kernels can capture localized patterns that may be found at different locations within a dataset

This intuition from image processing extends well to time series analysis, making CNNs a good candidate model for the prediction of stock returns

CNN - Data Preprocessing

Subsets were taken for each of the 200 companies, and a CNN was trained on each.

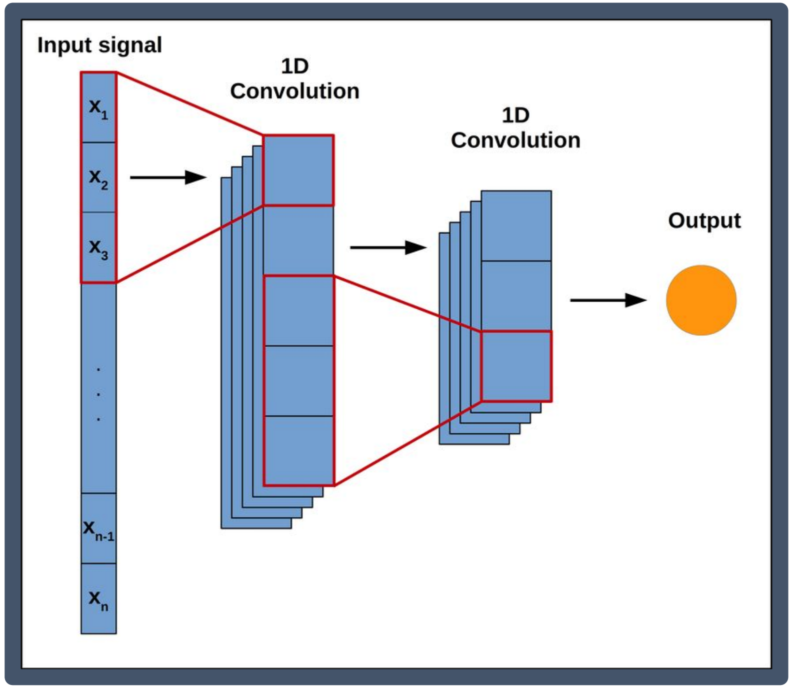
- Company #79 was dropped due to issues with its dataset, leaving us with 199 companies to analyze
- “Far price” and “near price” dropped due to high & concentrated NaN values, leaving us with 10 features
- Other NaN values were replaced by large negative numbers



CNN - Implementation

A standard network with two layers of convolutions is used

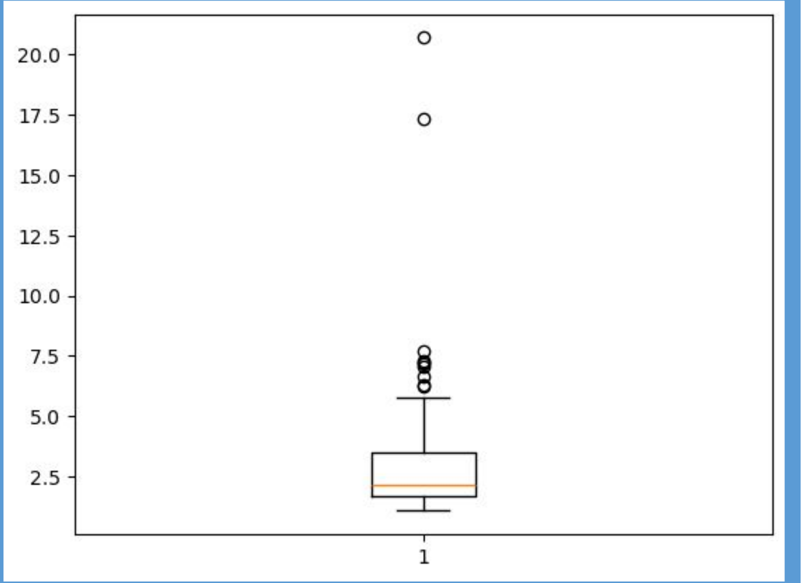
- 1D Convolution -> ReLU Activation -> Max Pooling for dimension reduction
- Two linear layers, the latter of which forms the final output
  - After the first layer, we utilize a 50% dropout rate (half of all neurons are randomly selected to be disabled)



CNN - Runtime and Performance

On a MacBook Air with an M3 chip, all CNNs took roughly 64 minutes to run, with 6 epochs of training for each network

- The low epoch count was chosen as the training loss was found to quickly converge within ~0.001
- Converged very well for select companies, achieving MAEs between 3.4-3.9
- Overall model performance was still relatively poor, with a median MAE of 8.45



Spread of the log mean absolute errors

CNN - Future Work

Ahead of the final report, the CNN model will be thoroughly improved:

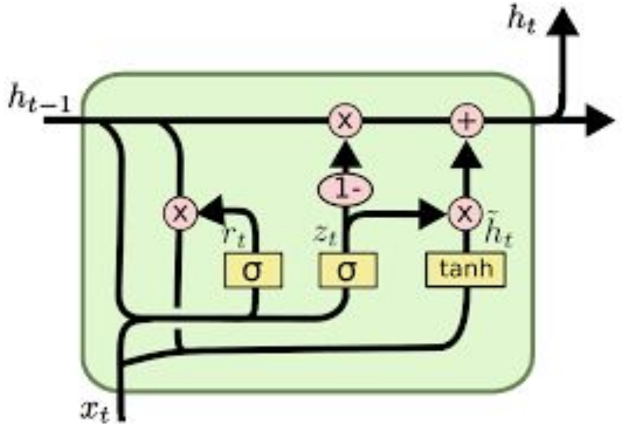
- **Diagnostics**: Investigate some of the companies with high MAEs, especially the outlier company
- **Research**: The architecture for the CNN is fairly simple - perhaps it could be more specialized for the dataset
  - Hyperparameter Fine-Tuning: Is a dropout rate of 50% ideal?
- **Computation**: Six epochs was a compromise, MAE has more room for convergence

GRU Intro

Motivation: We are dealing with an inherently time-dependent regression task.

What this means: An effective model should learn the time-based dependencies in the data to predict continuous future values.

Enter GRU: A gated recurrent architecture for temporal learning.



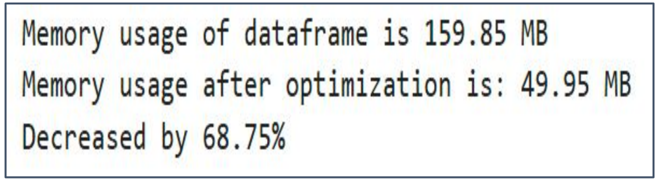
LSTM vs GRU

- Similarities
  - **Shared Goal:** Both LSTMs and GRUs are Recurrent Neural Network (RNN) architectures designed to address the vanishing gradient problem and effectively learn from sequential data.
  - **Core Mechanism - Gating Units:** Both utilize gating mechanisms to control the flow of information within the network.
- Differences
  - Only has 2 gates.
  - **Update Gate:** Acts similarly to the combined input and forget gates of the LSTM, controlling how much of the previous hidden state to keep and how much new information to incorporate.
  - **Reset Gate:** Controls how much of the previous hidden state to forget.
- Main Takeaway
  - GRU is a simpler architecture, leading to faster runtimes.

GRU Setup

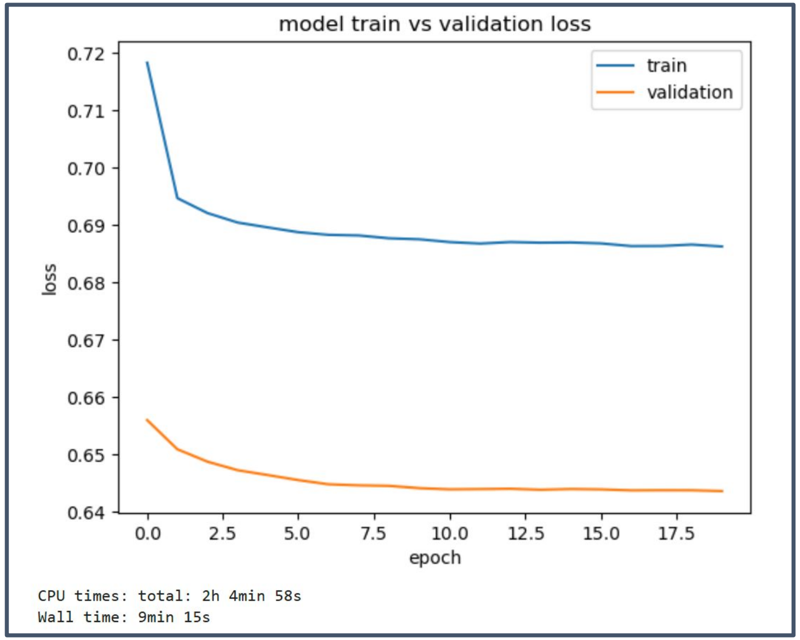
Workflow:

1. Data loading and cleaning
2. Memory optimization
3. Creating a complete time grid
4. Train/Test Split (390 days for train/90 days for test)
5. Overlapping windows for batching
6. GRU Model Building
   a. Input - > Dropout -> GRU -> Dropout ->Dense Output
7. Model Training
8. Model Evaluation

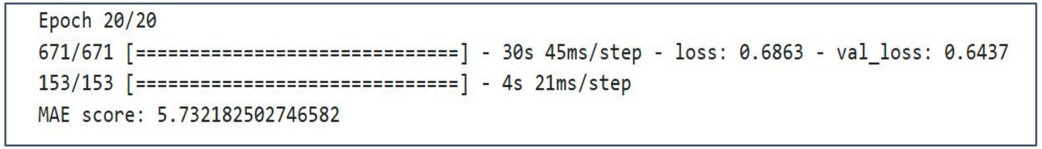


```
Memory usage of dataframe is 159.85 MB
Memory usage after optimization is: 49.95 MB
Decreased by 68.75%
```

GRU Results



```
CPU times: total: 2h 4min 58s
Wall time: 9min 15s
```



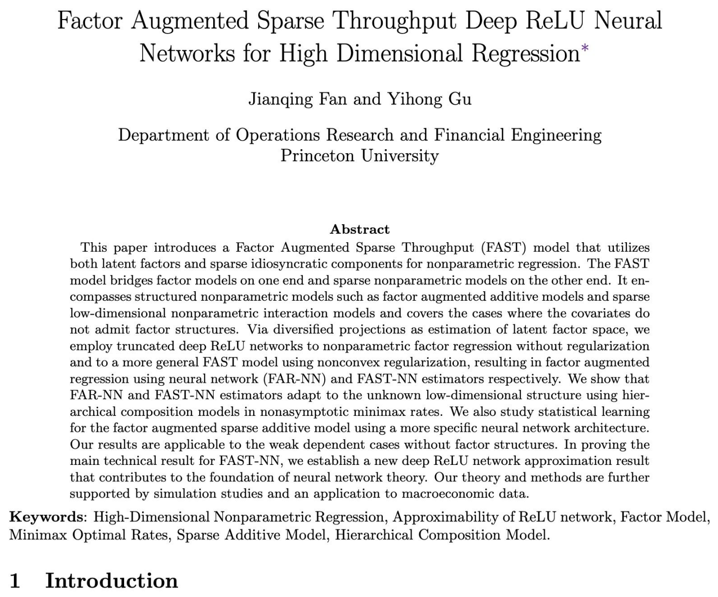
```
Epoch 20/20
671/671 [==============================] - 30s 45ms/step - loss: 0.6863 - val_loss: 0.6437
153/153 [==============================] - 4s 21ms/step
MAE score: 5.732182502746582
```

- MAE Score: 5.73
- Therefore, for any given stock, GRU will be on average 5.73 basis points away from the actual values.
- Or 0.0573% in terms of that relative price movement.

FAST-Transformer: Intro

- Based our model on the paper “Factor Augmented Sparse Throughput Deep ReLU Neural Networks for High Dimensional Regression” by Fan & Gu.
  - Broken up into the The Factor Augmented Spares Throughput (FAST) component and the neural network component
- Suppose we have a large number of features, but these features might be influenced by a smaller subset of underlying factors
  - FAST step of the model aims to uncover these hidden factors
- FAST-NN utilizes latent factors and sparse idiosyncratic components for non-parametric regression
  - Latent factors (f): variables outside of the context of our dataset that impact our predictor (impacts most if not all features)
  - Idiosyncratic components/throughput (u): underlying noise that remains after accounting for the shared influence of latent factors

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{f} + \boldsymbol{u}$$



Factor Augmented Sparse Throughput Deep ReLU Neural Networks for High Dimensional Regression*

Jianqing Fan and Yihong Gu

Department of Operations Research and Financial Engineering
Princeton University

Abstract

This paper introduces a Factor Augmented Sparse Throughput (FAST) model that utilizes both latent factors and sparse idiosyncratic components for nonparametric regression. The FAST model bridges factor models on one end and sparse nonparametric models on the other end. It encompasses structured nonparametric models such as factor augmented additive models and sparse low-dimensional nonparametric interaction models and covers the cases where the covariates do not admit factor structures. Via diversified projections as estimation of latent factor space, we employ truncated deep ReLU networks to nonparametric factor regression without regularization and to a more general FAST model using nonconvex regularization, resulting in factor augmented regression using neural network (FAR-NN) and FAST-NN estimators respectively. We show that FAR-NN and FAST-NN estimators adapt to the unknown low-dimensional structure using hierarchical composition models in nonasymptotic minimax rates. We also study statistical learning for the factor augmented sparse additive model using a more specific neural network architecture. Our results are applicable to the weak dependent cases without factor structures. In proving the main technical result for FAST-NN, we establish a new deep ReLU network approximation result that contributes to the foundation of neural network theory. Our theory and methods are further supported by simulation studies and an application to macroeconomic data.

Keywords: High-Dimensional Nonparametric Regression, Approximability of ReLU network, Factor Model, Minimax Optimal Rates, Sparse Additive Model, Hierarchical Composition Model.

1 Introduction

FAST Step Explanation

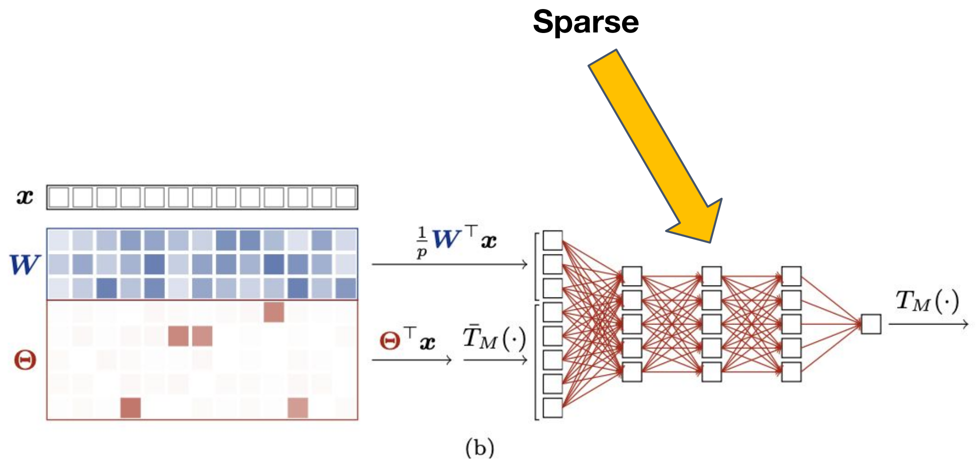
- Layer 1: Diversified Projection Matrix
  - Pretrained projection layer
  - Extracts latent information from the covariate x
  - Parameterized by r_bar (number of features to keep)
- Layer 2: Variable Selection Matrix
  - Jointly trained with NN
  - Extracts a sparse subset of important idiosyncratic variables (dimensionality reduction)

Diversified Projection Matrix (Layer 1) pseudocode

```
$\text{Input:} \quad X \in R^{n \times p}, \bar{r} \in N$
$p = shape(\mathbf{X})[1]$
$\mathbf{\Sigma}_X = \mathbf{X}^\top \mathbf{X}$
$\lambda_i, \mathbf{v}_i = eigsh(\mathbf{\Sigma}_X, \bar{r}, which =' LM')$
$\mathbf{W} = \frac{\mathbf{v}_i}{\sqrt{p}}$
$\mathbf{F} = \mathbf{X}\mathbf{W}$
$\mathbf{\Sigma}_F = \mathbf{F}^\top \mathbf{F}$
$\mathbf{\Sigma}_{FX} = \mathbf{F}^\top \mathbf{X}$
$\mathbf{R} = \mathbf{\Sigma}_F^+ \mathbf{\Sigma}_{FX}$
$Return : \mathbf{W}, \mathbf{R}$
```

Original Implementation

- Original implementation order of operations:
    - Diversified projection
    - Variable selection
    - Sparse MLP
- Loss: MSE + Regularization Loss
    - Regularization Loss: L1 penalty parameterized by a scaling hyperparameter Tau
        - Encourages sparsity



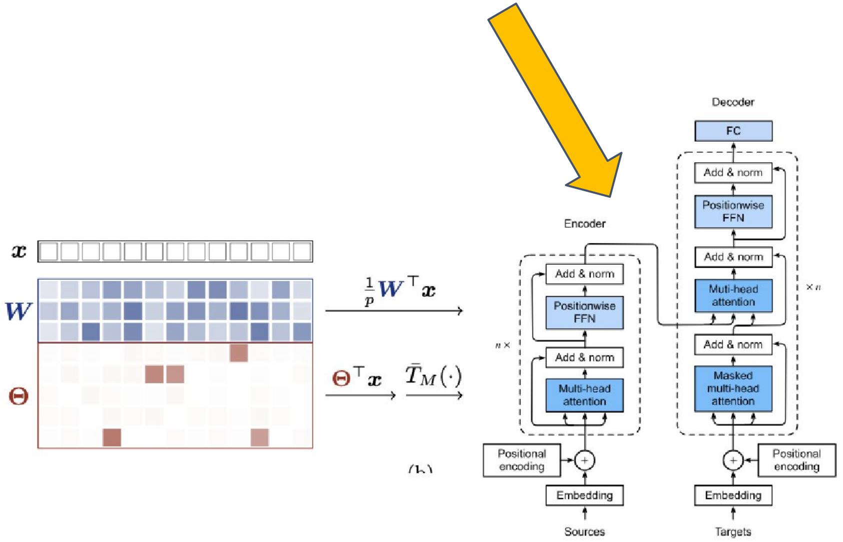
Adapting to Our Dataset

- Issues we needed to address
  - Original implementation was not compatible with time series
  - MLP likely not ideal for time series problem
  - Sparsity in NN could potentially lead to issues due to temporal nature of time series data
  - Likely requires additional feature engineering to take full advantage of FAST

Our Implementation

- Make necessary dataset adaptations for time series (redesign training loop and make custom data loader)
- Engineer denser features (rolling statistics, price change, etc.)
- Architecture
  - Keep FAST layers the same but use a transformer instead of MLP
    - Capture dense, long-range dependencies in our data (allows for more information captured through lags)
  - Use L2 instead of L1 penalty on transformer layers to control magnitude of attention weights

Weights Minimized via L2 Penalty



FAST-Transformer Results:

- Results on subset after 50 epochs
  - MAE: 5.49
  - RMSE: 6.7
  - Train Loss: 1.02
  - Val Loss: 1.07

Potential Improvements

- Feature engineering: Creating new and also meaningful features could aid in model improvements across the board.
- Hypertuning (specific to XGBoost): Increasing number of trees made, regularization term tweaking, etc.
- Increased computation & research for CNN
- Experimenting with stacked GRU layers could potentially increase the model's ability to learn complex temporal dependencies.
- Exploring other models: i.e. ARIMAX, AdaBoost, and CatBoost, SSM, etc.

Questions?