Hybrid LSTM-Transformer Approach for Intraday Financial Forecasting

Filippo Tomelleri, Sean Boos, Giovanni Parri, Said Haji Abukar

Advanced Machine Learning - University of Zurich

Introduction

- **Goal**: Predicting Intraday Financial Time Series using Transformers
- **Model**: based on the paper “*Trading with the Momentum Transformer: An Intelligent and Interpretable Architecture*” (arXiv:2112.08534)
- **Data**: Kaggle competition “Optiver - Trading at the Close”

Formal Problem Setting

- **Observables**: limit order book and auction data at time available at time $\boldsymbol{t}$ which we denote as $\boldsymbol{X_t}$
- The filtration $\boldsymbol{F_t}$ contains all the information available up to time $\boldsymbol{t}$
  - Formally, the filtration $\boldsymbol{F_t}$ is defined as: $\boldsymbol{F_t} = \sigma(\boldsymbol{X_t} : 0 \leq s \leq t)$
- **Objective**: Find a model f and predict $\boldsymbol{\hat{y}_{t+1}} = f(\boldsymbol{X}\square, \boldsymbol{\Theta})$ that minimizes the objective function
  - Formally, we define f to be adapted to the filtration $\boldsymbol{F_t}$. This ensures that the model only uses information known at that point in time.
- **Objective Function**: Mean Squared Error (MSE): $\boldsymbol{MSE(\hat{y}_{t+1} ,\ y_{t+1})}$

Features and target

- For each stock ID, historic data for the **daily ten minute closing auction** on the NASDAQ stock exchange
- **Features** on trade timing and more technical aspects such as imbalance size, bid/ask price, far/near price and the weighted average price (WAP) in the non-auction book
- **Target** measures the difference between the 60-second future move in the stock's WAP and the 60-second future move of a custom synthetic index.

Features Amplification

- We added features as:
  - Exponential moving average (**EMA**), which gives more weight to recent prices
  - **Spread**, referring to the difference between the bid price and the ask price
  - **Indices**, representing the collective performance of a group of selected stocks, providing a benchmark for assessing market trends and investment performance.
- D = 65 **features** in total

Data Preparation

- Splitting data according to time:
  - **training** ~ 70%, *“oldest”*
  - **validation** ~ 15%, *“2nd oldest”*
  - **test** ~ 15%, *“newest”*
- Scaling data:
  - features: **StandardScaler**
  - targets: **MinMaxScaler in [-1,1]**

- Splitting data by stock
- Turning data into **sequences** (S=10)

  $\rightarrow \boldsymbol{X} \in \mathbb{R}^{S \times D}$

  $\rightarrow \boldsymbol{T} \in \mathbb{R}^{1}$

- Turning sequences into **batches** (B=64)

  $\rightarrow \mathscr{X} \in \mathbb{R}^{B \times S \times D}$

  $\rightarrow \mathscr{T} \in \mathbb{R}^{BX1}$

Overview of Models

Model from Reference Paper

- Momentum Transformer

Variations of Momentum Transformer

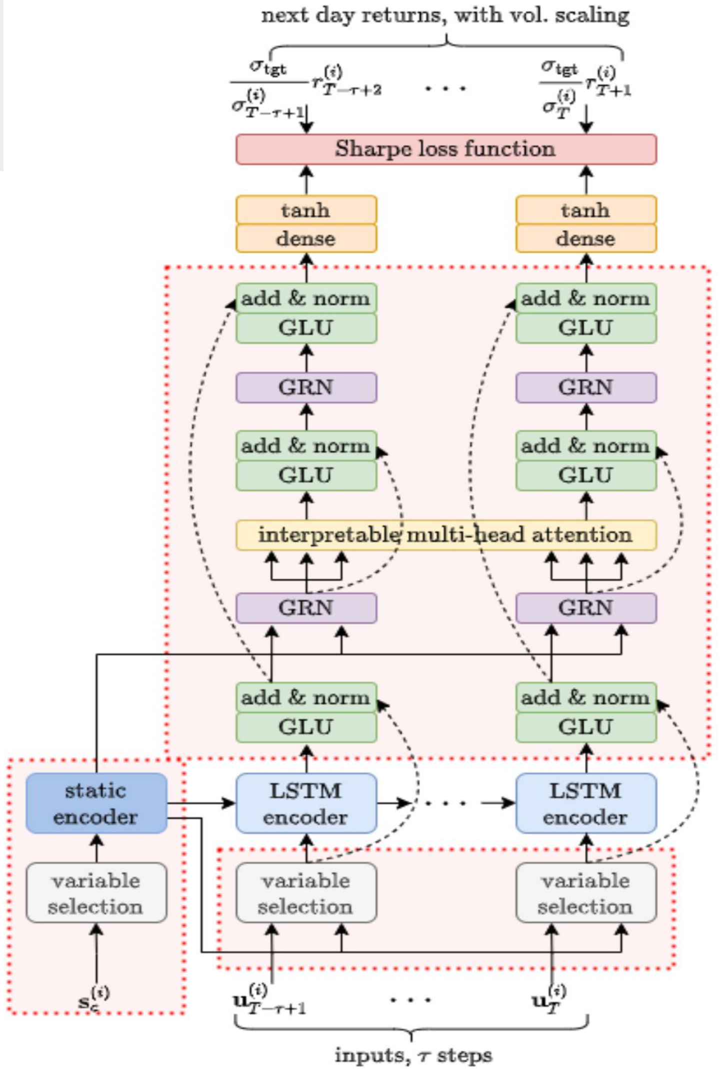
- Self-Attention Momentum Transformer
- GRU Momentum Transformer
- CNN Momentum Transformer

Benchmark Models

- LSTM
- Transformer (Encoder & Decoder)
- Transformer (Encoder Only)
- Random Forest

Reference Paper Model Architecture

- Inputs: **τ** time steps of the stock *i*
- Static Encoder
- Variable selection
- LSTM encoder
- Gated Linear Unit (GLU)
- Gated Residual Network (GRN)
- Interpretable Multi-Head Attention (IMHA)


next day returns, with vol. scaling
$\frac{\sigma_{tgt}}{\sigma^{(i)}_{T-\tau+1}} r^{(i)}_{T-\tau+2}$
$\frac{\sigma_{tgt}}{\sigma^{(i)}_{T}} r^{(i)}_{T+1}$
Sharpe loss function
tanh
dense
add & norm
GLU
GRN
interpretable multi-head attention
static encoder
LSTM encoder
variable selection
$\mathbf{s}^{(i)}_c$
$\mathbf{u}^{(i)}_{T-\tau+1}$
$\mathbf{u}^{(i)}_T$
inputs, τ steps

Variable Selection Network & Static Encoder

Variable Selection Network

- Based on the idea that you have multiple features that belong to one feature
- Take a weighted average of the features belonging to the same variable
- Not implemented in our models because it is not applicable

Static Encoder

- Encodes information for static covariates
- In the paper they used it to encode what kind of asset the model was predicting
- Not implemented because not directly applicable (all instances of the same asset class)

Gated Linear Units (GLU)

- Suppress components of the architecture that aren't necessary
- Applied position wise

$$\mathrm{GLU}(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right) \odot \left(\boldsymbol{W}_2 \boldsymbol{x} + \boldsymbol{b}_2\right)$$

- $x \in \mathbb{R}^{\mathrm{d}_{\mathrm{model}}}$: Input vector
- $W_1, W_2 \in \mathbb{R}^{\mathrm{d}_{\mathrm{model}} \times \mathrm{d}_{\mathrm{model}}}$: Weight matrices
- $b_1, b_2 \in \mathbb{R}^{\mathrm{d}_{\mathrm{model}}}$: Bias terms
- σ : Sigmoid activation function
- $\odot$: Element-wise multiplication (Hadamard product)

Gated Residual Network GRN

- Allows the model to only apply non-linear processing when needed
- Applied position wise

$$\mathrm{GRN}(\boldsymbol{x}, \boldsymbol{c}) = \mathrm{LayerNorm}(\boldsymbol{x} + \mathrm{GLU}(\boldsymbol{\eta}_1))$$
$$\boldsymbol{\eta}_1 = \boldsymbol{W}_1 \boldsymbol{\eta}_2 + \boldsymbol{b}_1$$
$$\boldsymbol{\eta}_2 = \mathrm{ELU}(\boldsymbol{W}_2 \boldsymbol{a} + \boldsymbol{W}_3 \boldsymbol{c} + \boldsymbol{b}_2)$$

- $x \in \mathbb{R}^{d_{\text{model}}}$: Input Vector
- $c \in \mathbb{R}^{d_{\text{model}}}$: Context vector (not applicable to our case)
- $W_1, W_2, W_3 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$: Weight matrices
- $b_1, b_2 \in \mathbb{R}^{d_{\text{model}}}$: Bias terms

Interpretable MHA

- Similar to standard MHA
- **Share values** across all attention heads
- *Given that different values are used in each head, attention weights alone would* ***not be indicative of a particular feature's importance***

$$\text{InterpretableMultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \tilde{\boldsymbol{H}}\ \boldsymbol{W}_H,$$

$$\begin{aligned}\tilde{\boldsymbol{H}} &= \tilde{A}(\boldsymbol{Q}, \boldsymbol{K})\ \boldsymbol{V}\ \boldsymbol{W}_V, \\ &= \left\{1/H \sum\nolimits_{h=1}^{m_H} A\left(\boldsymbol{Q}\ \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\ \boldsymbol{W}_K^{(h)}\right)\right\} \boldsymbol{V}\ \boldsymbol{W}_V, \\ &= 1/H \sum\nolimits_{h=1}^{m_H} \text{Attention}(\boldsymbol{Q}\ \boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\ \boldsymbol{W}_K^{(h)}, \boldsymbol{V}\ \boldsymbol{W}_V),\end{aligned}$$

Model Architecture: Self-Attention Momentum Transformer

Main Components

- LSTM Encoder
- GRN Components
- (Regular) Multihead Attention

Motivation:

- Finding out if using the custom attention architecture is beneficial over using regular multihead attention

Model Architecture: GRU Momentum Transformer

Main Components

- GRU Encoder
- GRN Components
- Interpretable Multihead Attention

Motivation:

- Literature tells us for some tasks LSTM networks are better than GRU ones but for others it’s the other way around.

Model Architecture: CNN Momentum Transformer

Main Components

- LSTM Encoder
- CNN Components
- Interpretable Multihead Attention

Motivation

- CNNs have the benefit of having fewer parameters than fully connected networks, making them better suited for handling noisy data due to their efficient feature extraction capabilities.

Architecture

- We treat the different time steps of the sequences as different channels.

Benchmark models

- Transformers (Encoder only and Encoder-Decoder)
- LSTM model
- Random Forest model

Transformer- Components Encoder

Neural network architecture designed for sequential data, leverages self-attention for efficient parallel processing to capture long-range dependencies, widely used in NLP.

- Transformer Encoder Layer:
    - Multi-Head Self-Attention: Allows the model to focus on different parts of the sequence simultaneously
    - Feed-Forward Network Applies Transformation to each position
    - Layer Normalization: Stabilizing training

Transformer- Components Decoder

- Target Mask: Each position in the target can only attend to previous positions, not future ones
- Transformer Decoder: Processes the target input (tgt) along with the encoded source (src)
- Output Linear Layer: maps output of the transformer to the target feature dimension
- Tan Activation: Ensures output values are between -1 and 1

Long Short-Term Memory (LSTM) Model

Long short-term memory (LSTM) is a variation of the RNN architecture. RNN models can only memorize short-term information, LSTM can better handle long term relationships in the data. Moreover, RNN models suffer from the vanishing gradient problem for long data sequences; however, LSTM can reduce this issue.

Components:

- LSTM Layer: Captures sequential dependencies
- Dropout Layer: Regularizes the model
- Linear Layer: Maps hidden state output to the desired output size
- Tan Activation: Ensures output values are between -1 and 1

Random Forest Model

The Random Forest model is bagging based ensemble learning method used for regression and classification tasks. It operates by constructing multiple decision trees during training and outputting the average prediction of the individual trees to improve predictive accuracy and control overfitting.

Components:

- RandomForestRegressor: sklearn model
- n_estimators: number of trees in forest
- max_depth: maximum depth of trees
- criterion: measurement of split quality (squared error)

Training

Training

- Batches shuffled at every epoch
- Gradient clipped
- Optimizer: **Adam**
- Loss function: **MSELoss**
- Long time to train some models (≈ 4 hours for GRU Momentum Transformer)
- **Mostly on CPU** because free GPU on colab was used up quickly

Hyperparameters Optimization & Evaluation Protocol

Hyperparameters Optimization

- Implemented **Random Gridsearch**
- Not enough power and time to run it for the large models

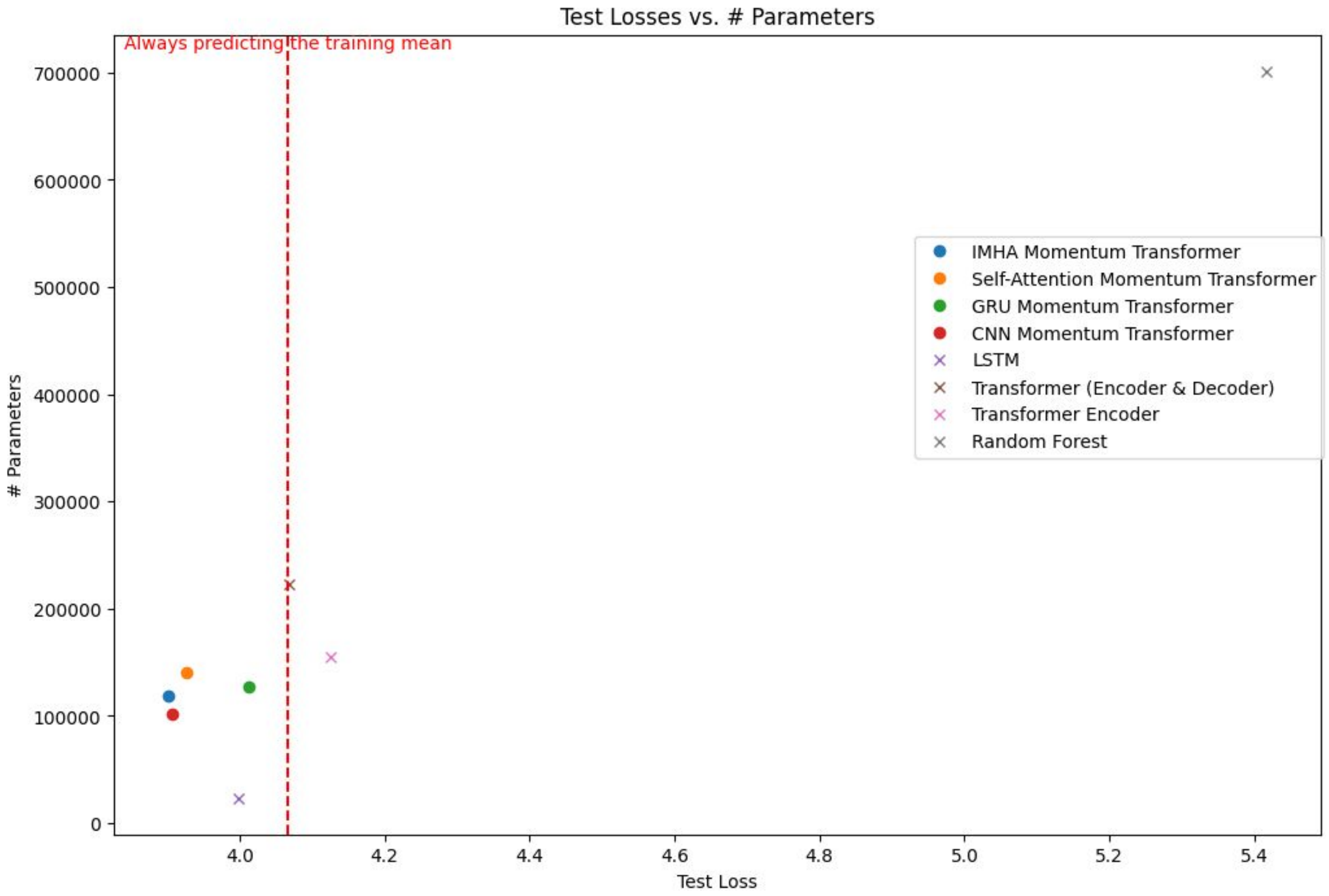
Evaluation Protocol

- Time-based Splitting (15% of the most recent part of the time series)
- Testing on **unseen Test Set**
- **MSE** as Evaluation Metric

Evaluation Results

MODEL	# PARAMETERS	TEST LOSS
IMHA Momentum Transformer	119246	**3.9019851267221384**
Self-Attention Momentum Transformer	140206	3.926034248928772**3**
GRU Momentum Transformer	126751	4.01184778381139
CNN Momentum Transformer	101919	3.907413617038401
BENCHMARK MODEL	**# PARAMETERS**	**TEST LOSS**
LSTM	23451	3.998927083594026
Transformer (Encoder+Decoder)	223205	4.068714729597559
Transformer Encoder	154701	4.125702866062056
Random Forest	701328 (# of Leaf Nodes)	5.415680542938186 (based on test data subset)
Constantly Guessing Training Mean	0	4.0670643

Evaluation Results



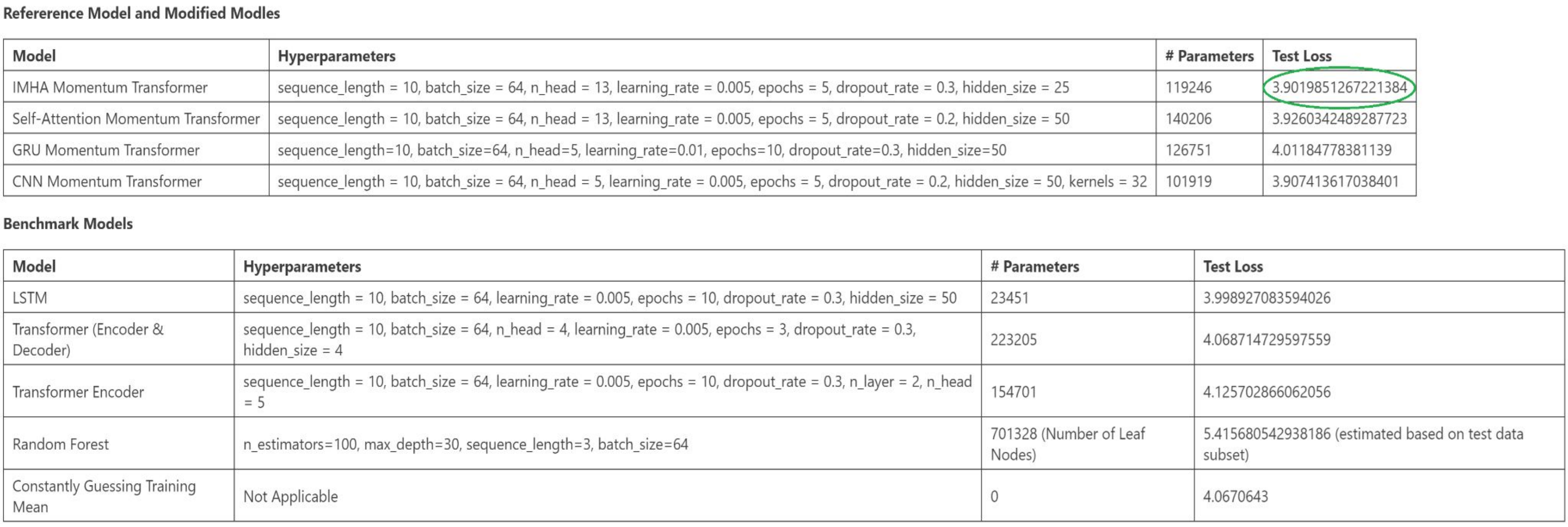
Conclusions

- Results should be taken with a grain of salt given that the Hyperparameters were not optimized
- After Hyperparameters Optimization results might be different
- Momentum Transformers work **better than benchmark models**
- More parameters $\nRightarrow$ better results (at least across different models)
- Modification study showed that for our given setup the LSTM, GRN and IMHA combination proposed by the paper performed better than the presented alternatives (GRU, CNN, regular MHA)
- The CNN modification also performed well, suggesting potential in using CNNs for financial time series prediction, specifically by treating sequence elements as separate channels.

References

- "*Trading with the Momentum Transformer: An Intelligent and Interpretable Architecture*" (https://arxiv.org/abs/2112.08534)
- "*Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting*" (https://arxiv.org/abs/1912.09363)
- Data Set https://www.kaggle.com/competitions/optiver-trading-at-the-close

Appendix - Complete Testing Results



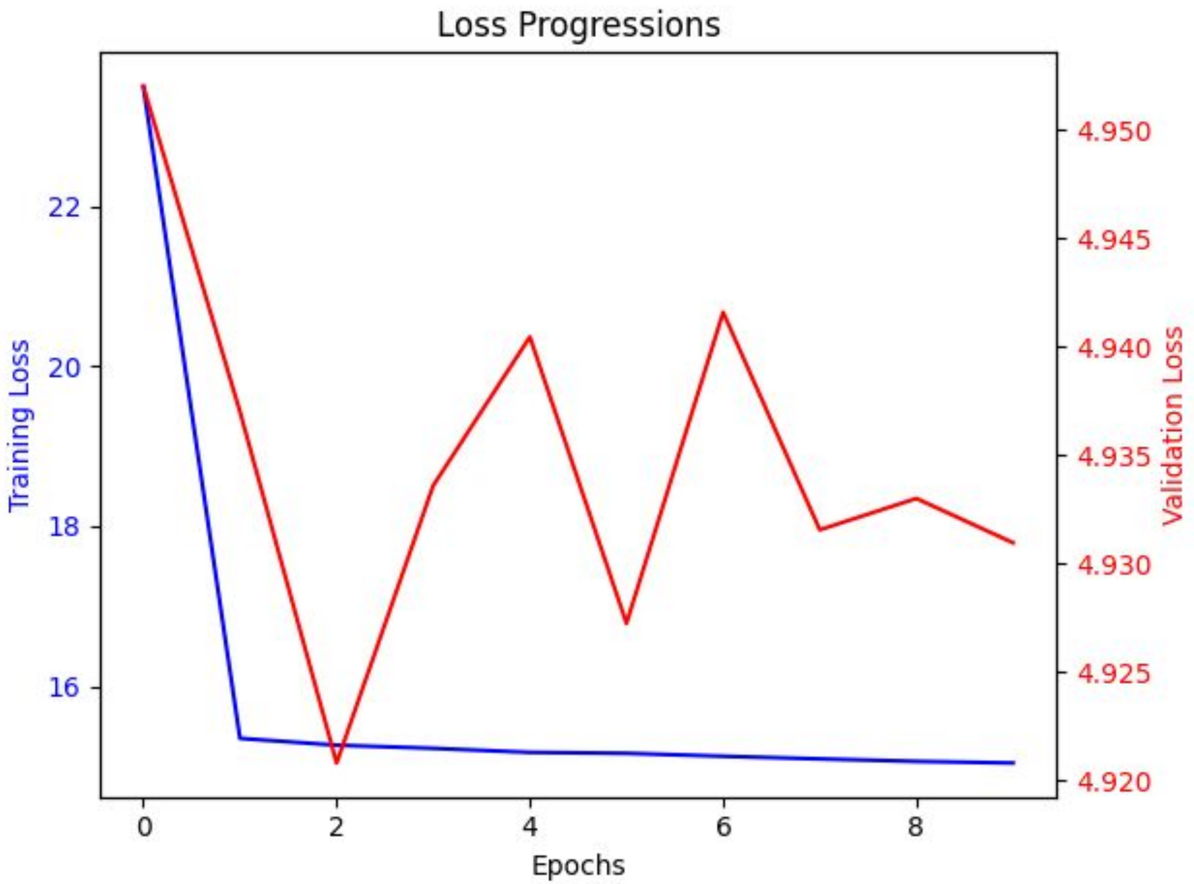
Refererence Model and Modified Modles

Model	Hyperparameters	# Parameters	Test Loss
IMHA Momentum Transformer	sequence_length = 10, batch_size = 64, n_head = 13, learning_rate = 0.005, epochs = 5, dropout_rate = 0.3, hidden_size = 25	119246	3.9019851267221384
Self-Attention Momentum Transformer	sequence_length = 10, batch_size = 64, n_head = 13, learning_rate = 0.005, epochs = 5, dropout_rate = 0.2, hidden_size = 50	140206	3.9260342489287723
GRU Momentum Transformer	sequence_length=10, batch_size=64, n_head=5, learning_rate=0.01, epochs=10, dropout_rate=0.3, hidden_size=50	126751	4.01184778381139
CNN Momentum Transformer	sequence_length = 10, batch_size = 64, n_head = 5, learning_rate = 0.005, epochs = 5, dropout_rate = 0.2, hidden_size = 50, kernels = 32	101919	3.907413617038401

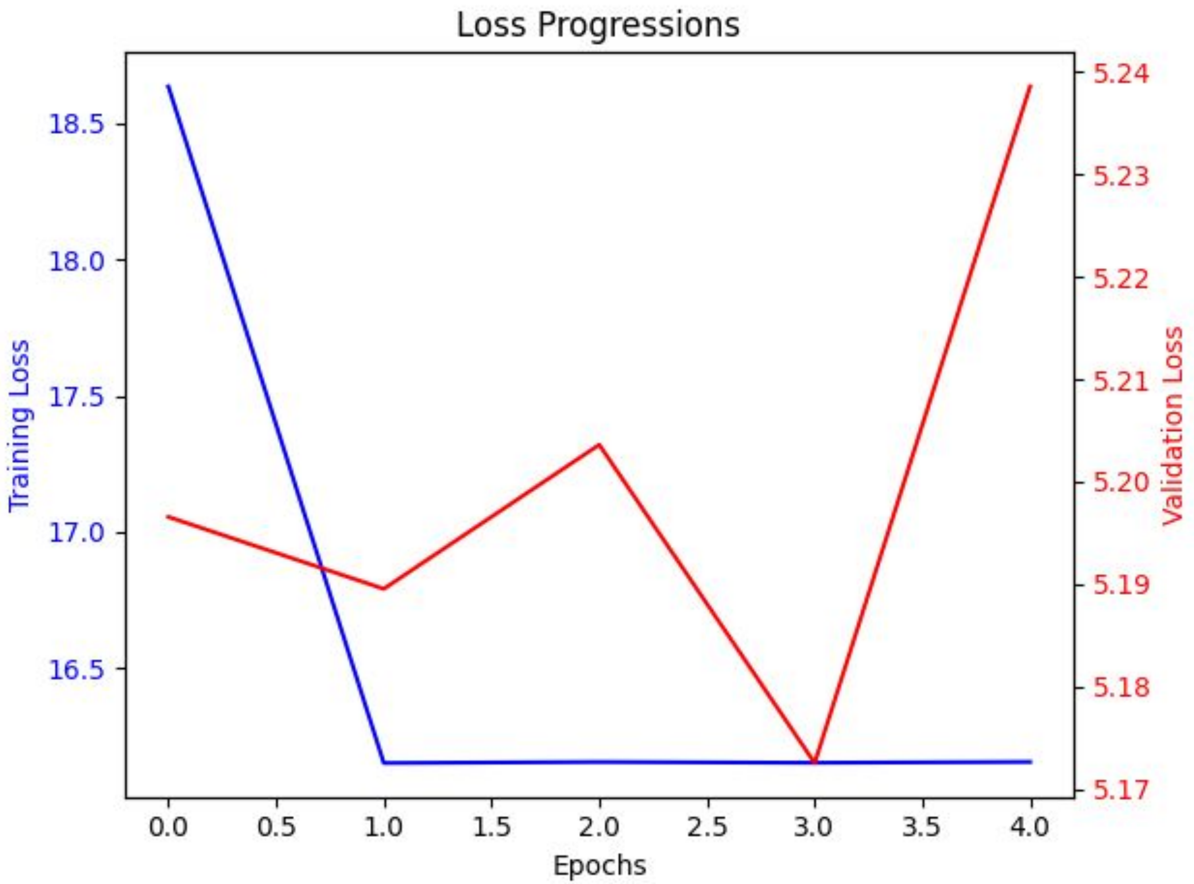
Benchmark Models

Model	Hyperparameters	# Parameters	Test Loss
LSTM	sequence_length = 10, batch_size = 64, learning_rate = 0.005, epochs = 10, dropout_rate = 0.3, hidden_size = 50	23451	3.998927083594026
Transformer (Encoder & Decoder)	sequence_length = 10, batch_size = 64, n_head = 4, learning_rate = 0.005, epochs = 3, dropout_rate = 0.3, hidden_size = 4	223205	4.068714729597559
Transformer Encoder	sequence_length = 10, batch_size = 64, learning_rate = 0.005, epochs = 10, dropout_rate = 0.3, n_layer = 2, n_head = 5	154701	4.125702866062056
Random Forest	n_estimators=100, max_depth=30, sequence_length=3, batch_size=64	701328 (Number of Leaf Nodes)	5.415680542938186 (estimated based on test data subset)
Constantly Guessing Training Mean	Not Applicable	0	4.0670643

Appendix - LSTM benchmark model



Appendix - Transformer (Encoder) benchmark model


Loss Progressions
Training Loss
Validation Loss
Epochs