Deep Learning Final Report

Interest Rate Prediction

Ayman Khan (1006313) , Manishansh Shaswat (1007124) , Mikhail Y. Jeffrey (1007491)

Introduction

Predicting interest rates is a challenge in the fields of finance and economics. Traditionally, interest rates were predicted through market analysis and key economic indicators. However, the complex relations between these factors make it difficult to accurately predict interest rates, but recent advances in machine learning could solve this problem.

This paper aims to explore the application of deep learning techniques in predicting interest rates. For better specificity, this paper will focus on the U.S. interest rates. By including some macroeconomic factors which might have been overlooked, and experimenting with different types of deep learning models, we aim to find a model which could accurately and robustly forecast U.S. interest rates.

Dataset

For the prediction of U.S. interest rates, the merged_dataset.csv integrates key macroeconomic indicators which impact financial values. Firstly, Core Consumer Price Index (**CPILFESL**), Core Personal Consumption Expenditure Price Index (**PCEPILFE**), and Sticky Price Consumer Price Index (**CORESTICKM159SFRBATL**) produced measurement of underlying inflation trend by utilising guiding Federal Reserve policies. For long-term bond yields, the 30-Year Expected Inflation Rate (**EXPINF30YR**) determines prolonged interest rate outlooks. Metrics associated with Gross Domestic Product (**GDP**), Personal Saving Rate (**PSR**), and Unemployment Rate (**UNRATE**) represent broader economic performances since low unemployment often promotes a spike in interest rates. Disposable Personal Income (**DSPIC**) and M2 Money Supply (**M2**) are a reflection of consumer spending capacity associated with inflation. Moreover, for providing insights into manufacturing strength and consumer demand, Industrial Production Index (**INDPRO**) and Personal Consumption Expenditures (**PCE**) were incorporated as metrics into the dataset. Real Effective Exchange Rate (**REER**) essentially captured U.S. dollar global competitiveness in the market ultimately transforming trade. The Federal Funds Effective Rate (**FFER**) provides a clear perception of Fed’s monetary stance. Lastly, U.S. Dollar Index Values (**Price, Open, High, Low, Change%**), emphasised on market sentiment eventually enabling deep learning models to comprehend complex influencers of interest rate movement.

Related Works

1. Deep Dynamic Factor Model (D^{2}FM) - University of Warwick

The Deep Dynamic Factor Model (D^2FM) has been developed by the research team at the University of Warwick. It utilises autoencoder-based neural networks for extraction of latent data from large-scale financial time series data. This model is essential in the improvement of forecasting economic indicators comprising interest rates and GDP through capturing nonlinear dependencies which traditional factor models fail to identify. [1]

2. LSTM-Based Forecasting - Bank of England

The analytics team at the Bank of England employed Long Short-Term Memory (LSTM) networks for forecasting inflation trends for demonstration of deep learning model's capabilities in capturing non-linear patterns evident in macroeconomic data. This achievement outperforms traditional autoregressive methods due to better adaptation to economic shocks in addition to providing support for a more accurate forecasting in policy-making decisions. [2] Their gating mechanisms ensure captivation of compounding or delayed effects fluctuating across inflation rates, monetary policy indicators, and GDP growth. Thus, they are considered a leading model for dynamic economic forecasting.

3. Deep Vector Autoregression (Deep VAR) - Barcelona School of Economics

Deep VAR has been implemented at the Barcelona School of Economics. It comprises an advanced deep learning alternative to basic vector autoregression. It employs multilayer neural networks for comprehension of nonlinear relationships evident across macroeconomic indicators such as unemployment, interest, and inflation rates. Deep VAR is highly suitable for high-impact economic forecasting since it produces accurate predictions by modelling complex dynamics which are often overlooked by VAR models. [3]

Methods

Three types of models will be trained and evaluated for this task:

1. LSTM
2. Seq2Seq
3. GRU

Each model type will start with a generic base model, then changes will be made iteratively to try and improve the model. These iterations will be referred to as “generations”. Performance of each model will be tracked through comparison to a test set, visualized as a line graph.

Experiments & Result

Models Evaluated

LSTM

Dependencies:

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from torch.utils.data import DataLoader, TensorDataset
```

This section covers the models which make use of the LSTM. All models covered in this section use the Adam optimizer with learning rate halved. Trainer function for this class of model implements early stopping, this is because LSTMs are relatively more prone to overfitting. The data was processed into sequences for the LSTM input, doing such trades the number of possible samples for more accuracy. The window size of each sequence is 30.

Each generation starts with new initialised weights, except for the Sixth Generation which uses weights from the pre-trained fifth generation.

<u>First Generation:</u> This model acts as the starting point for improvements. At this stage, most hyperparameters were a guess, except hidden size. Hidden size for LSTM was chosen based on the rule of thumb $N = \frac{\#Samples}{(\#Inputs + \#Outputs) * Dimensionality}$. The loss function was chosen to be MSELoss, this is because MSE is very sensitive to large mistakes - which we want to minimize.

As for the trainer function, an early stopping mechanism is implemented with a patience of 8. The model will be trained on mini-batches of size 512.

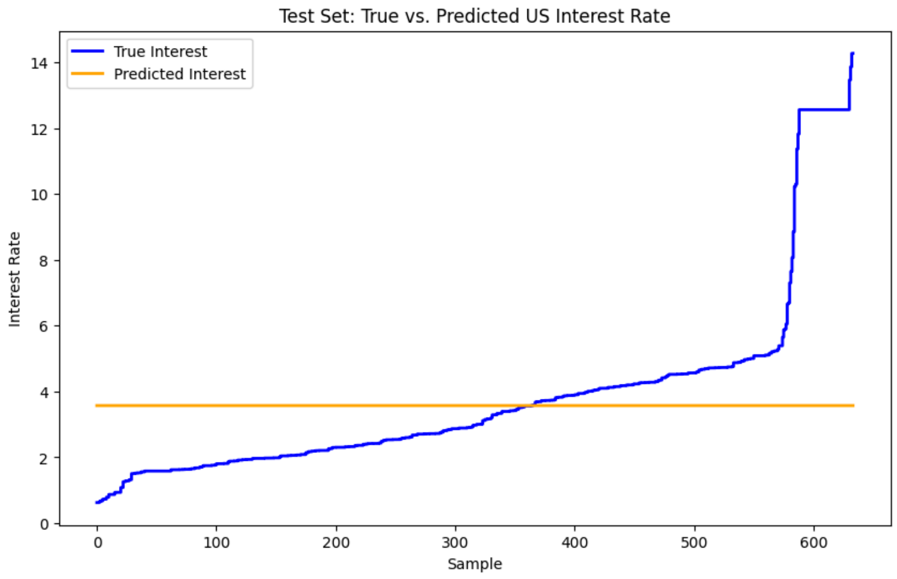
Input Size = 21

Hidden Size = 272
No. Layers = 4

Architecture: LSTM + Dropout(0.35) → BatchNorm1D → Linear

For clarification, the dropout layer is applied after every individual LSTM layer. This was done by changing the "dropout" parameter of the LSTM. This comes with its own advantages and disadvantages. One advantage is that since the dropout layers are applied after every LSTM layer, the LSTM would have better generalizability compared to just one dropout layer applied at the end. But the cost of this is that the value of the dropout layer cannot be changed dynamically during training.

The model ended on a final loss value of 7.965 on the test data. After an early stop after 40 epochs.



Output of First Generation

As seen on the chart above, the model minimized error by predicting one value. This is most likely due to the fact that interest rate does not change very often, so choosing an average value would work for most cases. The range of magnitudes of interest rates might also have played a role in this.

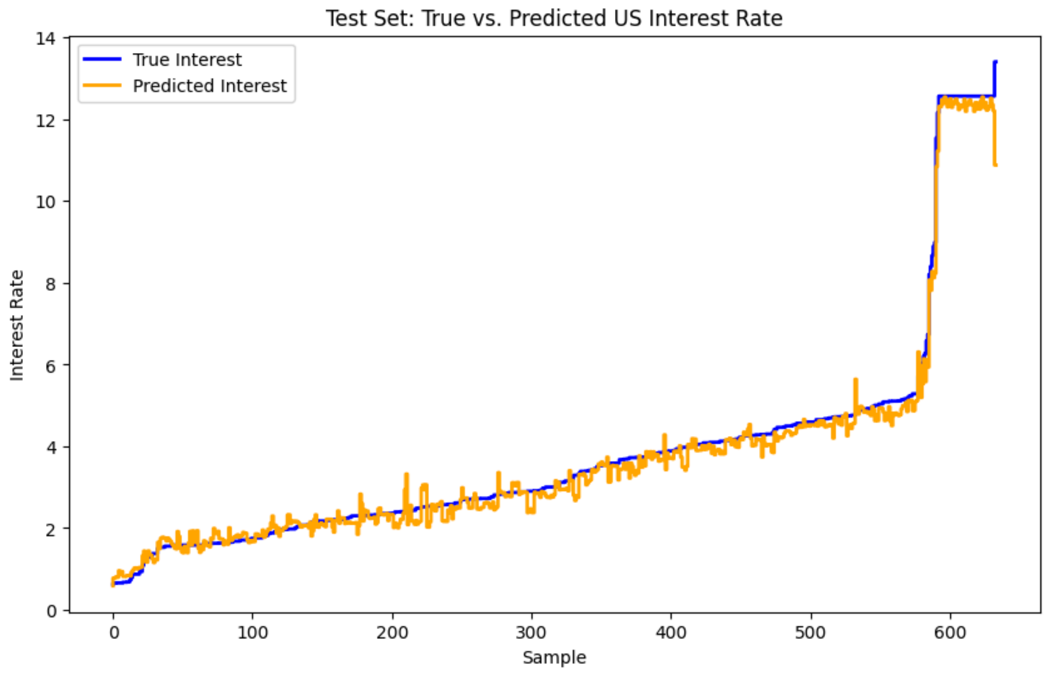
<u>Second Generation:</u> Since interest rate has to be accurate to the thousandth, and the magnitude of interest can go up to the tens, the second generation makes use of RobustScaler() from sklearn. The purpose of RobustScaler is to normalize the data. RobustScaler was specifically chosen as it deals with outliers better - this is especially

helpful for this context as interest rate can be affected or fixed by many outside factors not captured in the dataset. Two different RobustScalers will be used for:

1. Interest Rate
2. Input Features

They need to be separate as it will allow for the values to be inverted back for presentation purposes. Everything else is kept the same.

This time, the early stop was triggered at epoch 28, which implies that the model trained quicker. The model ended with a loss value of 0.0107 on the test data. This value is magnitudes different from the first generation, but it is mostly because of the scaling.



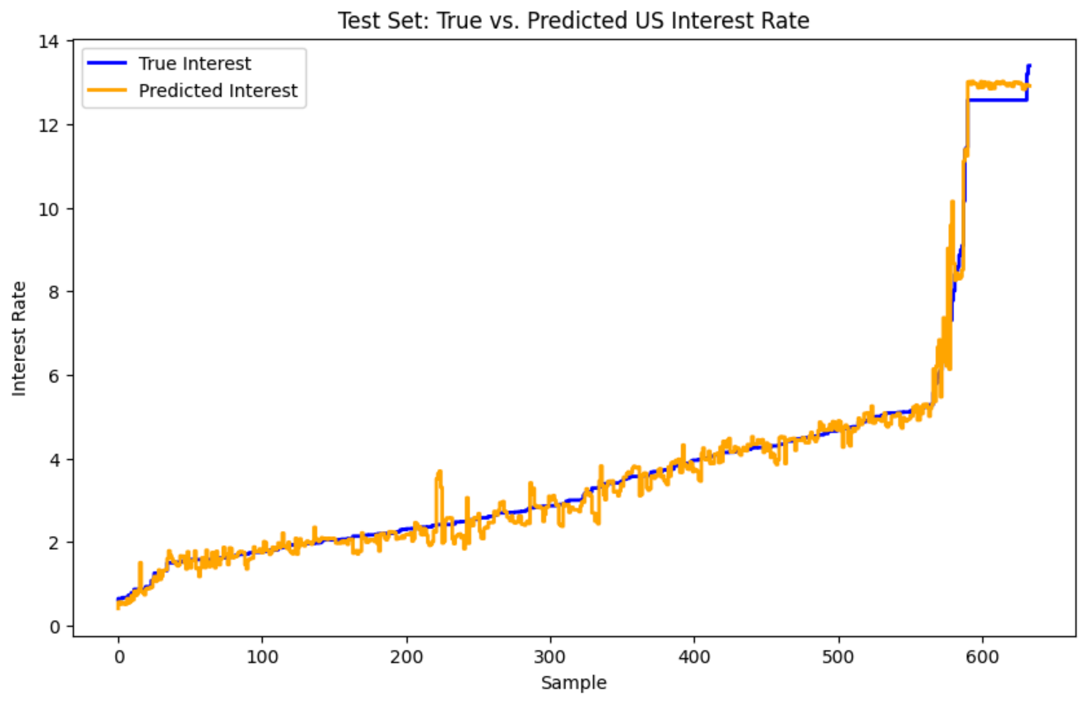
Output of Second Generation

As seen above, the performance of the second generation has significantly improved from the first. It looks like the trend is captured quite accurately but the model's predictions are still quite erratic. In practical use, this model would be good at predicting long term trends of the U.S. interest rate - it can provide a rough sketch of what is to come in the long run. But in terms of predicting interest rates for any given month, there could be a problem.

<u>Third Generation:</u> The previous generation had its training cut short by the early stop mechanism at epoch 28. Although early convergence at the global minimum might be a possibility, it is more likely that it converged on a local minima. To check for this, the third generation divides the current learning rate by a factor of 10 (lr = 0.00005). In addition to changing the learning rate, it would also make more sense to use L1 Loss

instead of MSELoss as the model trains on the scaled values. To account for the smaller learning rate, the patience for early stopping will be doubled, from 8 to 16 steps.

After training the model ended up with an test L1 loss of 0.080609, which if squared would be approximately 0.0065, significantly lower than the squared loss from the previous generation.



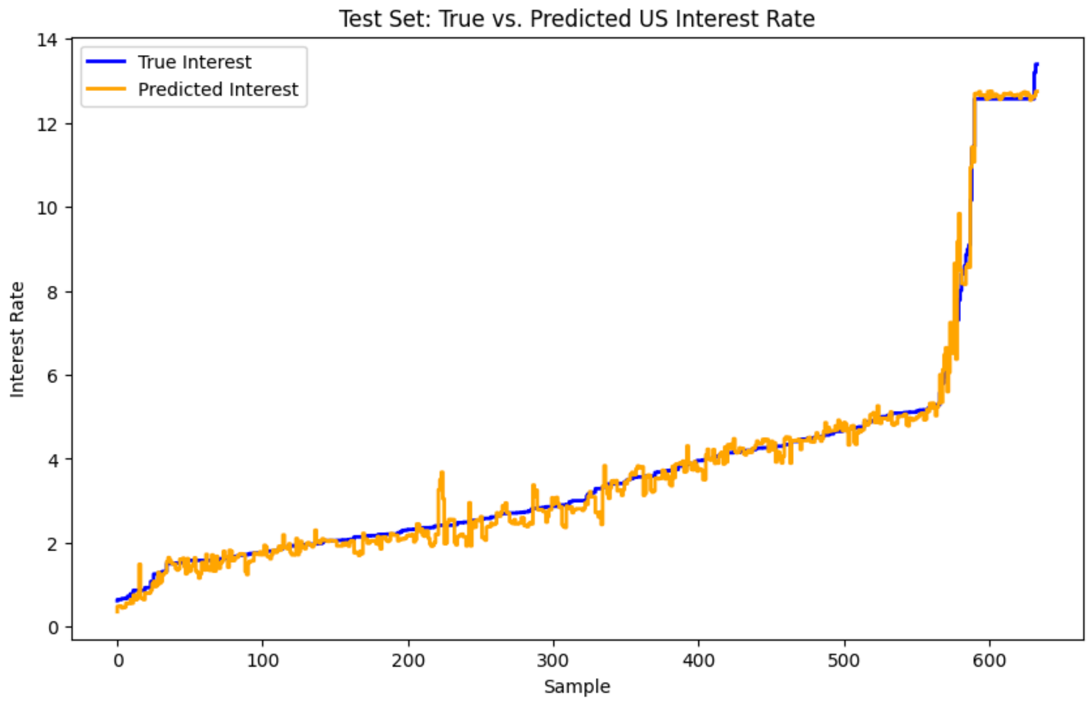
Output of Third Generation

This lower loss is reflected in this graph. The predicted interest seems to be closer to the true interest, while in the previous generation, it looked like the orange line was slightly below the blue line. However, the values are still quite erratic.

One observation made during the training is that in the beginning, the validation loss seems significantly higher than the train loss, but as it converged, the train loss was well above the validation loss. This could mean that the model is underfitting to the train data. One possible way to test this is to decrease or completely remove the dropout layer.

<u>Fourth Generation:</u> The fourth generation decreases the dropout rate to 0.1. The dropout layer was not completely removed to preserve some generalization ability of the model.

The model trained for all 100 epochs, but this time the training loss was not too much higher than the validation loss. There was also a downward trend for both training and validation loss every 10 epochs. The model ended with a test L1 loss of 0.0722, which is an improvement from the previous generation.



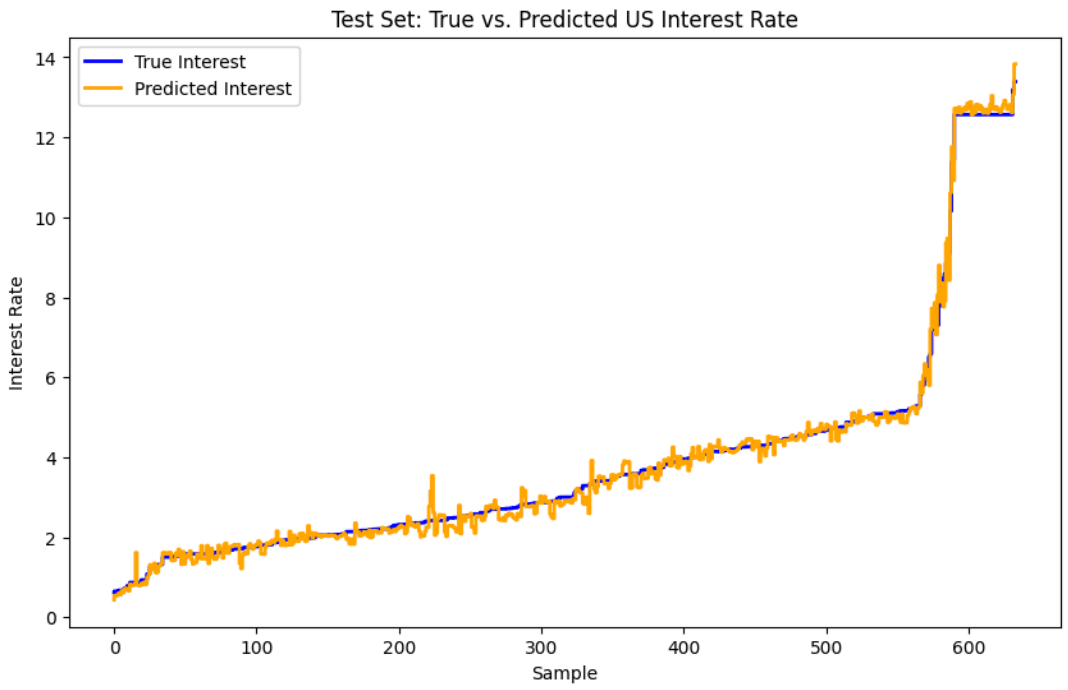
Output of Fourth Generation

The output for this model seems to be able to handle the fixed interest rate pretty well - so the decrease in loss might have come primarily from that section. However, the pattern of high validation loss in the start and high training loss at the end persisted.

One interesting observation with the loss values is that the more they start getting lower in magnitude, the more difficult it is for the model to train.

<u>Fifth Generation:</u> To address this problem, the loss function will be changed again - this time to RMSE, but since PyTorch does not have a built-in RMSE function, a custom function was created by applying torch.sqrt() to the MSELoss function.

Epochs and patience were increased to 200 and 20 respectively to encourage more training. The model managed to train for all 200 epochs and ended with an RMSE loss of 0.0814 on the test data, this translates to a 0.0581 MAE loss.

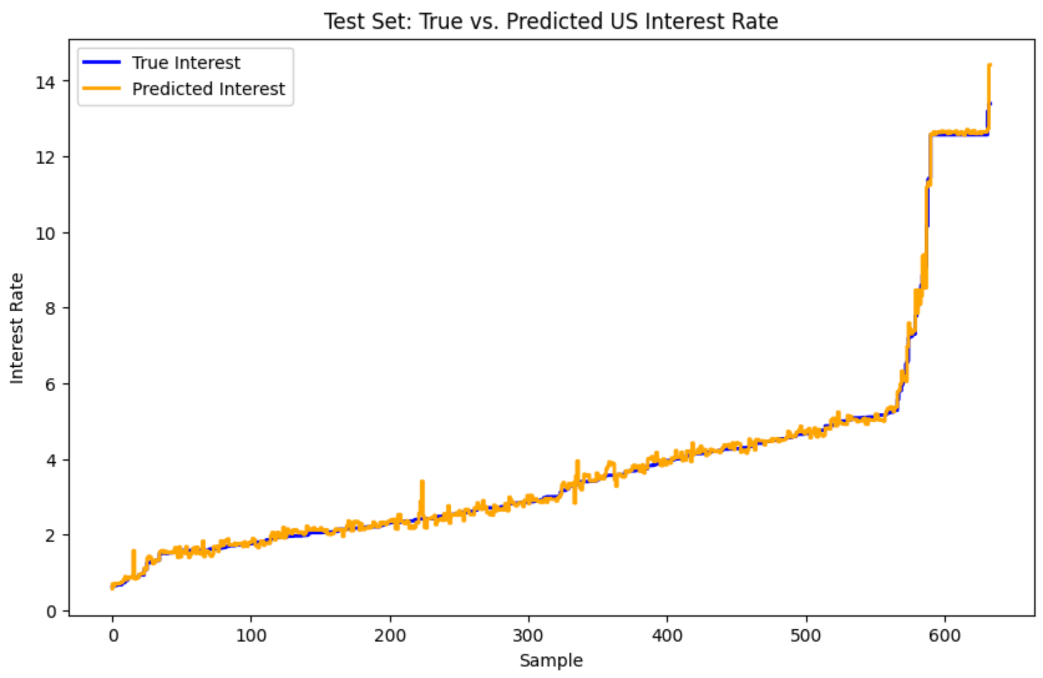


Output of Fifth Generation

The new chart seems less erratic than before, and captures the trend of the true interest rate even better. Furthermore, the difference in training loss and validation loss was not very high. The training loss was 8% higher than the validation loss, which is not out of the ordinary since the dropout rate was set to 10%.

From the trend of training and validation loss and the fact that no early stop was triggered, it seemed like the model had not yet reached a point of convergence. So it ran through another 400 epochs of training. Patience is kept the same.

An early stop was triggered at epoch 227. The model had a RMSE loss of 0.0594 on the test data, equivalent to an MAE loss of 0.0382; a huge improvement from the first 200 epochs of training.

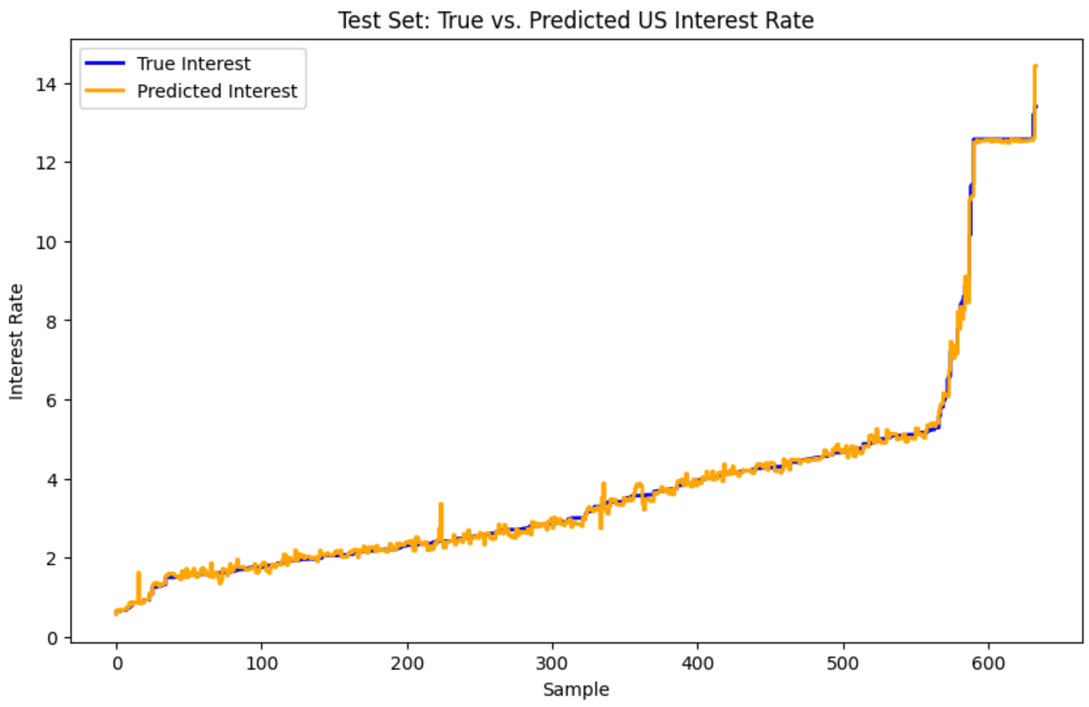


Output of Fifth Generation After 427 Epochs

Seems like using more blatant loss functions, functions where the loss is of higher magnitude, improves the performance of the model.

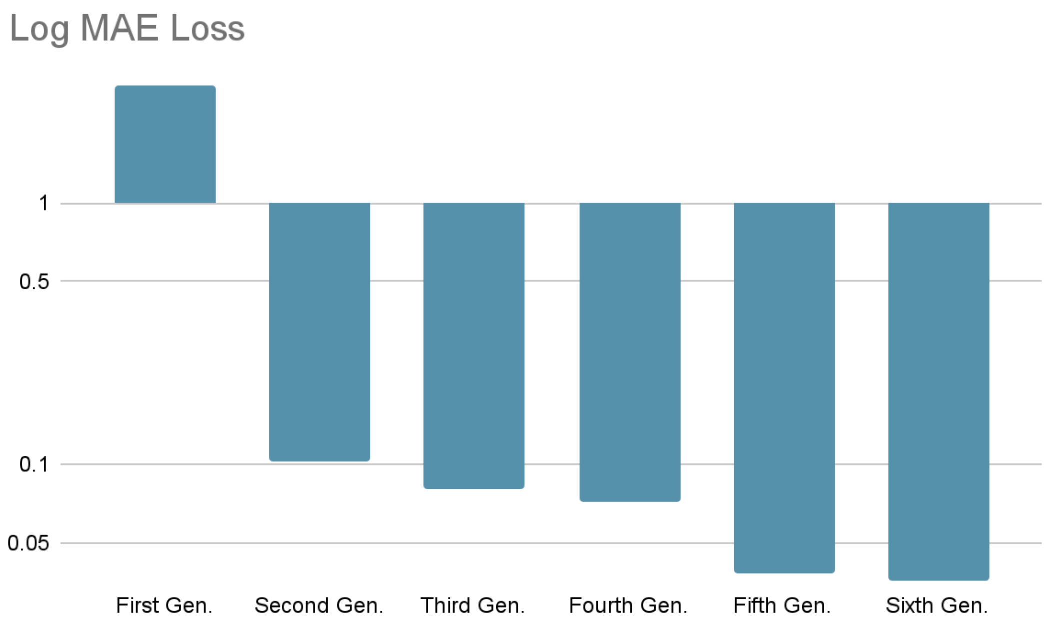
<u>Sixth Generation:</u> This generation makes use of a custom loss function named "DoubleRMSE", it works like RMSE but with two torch.sqrt() instead of just one on MSELoss. The goal of this is to make the decimal loss value bigger. This generation will also be the first to use weights from the previous generation. This is done because the initial loss has to be less than 1 for DoubleRMSE to work as intended.

The model trained until 122 epochs. The improvement with this new loss function was incremental. The final loss during testing was 0.2358, which is approximately 0.0355 in MAE loss.



Output of Sixth Generation

The chart looks very similar to the fifth generation chart. Since the fluctuations in prediction have somewhat stabilized, this model is fairly appropriate for short term interest rate predictions as well as long term interest rate predictions.



MAE Loss By The End of Each Generation on a Logarithmic Scale

Through analysing the bar chart above, it is clear that the most effective changes were from the first to second generation, second to third generation, and fourth to fifth generation.

Respectively, the changes made were:
- Used RobustScaler
- Decrease Learning Rate, Use L1 Loss (MAE Loss), Increase Patience
- Use RMSE Loss, More Training

Seq2Seq

Dependencies:

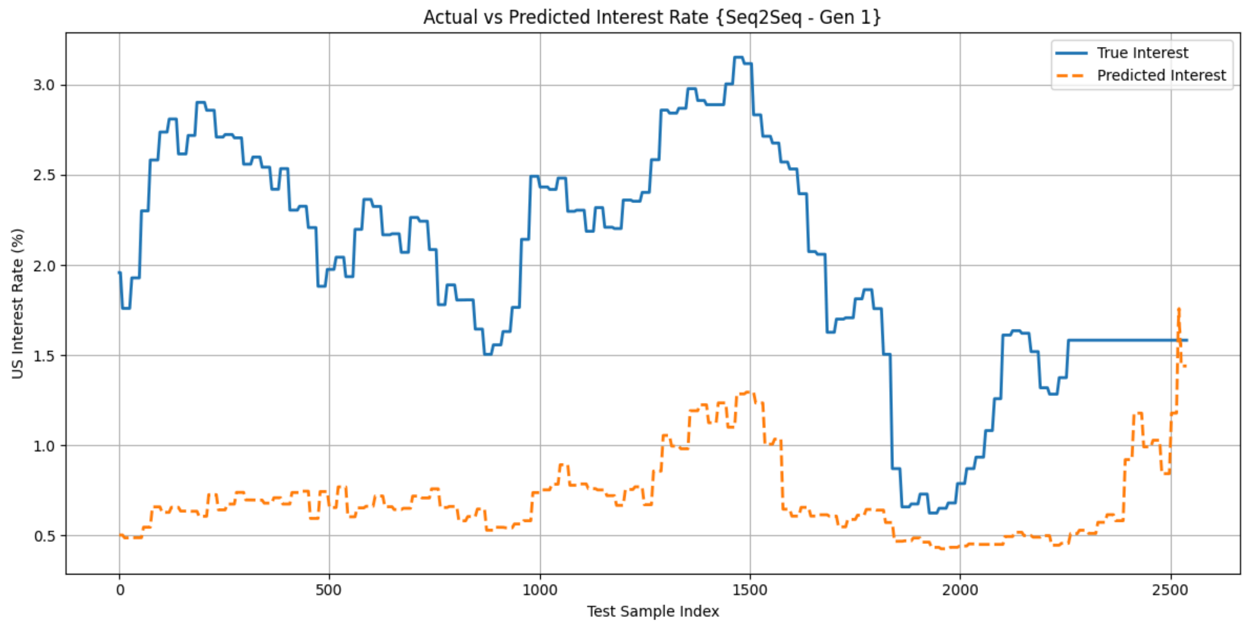
```
import numpy as np
import pandas as pd
import torch import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,
accuracy_score, f1_score
 import matplotlib.pyplot as plt
import seaborn as sns
```

This section emphasises on the progressive development of a PyTorch-based Sequence-to-Sequence (Seq2Seq) model. It comprises Long Short-Term Memory (LSTM) networks for multi-step time series forecasting of U.S. interest rates. Each iteration incorporated incremental modifications to the model's preprocessing pipeline, loss function, and optimized architecture. A range of macroeconomic indicators as input features such as inflation rate, personal saving rate, unemployment rate, gross domestic product and more has been leveraged by the model. These features are structured into a sliding window of 30 consecutive time steps as an input for the model. The Seq2Seq architecture is then trained for forecasting the corresponding sequence of interest rate values in a regression setting.

Targeted modifications to the preprocessing pipeline (e.g. scalability), optimisation configuration (e.g. learning rate scheduler, loss function), or architecture arrangement (e.g. number of LSTM layers, dropout rate) were introduced in each generation. Each model is trained using the Adam optimiser following a robust learning rate approach using ReduceLROnPlateau for improving convergence to minima. This methodical experimentation's objective is to enhance long-term trend tracking alongside short-term prediction accuracy in a fluctuating financial forecasting environment.

<u>First Generation:</u> The initial generation is implemented as a control baseline. The model employs a 2-layer LSTM encoder-decoder architecture with a hidden size of 129 and a dropout regularisation of 0.3. Notably, this generation omits any form of input normalisation or target scaling practices. Significant instability can be observed in training due to differing distribution in magnitudes of the target values (ir) and input features. Mean Squared Error (MES) is the loss function implemented since it penalizes large deviations severely. Moreover, it is sensitive to unnormalized data which assists in producing observable results.

Model training has been conducted for 100 epochs with disabled early stopping. A static learning (lr = 0.001) has been employed in the absence of an adaptive scheduling. Mini-batch stochastic gradient descent via Adam optimiser is used for training the model. Nonetheless, convergence is hindered due to lack of normalisation and high variance in unscaled financial data.

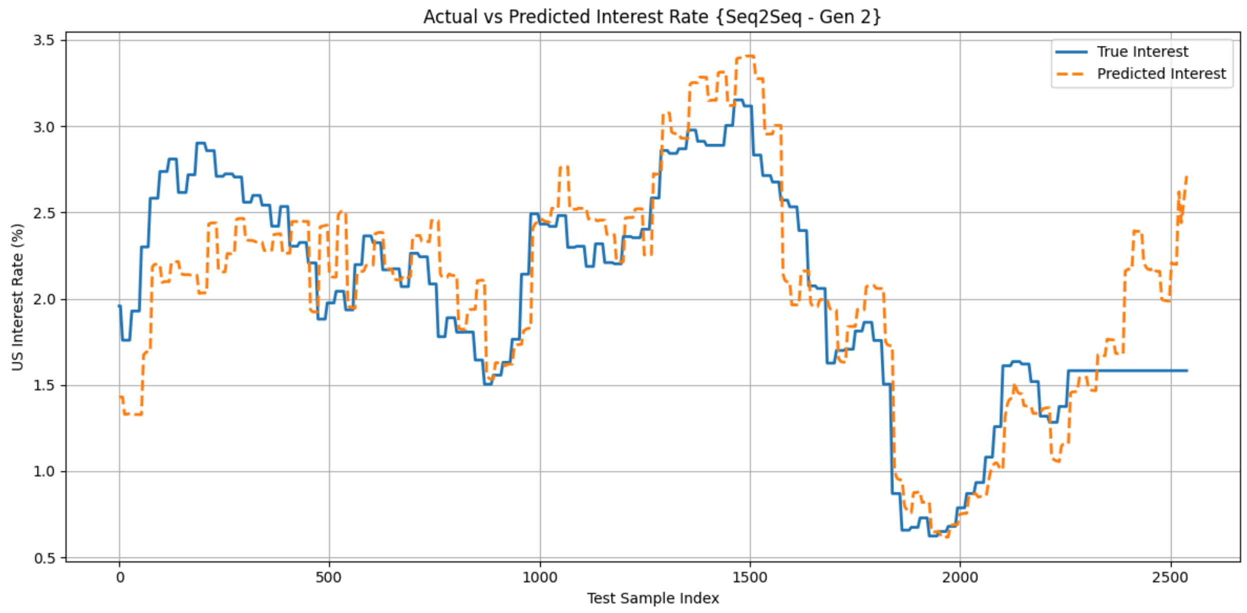


Output of First Generation

As illustrated, the model remains unsuccessful in capturing insightful temporal dependencies. Its predictions are confined between 0.45% and 1.25% while the actual interest rate values range from 0.1% to 3.1%. This compression is a reflection of the model's inefficiency to learn insightful temporal dependencies. In the absence of scaling, large disparities in input magnitude are impairing gradient flow. It is responsible for creating vanishing gradients of concern in the model. The model struggled to discern any significant trend despite 100 epochs of training with MSELoss along with a fixed learning rate of 0.001. This generation highlights the significance of proper preprocessing for facilitating stable optimisation.

Second Generation: This generation introduces preprocessing using RobustScaler from scikit-learn for addressing the gradient scaling issue evident in the first generation. This scaler is quite resilient to outliers since it normalises the target according to the interquartile range (IQR). Crucially, distinct instances of RobustScaler are employed for the target (interest rate) and input features for preserving the ability to inverse-transform. This ensures a meaningful training assessment. The model architecture, optimizer, and loss function remains unaltered.

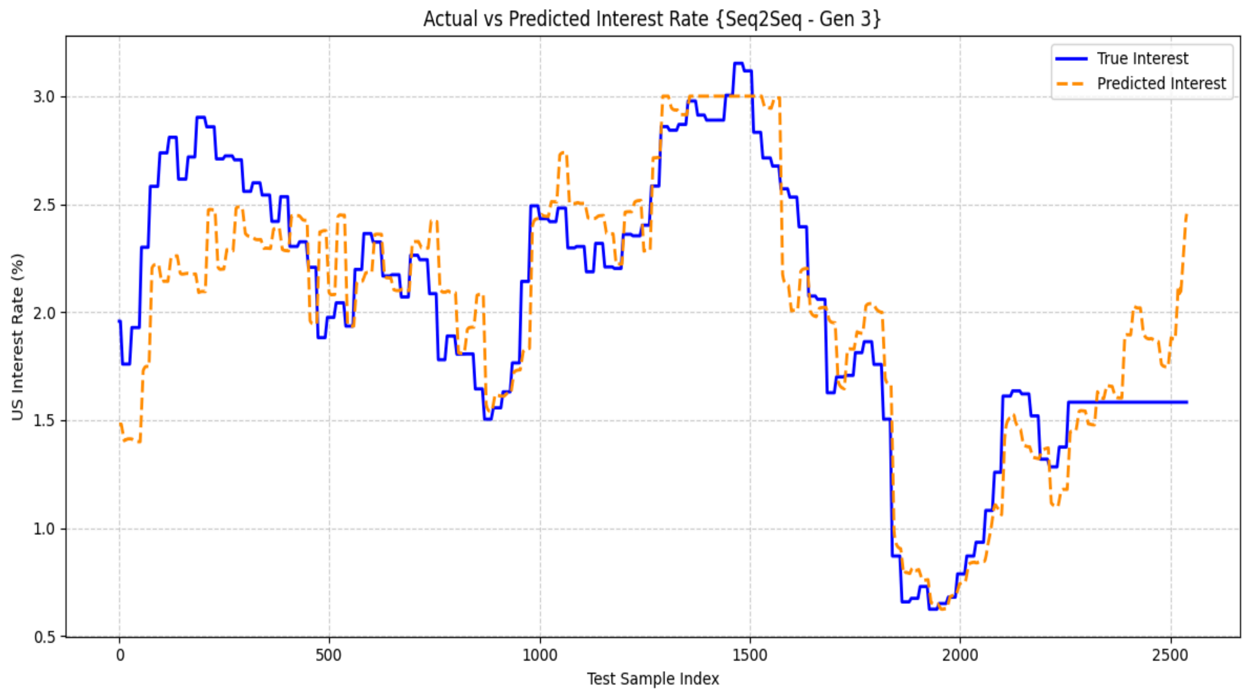
Numerical stability and gradient behavior during backpropagation are greatly enhanced due to normalisation. It consists of Adam optimiser with a learning rate of 0.001 analogous to the previous iteration. The training loss decreased significantly which was essential for generating a more consistent convergence.



Output of Second Generation

After implementation of proper normalization using RobustScaler in addition to inverse transformation, the model's predictions exhibit strong alignment with actual interest rate trajectory. It nearly tracks both long-term trends and local variations. The predicted output range spans from 0.6% to 3.4%. It reflects realistic magnitudes along with improvement in scaling in comparison to generation 1. The model effectively captures directional momentum despite evident slight overshoot. Furthermore, it avoids collapse which is an indication of improvement in gradient flow dynamics enabling the model to learn more generalizable temporal representations.

Third Generation: Two major refinements were incorporated for improving model generalisation while optimizing error. Firstly, change was made from MSELoss to L1Loss (Mean Absolute Error) for reducing sensitivity to outliers apparent in financial time series data which is reliant on abrupt shifts in market sentiment and economic policies. Secondly, the dropout regularization was decreased from 0.3 to 0.1 for enabling the LSTM layers to retain greater long-range temporal dependencies. This initiative addresses the excessive information loss prevalent in previous generations. The learning rate was maintained at 0.001 while a ReduceLROnPlateau scheduler was implemented for adaptive decrease in the learning rate. Thus, this scheduler assisted the optimizer in converging more effectively in the later training epochs.

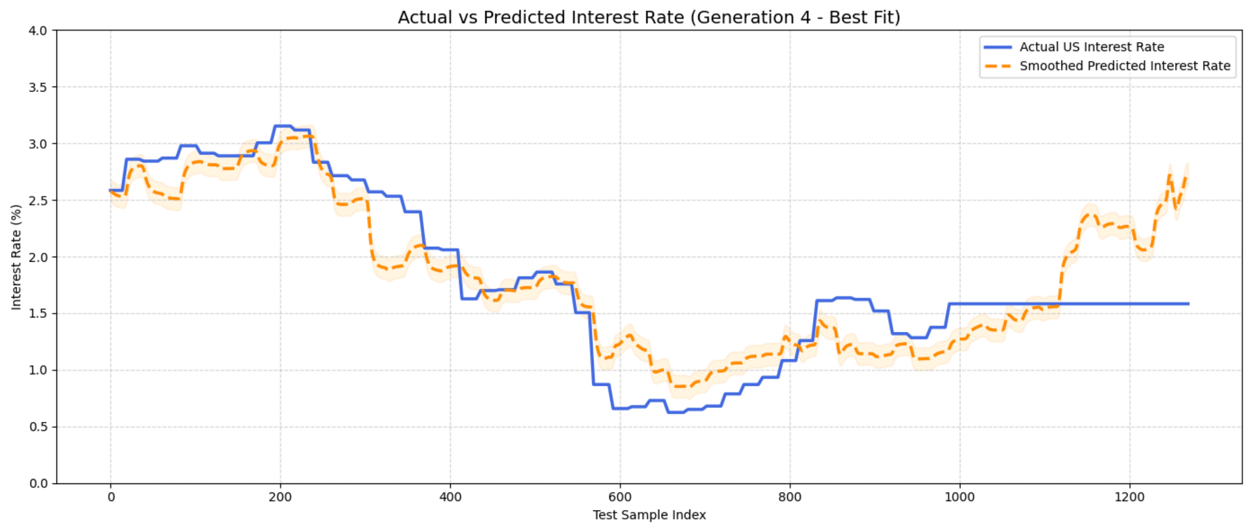


Output of Third Generation

This model outperforms generation 2 in terms of alignment and shape accuracy. Moreover, it exhibits a noticeable better capacity in tracking dynamic interest rate swings. The predictions now capture the curvature of upward and downward trends more accurately especially around inflection points. Due to centered smoothing, the prediction range remains well-calibrated (~0.6% to 3.4%). The model avoids collapsing to local means as well as revealed robust learning behavior across volatile segments.

These enhancements affirm the intuition of improved gradient behaviour combined with more stable sequence modeling.

Fourth Generation: Additional filtrations were conducted for addressing instability observed in prior models. The dropout rate was maintained at 0.1 to preserve temporal memory while the core architecture remained unaltered. The training runs on 10 epochs using the MSELoss function for a faster convergence.In addition, ReduceLROnPlateau scheduler was employed for dynamic adjustment of the learning rate depending on training loss trends. Significant improvement was observed from post-processing enhancements where Gausian-weighted moving average was implemented. It ensured the predicted sequence to have a smooth sequence. Furthermore, clipping ensured the interest rate remains in a realistic range. Therefore, numerical stability of model performance was ensured by abiding by these modifications.



Output of Fourth Generation

The model exhibits a stable forecast profile. The predictions capture the fine tune local trends. The predicted trajectory remains tightly aligned to that of the ground truth by staying within a range of approximately 0.6% to 3.3%. The inclusion of Gaussian weight distribution along with ±0.1% confidence band amplifies visual clarity while portraying the model's consistency. Finally, the reduced volatility, high temporal coherence, and persistent accurate amplitude tracking gives an indication of generation 4 being the most optimised iteration. It represents a significant improvement over all prior generations of the Seq2Seq model.

Steps to recreate this model:

Model Architecture and Training Steps:

- ➢ Preprocess the dataset using RobustScaler for input features. Make use of a separate RobustScaler for the interest rate (ir) target variable. This ensures inverse transformation of predictions

- ➢ Construct a 30 day input sequences with a 5 day forecast target sequence using a sliding window approach across the time series dataset

- ➢ Define a custom Seq2Seq model comprising:

    - ○ An LSTM encoder and decoder (2 layers each)
    - ○ Hidden size of 128 units
    - ○ Dropout set to 0.1 for ensuring regularization with temporal data retention

- ➢ Using the Adam optimizer with a learning rate of 0.001, train the model for 10 epochs

- ➢ Make use of MSELoss (Mean Squared Error) as the loss function for emphasizing large errors

- ➢ Apply the ReduceLROnPlateau scheduler for prevent exploding gradient issue

- ➢ After training to observe the output:
    - ○ Average the predicted 5-day output
    - ○ Use the target scaler for inverse-transforming the predictions

<u>Steps to recreate Generation 4 Figure:</u>

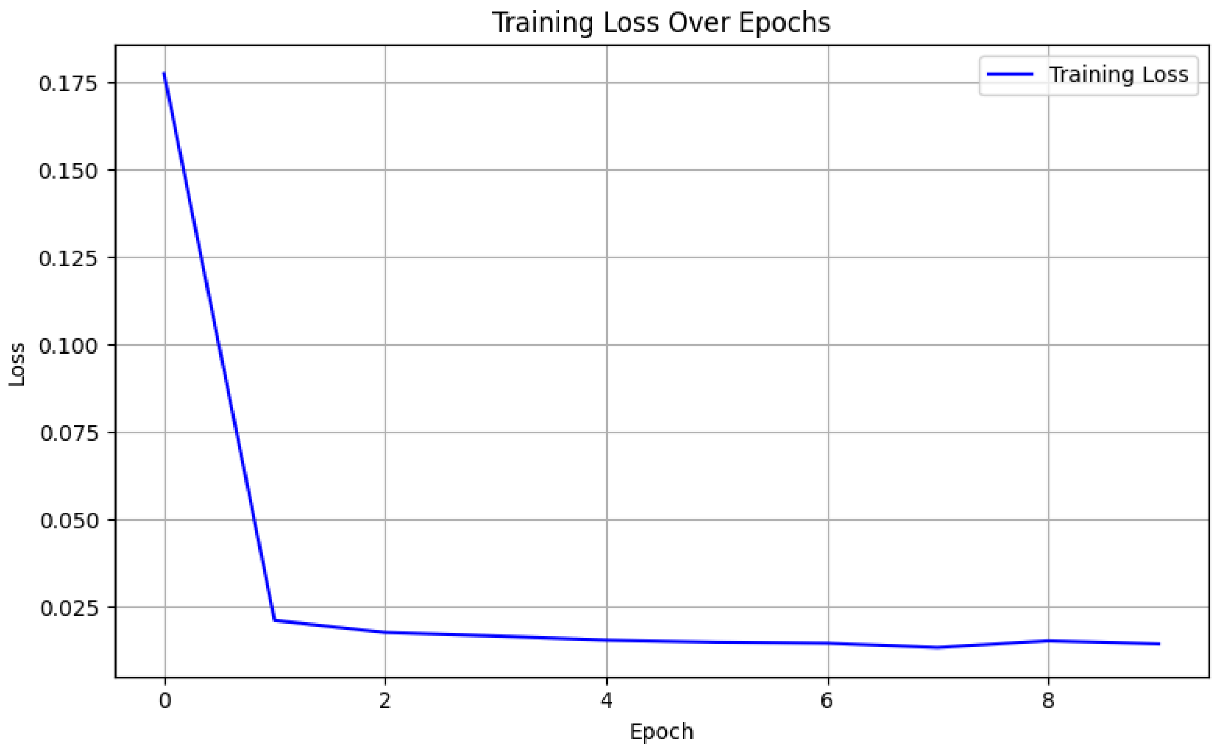
Figure Reproducibility Steps:

- ➢ Apply a centered Gaussian-weighted rolling average for making the predictions distribution smooth
    - ○ E.g. code: smoothed_pred = pd.Series(pred_ir).rolling(window=9, win_type='gaussian', center=True, min_periods=1).mean(std=1.5)

- ➢ Plot the Actual vs. Predicted Interest Rates:

    - ○ Make use of plt.plot for creating the lines
    - ○ Use plt.fill_between for creating a ±0.1% shaded confidence region
    - ○ Match plot settings as derided (e.g. color, linewidth, axis labels, legend, etc.)

- ➢ This figure is fully reproducible using the code provided in the notebook and adheres to the desired outputs for evaluation



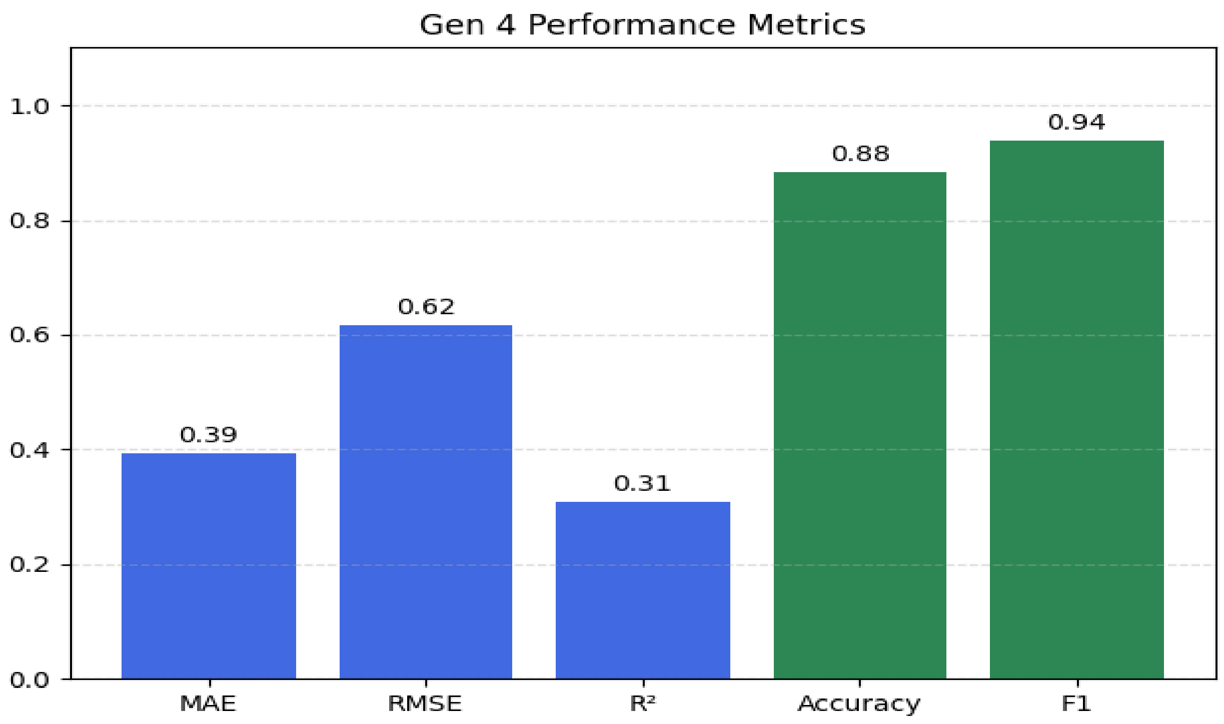
Output of Loss Curve for Fourth Generation

<u>Figure Interpretation:</u>

- ➢ The loss drops steeply after the first epoch. It is immediately followed by gradual convergence (an indication of stable learning behavior)
- ➢ This visualisation is a reflection of effective early optimization using the Adam optimizer with MSELoss

<u>Loss Curve Figure Reproducibility Steps:</u>

- ➢ Use a train_losses empty list to append the average loss at the end of each epoch
- ➢ After training, run the plotting code to generate the figure
- ➢ Save the figure as trainig_loss_curve.png for further evaluation



Seq2Seq Generation 4 Metrics

<u>Figure Interpretation:</u>

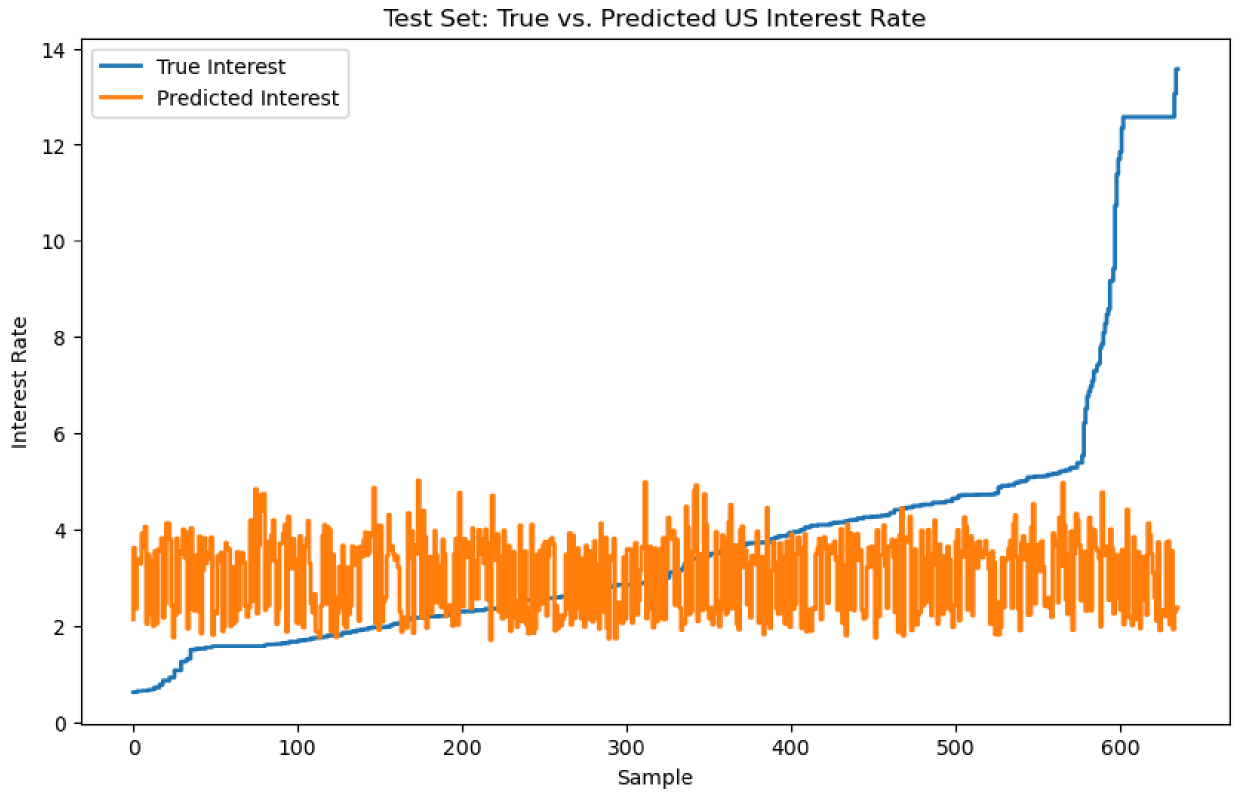
- ➢ MAE (0.39) and RMSE (0.62) give an indication of modest prediction errors
- ➢ Accuracy (0.88) and F1 Score (0.94) is a reflection of a reliable model with a good performance on temporal dataset

<u>Generation 4 Performance Metrics Figure Reproducibility Steps:</u>

- ➢ First run model evaluation to get predicted vs. true interest rates
- ➢ Next compute MAE, R^2, F1 Score, and Accuracy values
- ➢ Prepare metric labels and values
- ➢ Plot them using a bar chart from the matplotlib package
- ➢ Add value labels then set plot title and y-axis range for observability

GRU

No. Layers = 4
Neurons per layer = 1024



MAE Loss = 0.388344
The GRU underperformed and seemed to guess randomly, even with RobustScaler active. A possible reason for this underperformance could be due to the fact that, out of the 3 models, GRUs are the least competent in terms of capturing complex patterns in time series.

Analysis

Analysis of Results

Analysis on Final Model

After evaluating all models and their loss, it seems like LSTM is the best model type, specifically LSTM of the sixth generation. This model had the best ability to capture the overarching trend and the individual interest rate at any given point in time. This is reflected by the low loss value of the final model.

However, it is not without its drawbacks. Since the model takes the previous 30 data points as input, the model can be quite computationally expensive and this window size was not optimized to reduce computational cost - so it is possible that a slightly less expensive model with passable performance exists.

Analysis on Changes

Considering all the models and each model's generations, the most effective change applied to each is the application of RobustScaler. The reason for its effectiveness seems to stay the same for each model; the dataset has a wide range of values and many significant figures. This aligns with what we know about wide ranges and its effect on gradient descent. [4]

A lower learning rate also seemed to have a significant impact on each model. This is seen by the implementation of ReduceLROnPlateau scheduler on the third generation of Seq2Seq stabilizing the outputs. However, having a low learning rate also has its drawbacks. Such is the case with the fifth generation of LSTM, taking a total of 427 epochs to converge after its learning rate was drastically decreased on the third generation.

Limitations

Limitations on Resources

Compared to the researchers and companies designing current state-of-the-art models, we had very minimal resources to process data and train models.

Limitations on Approach

Since each model type was explored individually, without reference to other models, the final model chosen might be the most optimized version with reference to itself, but not the most optimized version with reference to all other models.

The test set was also different for all model types. Although it would still communicate the effectiveness of each model, using the same test set for all models would have allowed for clearer cross-model comparison and better presentation of results.

Not every possible change was explored for each model. (e.g. none of the models explored adding an extra layer) This saves time but, forgoes the benefit of exploring every possible option.

Conclusion

After investigating 3 types of models; LSTM, Seq2Seq and GRU - the best type of model for the task of predicting U.S. interest rate is the LSTM. With proper hyperparameter tweaking and support from RobustScaler, the LSTM model was able to achieve the lowest loss value of 0.0355 in absolute error.

References

[1] Andreini, Paolo, et al. “Deep Dynamic Factor Models.” *arXiv.Org*, Cosimo Izzo, 20 May 2023, arxiv.org/abs/2007.11887.

[2] *Bankofengland*, www.bankofengland.co.uk/-/media/boe/files/events/2020/november/modelling-with-big-data-event-livia-paranhos-presentation.pdf. Accessed 19 Apr. 2025.

[3] Communications, Author BSE. “Deep Vector Autoregression for Macroeconomic Data.” *BSE Voice*, 10 Sept. 2021, thevoice.bse.eu/2021/09/16/deep-vector-autoregression-for-macroeconomic-data/.

[4] Ioffe, S., Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv preprint arXiv:1502.03167.