SC4001 Group Project

Andreas Persson
apers@chalmers.se
N2304696K@e.ntu.edu.sg

April 2024

Contents

1 Introduction **1**

2 Literature **2**

2.1 Traditional Approaches to Stock Prediction 2

2.2 Neural Networks in Finance 2

3 Model Specifications **3**

3.1 Overview 3

3.2 Data & Feature Selection 3

3.3 Models 4

3.3.1 DNN 5

3.3.2 CNN 5

3.3.3 LSTM 6

3.3.4 GRU 6

3.3.5 Transformer 6

3.4 Training & Evaluation 7

4 Results & Analysis **8**

4.1 Performance Overview 8

4.2 Discussion 9

5 Conclusion **10**

6 References **i**

Appendix **iii**

1 Introduction

The financial marketplace embodies a paradigm of complexity and inherent unpredictability, affected by a large amount of variables that influence the pricing of financial instruments. The hard challenge of forecasting future prices is widely acknowledged within the field. Theoretically, the Efficient Market Hypothesis posits the notion of predictive impossibility, suggesting that current asset prices fully reflect all available information (Fama, 1970). However, empirical practices have identified anomalies and patterns, which one can find and use to outperform the market.

The domain of stock prediction has historically been anchored in two predominant methodologies: fundamental analysis and technical analysis. Both strategies require human intervention for the identification and interpretation of potential market indicators and patterns. This reliance introduces a significant constraint, as it is the capacity of the analyst to interpret these trends that ultimately circumscribes the efficacy of the predictive model.

The global market for trading financial instruments has reached an impressive volume, making it an attractive arena for devising alternative strategies to gain a competitive edge. A crucial component of these strategies is accurate price predictions, which is where machine learning, and more specifically neural networks (NNs), come into play. NNs have the potential to predict prices more accurately, thereby reducing the risk associated with the strategies and investments made by traders and investors.

Neural networks has revolutionized machine learning mainly because of the rise of more efficient computing resources and data availability. Their ability to model complex nonlinear relationships and learn from large amount of data makes them a great candidate to model the complex nature of stock prices. I recent years many different types of architectures has been proposed which has increased the performance of neural networks. In this paper we will take a closer look at Convolutional Neural Networks (CNN), Long short-term memory (LSTM), Gated Recurrent Unit (GRU) and Transformers.

The aim is to compare how different neural network architectures affect the accuracy of price predictions for the S&P500 index. The S&P500 index is the market-capitalization-weighted index for the 500 largest companies in USA and serves as a great indication of the performance of the stock market in general. The data is sequential by nature which needs to be considered when building the models.

2 Literature

In this section, we review existing literature pertinent to the use of neural networks for stock price prediction. This includes a discussion on different neural network architectures and their applications in financial markets, as well as an examination of traditional approaches to stock prediction.

2.1 Traditional Approaches to Stock Prediction

As mentioned in the introduction, traditional approaches to stock market forecasting are divided into two primary methods: fundamental analysis and technical analysis. In "A Random Walk Down Wall Street", Malkiel (2003) discusses how both fundamental and technical analysis have been used by investors trying to outperform the market, despite the prevalent belief in the market's efficiency introduced by Fama (1970).

Over time, however, the financial sector has increasingly leaned towards mathematical models to discover new approaches for predicting stock prices. There have been and continue to be numerous different strategies for modeling stock prices. For instance, the Black-Scholes model has been used for pricing options, which is based on the underlying assumption that prices follow a geometric Brownian motion (Black and Scholes, 1973). Furthermore, more modern models like ARIMA and GARCH have been utilized to model stock prices in later years (Ogneva and Golembiovskii, 2018).

New approaches which leans more towards the machine learning domain has also been proposed. This includes the use of Support Vector Machines (SVM) (Lin et al., 2013), Decision Trees and Random Forests (Luckyson et al., 2016), Reinforcement Learning (Meng and Khushi, 2019) and gradient boosting algorithms (Saha et al., 2016) just to mention a few.

2.2 Neural Networks in Finance

With the revolution that neural networks have brought to AI and machine learning, a significant number of articles have been published on how deep learning can be applied to stock market prediction. There are many different approaches to solving these problems; some utilize simple neural networks such as MLPs (Tüfekci, 2016), while others employ attention-based models like Transformers (Wang et al., 2022).

Studies comparing the performance of simple MLPs with RNNs such as LSTMs have been conducted (Khare et al., 2017; Naeini et al., 2010). Both studies concluded that MLP models showed greater potential than the RNNs used. The potential of attention-based neural networks was compared against MLPs, CNNs, and RNNs by Gao et al. (2020), where the results indicated substantial potential for the former method.

3 Model Specifications

3.1 Overview

The study aims to compare four distinct models: CNN, LSTM, GRU, and Transformers. Additionally, it will employ a foundational model—a straightforward feed-forward deep neural network (DNN)—as a benchmark for this comparison. The rationale for selecting this foundational model lies in its simplicity and general limitations for time series prediction, providing a baseline against which to measure the more specialized capabilities of the other models.

DNNs consist of multiple layers of neurons, each layer connected to the next, forming a deep architecture. This structure allows for the learning of complex and high-level features from input data through a process of weighted inputs and nonlinear activations.

CNNs are specialized for processing structured grid data, generally images. Their architecture includes convolutional layers that apply filters to the input, pooling layers that reduce dimensions, and fully connected layers that make predictions.

LSTMs are an advanced type of RNN and are designed to avoid the long-term dependency problem, enabling them to remember information for extended periods. Their architecture includes memory cells that store state over time, and gates that control the flow of information in and out of the cell.

GRUs simplify the LSTM model by combining several gates into two, reducing complexity while maintaining performance. The architecture of a GRU includes an update gate, which decides how much past information to keep, and a reset gate, which determines how to combine the new input with the past memory.

The transformer architecture departs from recurrent layers, utilizing instead a self-attention mechanism to process sequences of data. This allows each position in the sequence to directly attend to every other position, enabling the model to capture complex interdependencies. This is then combined with position encoding and a stack of encoder and decoder layers.

3.2 Data & Feature Selection

The raw data consists of historical stock market data for the S&P 500 ETF. The data was obtained from Yahoo Finance using the yfinance python library, which allows for the download of stock price information over the full extent of its availability, in this case from 1993-01-29 until 2024-04-12. The data contains 7857 rows and 6 columns. The columns are Open, High, Low Close, Adjusted Close and Volume for the ETF at that particular day.

From this data, 10 technical indicators were calculated and used as features. These are:

1. Simple Moving Average (SMA) with a 14-day window
2. Exponential Moving Average (EMA) also over a 14-day window
3. Average True Range (ATR) which provides volatility insight over 14 days
4. Average Directional Movement Index (ADX) for measuring trend strength over 14 periods
5. Commodity Channel Index (CCI) over 20 periods to identify new trends or warnings of extreme conditions
6. Rate of Change (ROC) to capture momentum based on closing price over 10 days
7. Relative Strength Index (RSI) to signal overbought or oversold conditions over 14 days
8. Williams %R for momentum over the past 14 days
9. Stochastic Oscillator %K with a 14-day look-back period and smoothing over 3 days
10. Stochastic Oscillator %D with a 14-day look-back period and smoothing over 3 days

After this, the target variable was created by setting the target to the next day's adjusted closing price. Both the creation of features and the target variable resulted in the first 20 and the last row containing NA values, which were dropped along with the columns: Open, High, Low, Close, and Volume. This means that the input contains 11 features, 10 technical indicators plus Adjusted Close, and the output is one continuous variable, the next day's adjusted close. In total, the data thus contains 7,829 data points.

As described in Section 3.4, due to the use of day-forward chaining cross-validation, the data was normalized separately for each fold. This normalization involved scaling the feature values to a range of 0 to 10 using the MinMaxScaler. For the CNN, LSTM, GRU, and Transformer models, a windowing technique was applied to organize the data into 30-day sequences to accommodate models that rely on temporal dependencies.

Thus the input and output for each model looks as following:

3.3 Models

Since the goal of the study is to test how different architectures affect performance in stock price prediction, it is crucial that the models have approximately the same number of trainable parameters. Therefore, it was decided that each model should have around 200,000 trainable parameters, in Table 2 a overview of the size of the models are displayed.

Table 1: Model Input and Output Specifications

Model	**Input Dimension**	**Output Dimension**
DNN	11	1
CNN	30×11	1
LSTM	30×11	1
GRU	30×11	1
Transformer	30×11	1

Table 2: Number of Trainable Parameters

Model	**Trainable Parameters**
DNN	203,521
CNN	179,913
LSTM	179,701
GRU	219,493
Transformer	178,631

3.3.1 DNN

This DNN comprises four dense layers, configured as follows:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_38 (Dense)             (None, 512)               6144

dense_39 (Dense)             (None, 256)               131328

dense_40 (Dense)             (None, 256)               65792

dense_41 (Dense)             (None, 1)                 257

=================================================================
```

Figure 1: DNN summary

3.3.2 CNN

The CNN architecture consists of the following layers:

Layer (type)	Output Shape	Param #
conv1d_14 (Conv1D)	(None, 28, 128)	4352
max_pooling1d_8 (MaxPooling1D)	(None, 14, 128)	0
conv1d_15 (Conv1D)	(None, 12, 128)	49280
max_pooling1d_9 (MaxPooling1D)	(None, 6, 128)	0
flatten_4 (Flatten)	(None, 768)	0
dense_62 (Dense)	(None, 164)	126116
dense_63 (Dense)	(None, 1)	165

Figure 2: CNN summary

3.3.3 LSTM

This LSTM architecture consists of the following components:

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 200)	169600
dense_74 (Dense)	(None, 50)	10050
dense_75 (Dense)	(None, 1)	51

Figure 3: LSTM summary

3.3.4 GRU

The GRU model is composed as follows:

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 256)	206592
dense_86 (Dense)	(None, 50)	12850
dense_87 (Dense)	(None, 1)	51

Figure 4: GRU summary

3.3.5 Transformer

The architecture of the Transformer model is composed as follows:

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 30, 11)]	0	[]
layer_normalization_6 (LayerNormalization)	(None, 30, 11)	22	['input_2[0][0]']
multi_head_attention_3 (MultiHeadAttention)	(None, 30, 11)	48139	['layer_normalization_6[0][0]', 'layer_normalization_6[0][0]']
dropout_7 (Dropout)	(None, 30, 11)	0	['multi_head_attention_3[0][0]']
tf.__operators__.add_6 (TFOpLambda)	(None, 30, 11)	0	['dropout_7[0][0]', 'input_2[0][0]']
layer_normalization_7 (LayerNormalization)	(None, 30, 11)	22	['tf.__operators__.add_6[0][0]']
conv1d_26 (Conv1D)	(None, 30, 256)	3072	['layer_normalization_7[0][0]']
dropout_8 (Dropout)	(None, 30, 256)	0	['conv1d_26[0][0]']
conv1d_27 (Conv1D)	(None, 30, 11)	2827	['dropout_8[0][0]']
tf.__operators__.add_7 (TFOpLambda)	(None, 30, 11)	0	['conv1d_27[0][0]', 'tf.__operators__.add_6[0][0]']
layer_normalization_8 (LayerNormalization)	(None, 30, 11)	22	['tf.__operators__.add_7[0][0]']
multi_head_attention_4 (MultiHeadAttention)	(None, 30, 11)	48139	['layer_normalization_8[0][0]', 'layer_normalization_8[0][0]']
dropout_9 (Dropout)	(None, 30, 11)	0	['multi_head_attention_4[0][0]']
tf.__operators__.add_8 (TFOpLambda)	(None, 30, 11)	0	['dropout_9[0][0]', 'tf.__operators__.add_7[0][0]']
layer_normalization_9 (LayerNormalization)	(None, 30, 11)	22	['tf.__operators__.add_8[0][0]']

Figure 5: First part of the Transformer model summary

conv1d_28 (Conv1D)	(None, 30, 256)	3072	['layer_normalization_9[0][0]']
dropout_10 (Dropout)	(None, 30, 256)	0	['conv1d_28[0][0]']
conv1d_29 (Conv1D)	(None, 30, 11)	2827	['dropout_10[0][0]']
tf.__operators__.add_9 (TFOpLambda)	(None, 30, 11)	0	['conv1d_29[0][0]', 'tf.__operators__.add_8[0][0]']
layer_normalization_10 (LayerNormalization)	(None, 30, 11)	22	['tf.__operators__.add_9[0][0]']
multi_head_attention_5 (MultiHeadAttention)	(None, 30, 11)	48139	['layer_normalization_10[0][0]', 'layer_normalization_10[0][0]']
dropout_11 (Dropout)	(None, 30, 11)	0	['multi_head_attention_5[0][0]']
tf.__operators__.add_10 (TFOpLambda)	(None, 30, 11)	0	['dropout_11[0][0]', 'tf.__operators__.add_9[0][0]']
layer_normalization_11 (LayerNormalization)	(None, 30, 11)	22	['tf.__operators__.add_10[0][0]']
conv1d_30 (Conv1D)	(None, 30, 256)	3072	['layer_normalization_11[0][0]']
dropout_12 (Dropout)	(None, 30, 256)	0	['conv1d_30[0][0]']
conv1d_31 (Conv1D)	(None, 30, 11)	2827	['dropout_12[0][0]']
tf.__operators__.add_11 (TFOpLambda)	(None, 30, 11)	0	['conv1d_31[0][0]', 'tf.__operators__.add_10[0][0]']
global_average_pooling1d_1 (GlobalAveragePooling1D)	(None, 30)	0	['tf.__operators__.add_11[0][0]']
dense_98 (Dense)	(None, 512)	15872	['global_average_pooling1d_1[0][0]']
dropout_13 (Dropout)	(None, 512)	0	['dense_98[0][0]']
dense_99 (Dense)	(None, 1)	513	['dropout_13[0][0]']

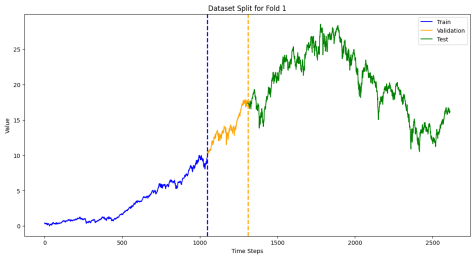
Figure 6: Second part of the Transformer model summary

3.4 Training & Evaluation

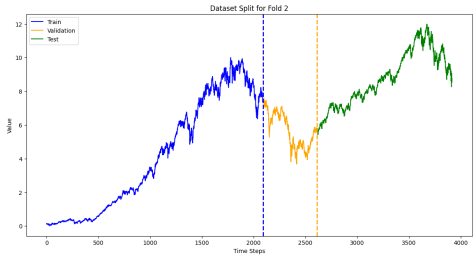
For the evaluation, a Cross-Validation technique known as Day Forward-Chaining was employed. This approach was selected due to its relevance in assessing model performance across varying data volumes and in preventing information leakage in time-series data. Figure 7 illustrates the five folds utilized in the process. Each fold is segmented into training and testing subsets using the TimeSeriesSplit function in sklearn. Furthermore, the training subset is further split into training and evaluation segments with the ratio 80% training and 20% validation.

All models used Mean Squared Error (MSE) loss function and the ADAM-optimizer with an initial learning rate of 0.001. The maximum number of epochs during training was set to 200 and early stopping with a patience of 5 was employed.

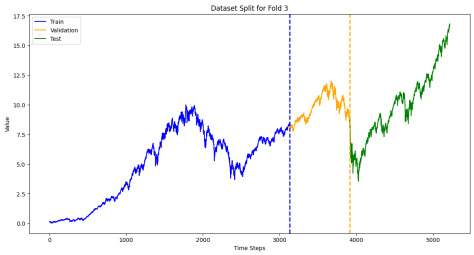
The evaluating metrics that was used were Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error. The reason for choosing these are that they ideal for stock price prediction due to their specific strengths: RMSE effectively highlights the average magnitude of prediction errors, penalizing larger mistakes, which is critical in financial decision-making. MAPE, on the other hand, offers a clear, scale-independent percentage-based measure of accuracy. Together, they provide a comprehensive assessment of model performance in forecasting stock prices.



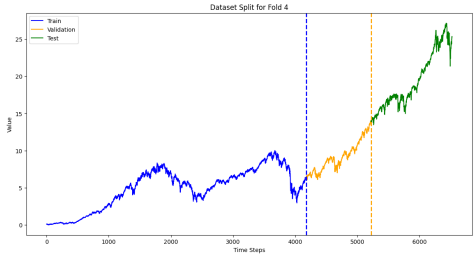
(a) Fold 1



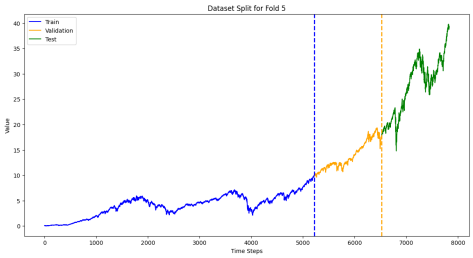
(b) Fold 2



(c) Fold 3



(d) Fold 4



(e) Fold 5

Figure 7: Dataset Split for Folds 1 to 5

4 Results & Analysis

4.1 Performance Overview

Table 3: Mean Absolute Percentage Error (MAPE) across different folds

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AVG
DNN	0.0599	0.0954	0.0575	0.0269	0.0170	0.05134
CNN	0.1043	0.1125	0.1139	0.0658	0.0333	0.08596
LSTM	0.1222	0.1167	0.1071	0.0928	0.2128	0.13032
GRU	0.1070	0.1022	0.0978	0.1193	0.1409	0.11344
Transformer	0.2089	0.1042	0.0746	0.2177	0.2685	0.17478

Table 4: Root Mean Square Error (RMSE) across different folds

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	AVG
DNN	1.3484	0.8948	0.6556	0.6093	0.6482	0.83126
CNN	2.3859	1.0873	1.3314	1.4536	1.1152	1.47468
LSTM	2.9289	1.1287	1.3032	2.8102	9.0295	3.44010
GRU	2.5471	0.9590	1.1330	2.7296	4.3395	2.34164
Transformer	5.1096	0.9815	1.1493	4.8955	8.5890	4.14498

Table 5: Average fold training time for each model

Model	Time
DNN	6.37s
CNN	9.47s
LSTM	55.27s
GRU	48.21s
Transformer	28.31s

4.2 Discussion

The result is somewhat surprising, as none of the models under study outperformed the baseline model on any of the metrics used. The observation appears to be that the more complex the architecture, the poorer the performance. One possible explanation is that these models excel when they have access to significantly larger datasets. For instance, it is known that Transformers perform optimally when there is an abundance of data available. While the stock market has a long history and considerable data are available, the volume is still minimal compared to other domains where such models are applied. Additionally, the stock market has inherent fluctuations over time, meaning that data from the 1990s may not be comparable to the current market conditions. Therefore, it is crucial for a model designed for stock market prediction to perform well on a small and constantly changing dataset.

All the models studied tend to consistently underpredict the price, which is odd. As shown in Table 3 and 4, the CNN performs the best among the models, which is also evident in the actual vs. predicted plots (see Appendix). However, what's peculiar is the smoothness of the curve for fold 5 (see Appendix), which is atypical for time-series data such as this. Additionally, the GRU also performs fairly well, and the graphs look very similar except for the underprediction occurring. The model that performs the worst is the Transformer model, according to both MAPE and RMSE. Yet, the graphs indicate that

the LSTM model is much more unstable, particularly evident in folds 1 and 5 (see Appendix).

In summary, all models evaluated in the study perform rather poorly, with the best model averaging a mean percentage error of 8.5%, a rate that would not be viable in practical applications. While the best-performing CNN model shows some promise, I believe that the GRU model has the potential to yield the best performance after hyperparameter tuning. This belief comes from its consistency with actual values but only having a pattern of underprediction, which I think could be addressed with certain modifications. It's important to note that no hyperparameter tuning has been conducted on any of the models, a process that would surely enhance their performance and should be considered in future research. Additionally, the author's limited experience with neural networks should be taken into consideration, as there is likely a substantial opportunity for improvement in this study.

With this in mind, it is challenging to draw any significant conclusions from the results of this study, other than it seems difficult to effectively predict stock prices using neural networks. Even though there is theoretical potential for neural networks in this domain, it may still be that other models, particularly those based on stochastic processes, are more suitable for a task like this.

5 Conclusion

In this study, we compared four different models—CNN, LSTM, GRU, and Transformer—with a baseline DNN model.

The results show that none of the four models outperform the baseline model. This could be due to their complexity not being well-suited for stock market prediction, but most likely it is due to the need for more hyperparameter tuning.

Among the four studied models, the CNN performs the best, but there are concerns about the nature of the actual vs. predictions graphs. All models underpredict, and the GRU model is the one that the author believes has the best potential.

6 References

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. Journal of Political Economy, 81(3), 637-654.

Fama, E. F. (1965). The behavior of stock-market prices. Journal of Business, 38(1), 34–105.

Gao, P., Zhang, R., & Yang, X. (2020). The application of stock index price prediction with neural network. Mathematical and Computational Applications, 25(3), 53. https://doi.org/10.3390/mca25030053

Khare, K., Darekar, O., Gupta, P., & Attar, V. Z. (2017). Short term stock price prediction using deep learning. 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 482-486. https://doi.org/10.1109/RTEICT.2017.8256643

Lin, Y., Guo, H., & Hu, J. (2013). An SVM-based approach for stock market trend prediction. The 2013 International Joint Conference on Neural Networks (IJCNN), Dallas, TX, USA, 1-7. https://doi.org/10.1109/IJCNN.2

Malkiel, B. G. (2003). A random walk down Wall Street. W. W. Norton & Company.

Meng, T. L., & Khushi, M. (2019). Reinforcement learning in financial markets. Data, 4(3), 110. https://doi.org/10.3390/data4030110

Naeini, M. P., Taremian, H., & Hashemi, H. B. (2010). Stock market value prediction using neural networks. 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), Krakow, Poland, 132-136. https://doi.org/10.1109/CISIM.2010.5643675

Ogneva, D. S., & Golembiovskii, D. Y. (2018). Option pricing with ARIMA–GARCH models of underlying asset returns. Computational Mathematics & Modeling, 29(4), 461–473. https://doi.org/10.1007/s10598-018-9425-2

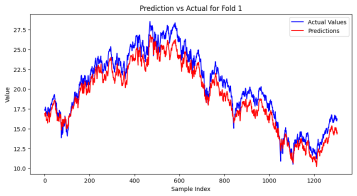
Saha, S., Bhattacharya, S., & Ravi, V. (2016). Forecasting to classification: Predicting the direction of stock market price using Xtreme Gradient Boosting. Retrieved from https://www.researchgate.net/publication/309492895

Tüfekci, P. (2016). Predicting the direction of movement for stock price index using machine learning methods. In A. Abraham, K. Wegrzyn-Wolska, A. Hassanien, V. Snasel, & A. Alimi (Eds.), Proceedings of the Second International Afro-European Conference for Industrial Advancement AECIA 2015. Advances in Intelligent Systems and Computing, vol 427. Springer, Cham. https://doi.org/10.1007/978-3-319-29504-6_45

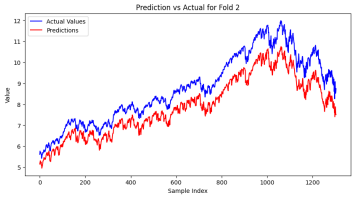
Wang, C., Chen, Y., Zhang, S., & Zhang, Q. (2022). Stock market index prediction using deep Transformer model. Expert Systems with Applications, 208, 118128. https://doi.org/10.1016/j.eswa.2022.118128

Appendix

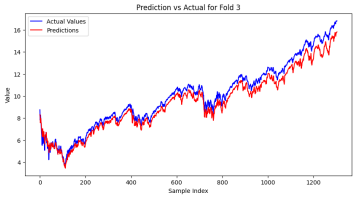
DNN



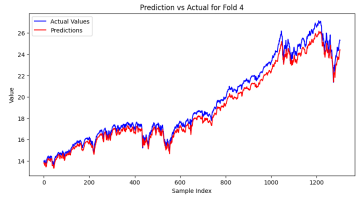
(a) Fold 1



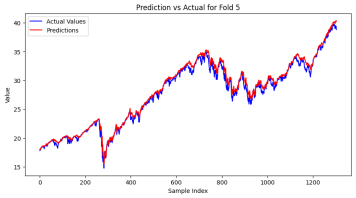
(b) Fold 2



(c) Fold 3



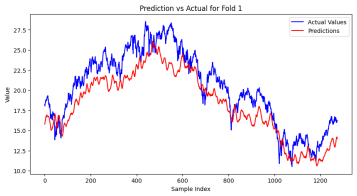
(d) Fold 4



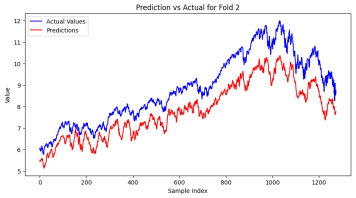
(e) Fold 5

Figure 8: Prediction vs Actual for Folds 1 to 5 for DNN

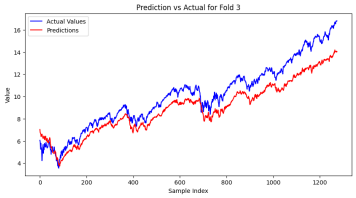
CNN



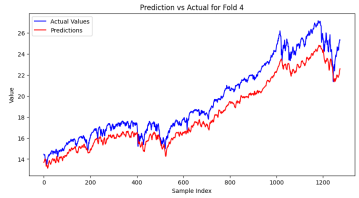
(a) Fold 1



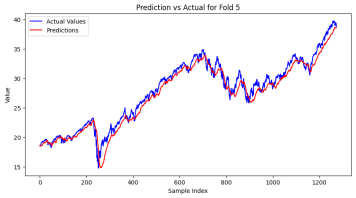
(b) Fold 2



(c) Fold 3



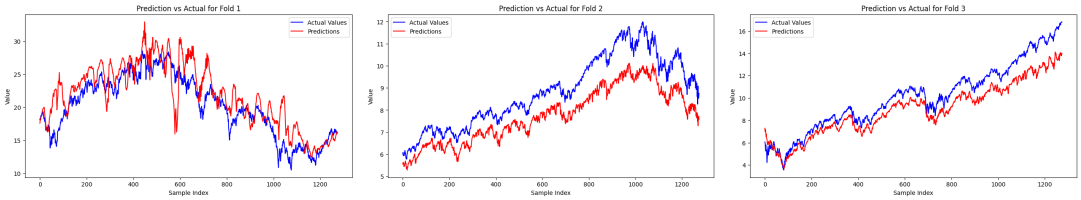
(d) Fold 4



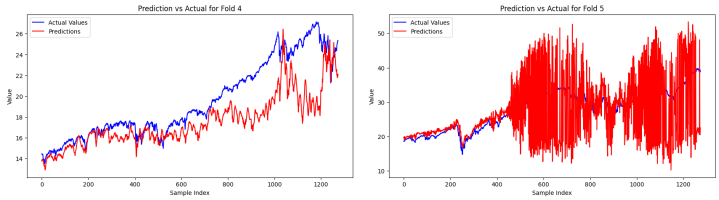
(e) Fold 5

Figure 9: Prediction vs Actual for Folds 1 to 5 for CNN

LSTM



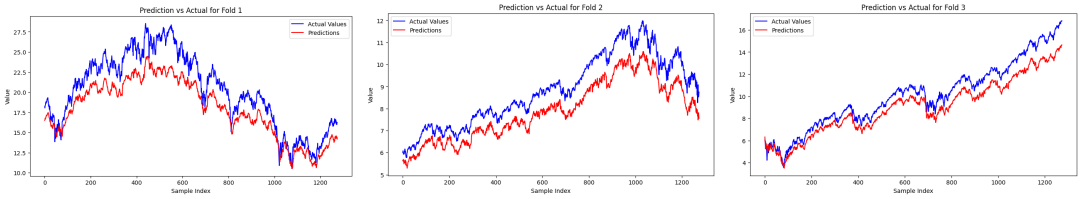
(a) Fold 1 (b) Fold 2 (c) Fold 3



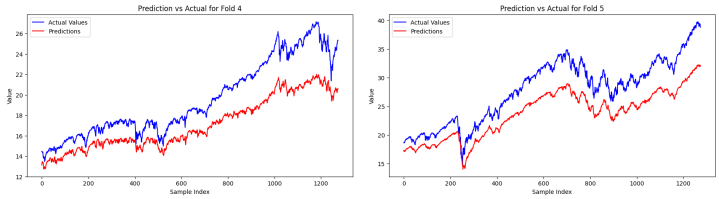
(d) Fold 4 (e) Fold 5

Figure 10: Prediction vs Actual for Folds 1 to 5 for LSTM

GRU



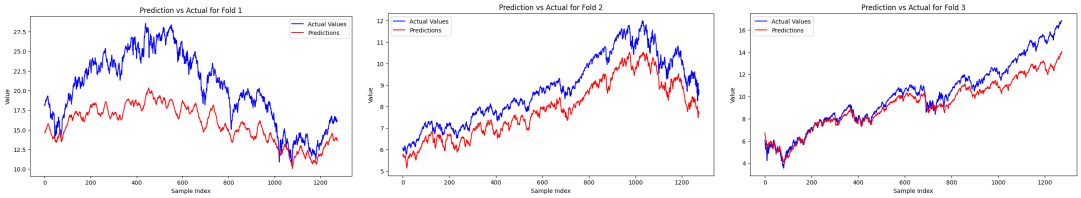
(a) Fold 1 (b) Fold 2 (c) Fold 3



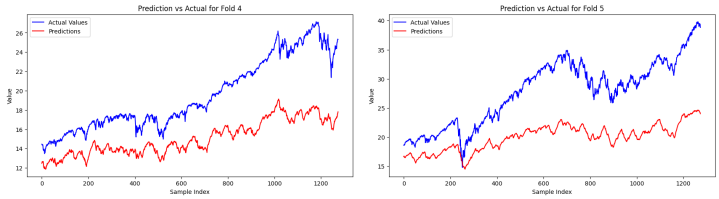
(d) Fold 4 (e) Fold 5

Figure 11: Prediction vs Actual for Folds 1 to 5 for GRU

Transformer



(a) Fold 1 (b) Fold 2 (c) Fold 3



(d) Fold 4 (e) Fold 5

Figure 12: Prediction vs Actual for Folds 1 to 5 for Transformer