Long-Short Term Memory Neural Network for incomplete data inputs[*]

Irfan Mahmood[1,2,*,†]

[1]*Silesian University of Technology, Akademicka 2A, 44-100, Gliwice, Poland*

[2]*University of L'Aquila, Via Vetoio, 40, 67100 Coppito AQ, Italy*

Abstract

In this article, I investigated the performance of various Long Short-Term Memory Neural Network architectures by varying the number of data inputs. LSTMs, a family of recurrent neural networks, are well-suited for tasks such as natural language processing, and time series prediction due to their ability to capture long-term dependencies in sequential data. I conducted extensive experiments using TensorFlow and PyTorch frameworks, analyzing both functional and sequential LSTM models. My analysis includes a variety of datasets with different time periods and data points based on assessing different key metrics. The results indicate significant variability in model performance based on input data volume and LSTM architecture. Key findings show that larger datasets generally improve accuracy and reduce error rates. However, an optimal window size is crucial to balance model complexity and prediction performance. While increasing data inputs enhances accuracy, it also introduces challenges related to overfitting and computational efficiency. This research provides valuable insights into deploying LSTM networks for time series analysis, emphasizing the importance of data quantity and preprocessing techniques. My findings contribute to optimizing LSTM architectures for robust and reliable predictive modelling in various real-world applications.

Keywords

Long Short-Term Memory, Recurrent Neural Network, Time Series Analysis, Sequential Data Modeling

1. Introduction

Long Short-Term Memory Neural Networks have garnered significant interest, particularly in time-series forecasting, due to their ability to capture temporal dependencies and effectively process sequential data. This article delves into different LSTM model configurations, emphasizing the impact of varying input data lengths on their performance. I begin by exploring the architecture and functionality of LSTM models, highlighting their recurrent nature and mechanisms to retain long-term dependencies while circumventing the vanishing gradient problem. The significance of input data length in LSTM modelling is also discussed, focusing on its implications for predictive accuracy. This research addresses key inquiries:

1. How do different LSTM configurations perform with varying input data lengths?

[*] *IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania*

[1,*] Corresponding author

[†] These author contributed equally.

im313818@student.polsl.pl; irfan.mahmood.pol@gmail.com (I. Mahmood)

0009-0000-5149-6403 (I. Mahmood)

2. What metrics are used to evaluate LSTM performance, and how do they vary across input data lengths?

3. What insights can LSTM model performance provide for time-series forecasting, and how can this inform real-world decision-making?

I analyze four case studies involving different LSTM configurations using TensorFlow and PyTorch, assessing performance across metrics like MAE, MSE, RMSE, MAPE, and Accuracy. This comprehensive analysis aims to enhance our understanding of optimal LSTM configurations for time-series forecasting, contributing valuable insights for practitioners and researchers.

2. Time series analysis & LSTM

2.1. Time Series Analysis

Time series analysis is a specialized branch of statistics and data science that deals with data points collected or recorded at successive points in time. It involves methods for analyzing time series data to extract meaningful statistics and identify patterns, trends, and seasonal variations. In this analysis, it is crucial for understanding temporal dependencies and patterns, making it widely used in various fields such as finance, economics, environmental science, medicine, and engineering[1][2]. Mathematically, a time series can be represented as:

$$y_t = T_t + S_t + R_t \tag{1}$$

where y_t is the observed value at time t, T_t is the trend component, representing the long-term progression, S_t is the seasonal component, capturing the repeating short-term cycle, R_t is the residual component, accounting for the random noise or irregularities[3].

A key concept in time series analysis is stationarity. A time series is considered stationary if its statistical properties, such as mean, variance, and autocorrelation, remain constant over time. Many time series models, including ARIMA, assume that the data is stationary. If the data is not stationary, techniques like differencing, transformation, or detrending are applied to achieve stationarity.[4]

A key objective of time series analysis is to model the underlying structure of the data so that future values can be predicted. This often involves decomposing the series into its constituent components: trend, seasonal, and residual (noise). The components of time series are:

Trend: The long-term movement in the data, which can be upward, downward, or constant. Trends show the overall direction in which the data is moving over a long period. **Seasonality:** Regular, repeating patterns or cycles in the data occurring at specific intervals, such as daily, monthly, or yearly. These patterns are often influenced by seasonal factors. **Cyclic Patterns:** Long-term oscillations or cycles that are not of a fixed period but are influenced by economic, environmental, or other factors. These are different from seasonality in that they do not occur at regular intervals. **Irregular or Residual Component:** The random noise or variability in the data cannot be explained by the trend, seasonal, or cyclic components. This is often modelled as a stochastic process.

2.2. Introduction to LSTM

Long-short-term Memory Neural Networks are a kind of recurrent neural network architecture designed to model sequences and time series data effectively. Traditional RNNs struggle to capture long-range dependencies in sequences and solve gradient problems. To solve this problem, Hochreiter and Schmidhuber introduced LSTM neural networks in 1997. The LSTM architecture incorporates specialized memory cells and gating mechanisms that allow it to remember or forget information over time selectively. This enables LSTM networks to retain important information for long periods and effectively learn from sequences with long-term dependencies. The critical components of an LSTM Neural Network:

1. Input Gate: This gate determines which new information to store in the cell state. It updates the cell state using the previous concealed state as well as the current input and output values between 0 and 1.[Eq. 2]

2. Cell State: The cell state runs horizontally through the network and serves as a conveyor belt, allowing information to flow relatively unchanged. It is regulated by various gates to control the flow of information. The candidate values that could be added to the cell state are calculated using the tanh function. The cell state is updated by combining the previous cell state and the candidate cell state.[Eq. 3][Eq. 4]

3. Forget Gate: This gate decides what information to discard from the cell state. It takes the previous hidden state (h_{t-1}) and the current input (x_t) as input and a value between 0 and 1 as output for each element in the cell state, where 0 means forget and 1 means keep.[Eq.5]

4. Output Gate: This gate controls the information flow from the cell state to the output. It decides what information from the cell state to output based on the current input and previous hidden state. The output gate determines what the next hidden state should be.[Eq. 6]

5. Hidden State: The hidden state contains information about the current input as well as past inputs. It is computed using the cell state, the previous concealed state, and the current input.[Eq. 7]

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{4}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{7}$$

where i_t, f_t, o_t are the activation vector for the gate, W is the weight matrix for the gate, b_i, b_C, b_o, b_f are the bias vector for the input, candidate, forget and output gate respectively, $\tilde{C}_t$, C_t, C_{t-1} are the candidate, updated and previous cell state respectively, σ is the sigmoid function, h_{t-1}, h_t are the hidden state from the previous and current time step respectively, x_t is the input at the current time step.[5]

During the training process, the LSTM network learns to adjust the parameters of its gates through backpropagation, optimizing its ability to learn and retain information from sequential data. Because of this, LSTM networks can efficiently model and forecast long-range dependent sequences, which makes them useful for applications like [6] [7] speech recognition, natural language processing, and time series analysis[8].

3. LSTM architectures

Various Long Short-Term Memory Neural Network architectures are used for sequential data analysis and time-series forecasting tasks. While they share a common underlying concept of LSTM networks, they are implemented using different frameworks and may have distinct architectural designs and features. Each framework offers its advantages and nuances regarding usability, flexibility, and performance optimization.

3.1. SEQUENTIAL TENSORFLOW

In TensorFlow, LSTM networks can be implemented in a sequential model using the Keras API. Sequential models are a layer-by-layer stack, making them easy to define and suitable for many deep learning tasks, including sequence modelling with LSTM networks.[Figure 1]

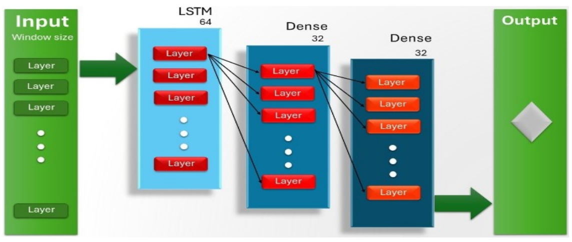

Figure 1: Sequential LSTM model overview

The process begins by creating a Sequential model, which organizes layers linearly, each with a single input and output tensor. To handle sequential data and capture temporal dependencies,

Algorithm 1: LSTM Model Structure

```
Defining the model :
    model = Sequential([layers.Input((3, 1)),
    layers.LSTM(64),
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)])
model.compile(loss='mse',optimizer=Adam(learning rate=0.001),metrics=['mean absolute error'])
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100)
```

the model add one or more LSTM layers to the Sequential model, which enables it to learn patterns and relationships over time. The model's LSTM layers are configured by setting parameters such as the number of units (neurons), activation functions, dropout rates, and whether to return sequences or only the last output, which influences the behaviour and performance of the LSTM network. Then, the model is compiled by specifying the loss function, optimizer, and evaluation metrics. This step sets up the training process, defining how the model learns from the training data. The model is trained using training data and labels with specific epoch numbers and batch sizes. The model's performance is assessed on a validation or test dataset using metrics like accuracy to determine how well the model generalizes to unseen data. Finally, the trained model is used to make predictions on new data, which can be used for inference or decision-making purposes.[Algorithm 1]

3.2. FUNCTIONAL TENSORFLOW

In TensorFlow, LSTM networks can be implemented using the functional API, which offers more flexibility and customization options than the Sequential API. The functional API allows for the creation of complex neural network architectures, including models with multiple inputs or outputs, shared layers, and branching architectures.[Figure 2]

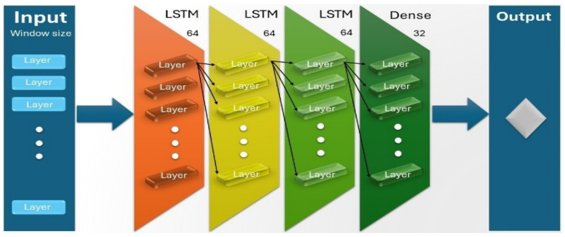

Figure 2: Functional LSTM model overview

The model building starts by defining an input layer to specify the shape of the input data. The

Algorithm 2: LSTM Model Structure

```
def define model():
    input1 = Input(shape=(window size,1))
    x = LSTM(units=64,return sequences=True)(input1)
    x = Dropout(0.2)(x)
    x = LSTM(units=64,return sequences=True)(x)
    x = Dropout(0.2)(x)
    x = LSTM(units = 64)(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='softmax')(x)
    dnn output = Dense(1)(x)
    model = Model(inputs=input1,outputs=[dnn output])
    model.compile(loss='mean squared error',optimizer='Nadam')
    model.summary()
    return model
model = define model()
history = model.fit(X train, y train, epochs=100, batch size=32, validation split=0.1, verbose=1)
```

model adds one or more LSTM layers by connecting the input layer to the LSTM layer(s). These layers handle sequential data and capture long-term dependencies. Then, the model instantiates an object, specifying its inputs and outputs. This step connects the layers defined earlier to construct the overall neural network architecture. The model is compiled by specifying the loss function, optimizer, and evaluation metrics which configures the training process for the model. For overview, this step shows the number of parameters in each layer, which provides insights into the structure of the neural network. The model is trained using training data and labels, specifying the number of epochs and batch size for training. Now, the model's performance is evaluated on a separate validation or test dataset to assess its effectiveness in making predictions and the trained model is used to forecast new data by passing it through the model to obtain predictions or classifications.

3.3. PYTORCH

The PyTorch LSTM Model leverages its dynamic computational graph and flexible design capabilities to construct LSTM-based architectures for sequential data analysis. PyTorch LSTM models can be built layer-by-layer, allowing for various configurations tailored to specific tasks.[Figure 3]

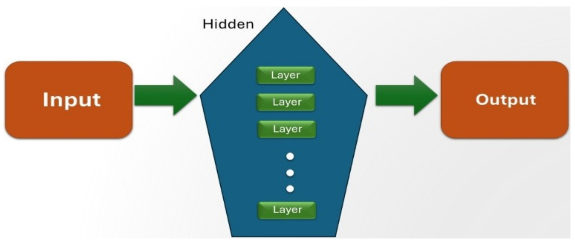

Figure 3: PyTorch LSTM model overview used in this research

Initializing by defining an LSTM model by subclassing the "torch.nn.Module" class. Within

Algorithm 3: Creating LSTM Model

```
class LSTM(nn.Module):
    def init (self, input size, hiddensize, num stacked layers):
        super(). init()
        self.hidden size = hidden size
        self.num stacked layers = num stacked layers
        self.lstm = nn.LSTM(input size, hidden size, num stacked layers, batch first=True)
        self.fc = nn.Linear(hidden size, 1)
    def forward(self, x):
        batch size = x.size(0)
        h0 = torch.zeros(self.num stacked layers, batch size, self.hidden size).to(device)
        c0 = torch.zeros(self.num stacked layers, batch size, self.hidden size).to(device)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out
model = LSTM(1, 4, 1)
model.to(device)
model
```

this model class, define the layers and operations that comprise the LSTM network. The model creates an instance of the LSTM layer using the "torch.nn.LSTM" class. This step specifies the input dimension, hidden dimension, number of layers, and whether the LSTM layer should batch first.[9] The forward method of the model class is overridden to define the forward pass computation. This involves passing input data through the LSTM layer(s) and optionally through additional layers for further processing.[10] So, the model initializes a hidden and a cell state either manually or by defining an initialization method within the model class. Unlike TensorFlow, PyTorch directly defines the loss function, optimizer, and other training configurations during model training.[11] The LSTM model is trained using training data and labels, the training loop is implemented to iterate over batches of data, the loss is computed, and the model parameters are updated using backpropagation and the chosen optimizer.[12] Then, the trained model's performance is evaluated on a separate validation or test dataset and computes evaluation metrics to assess the model's performance.[13] Finally, the trained LSTM model is used to make predictions on new data by passing the data through the model to obtain predictions or classifications.[14]

4. Description and Statistical analysis of the dataset

The dataset comprises over 20 years of data (more than 6000 data points), featuring columns for date, open, high, low, close, adjusted close, and volume. The dataset contains the Stock prices of Microsoft, QCOM and Amazon where the data has been collected from finance.yahoo.com. To analyze the LSTM Neural Network performance, the dataset was transformed to align with the requirements of LSTM modelling. Different subsets of data were extracted for analysis. These subsets enable a comprehensive evaluation of LSTM model performance across varying timeframes, facilitating insights into the effectiveness of the models in capturing and predicting patterns within the given data.

Basic Statistical Analysis: The dataset includes six key financial metrics for a series of stock data over a significant period. The statistical analysis of the dataset reveals significant variability in stock prices and trading volumes. The mean, median, and percentile values show a broad range of stock prices, suggesting periods of both low and high market activity. The wide range between minimum and maximum values, especially in volume, indicates periods of both low and extremely high trading activity. Percentiles provide insight into the distribution of prices and volumes, with medians (50th percentiles) offering a central value for comparison. This comprehensive statistical overview is crucial for understanding the stock's historical performance and volatility.

Correlation Matrix: The correlation matrix provided gives a detailed look at the relationships between various stock price variables and Volume. Each value in the matrix ranges between -1 and 1, indicating the strength and direction of the linear relationship between two variables. The correlation matrix highlights that all price-related metrics are highly positively correlated with each other, with correlation coefficients near 1. This indicates that when one price metric increases or decreases, the others are likely to move in the same direction. In contrast, the trading volume (Volume) shows a moderate negative correlation with all the price metrics, suggesting that higher prices are somewhat associated with lower trading volumes and vice versa.

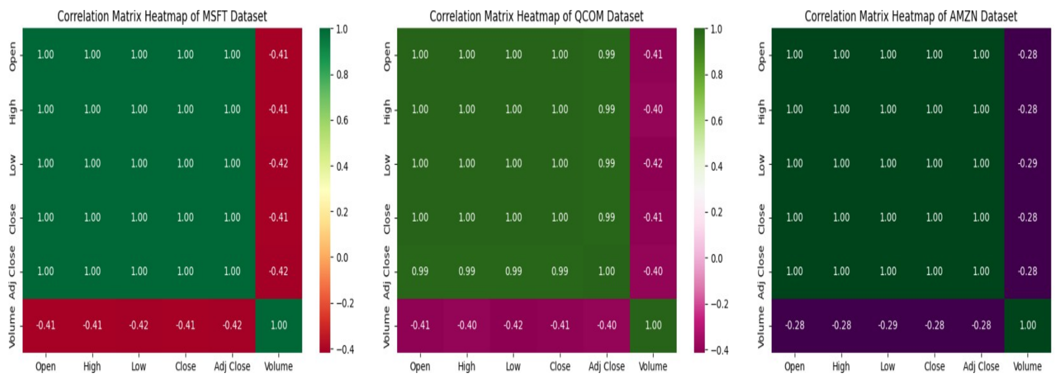

Figure 4: Correlation matrix shows strong positive correlation among the price variables and moderate negative correlation between the volume and the price variables.

This information is critical for understanding the interdependencies between different finan- cial metrics and can be used for predictive modelling and strategic decision-making in financial analysis.[Figure 4]

5. Performance analysis

5.1. Sequential LSTM(single training run)

This analysis evaluates the performance of Sequential LSTM models using TensorFlow across varying data input lengths and window sizes for different time spans of datasets. The data covers datasets ranging from 1 year to over 20 years, with varying window sizes (3, 5, 7, 10, 15, 30). For shorter time series (1-5 years), models show higher MAE and MSE with small window sizes. For example, the 1-year dataset with a window size of 5 had an MAE of 42.901 and MSE of 1895.011, suggesting less effective pattern capture. Performance varies with window size; a 1-year dataset with a window size of 7 had a lower MAE (30.332) and MSE (962.747). Medium time series (7-10 years) show significant improvement. The 10-year dataset with a window size of 7 achieved an MAE of 9.793 and MSE of 189.373, indicating high predictive accuracy, with an R2 score of 0.885. For longer time series (15-20+ years), model performance deteriorates. The 20+ years dataset with a window size of 10 had the highest MAE (204.036) and MSE (44558.06), and a negative R2 score (-20.568), suggesting noise or complexities that the model struggles to handle.[Figure 5]

Insights and Recommendations: Model's excellent ability to capture long-term depen- dencies and make accurate predictions was observed with the 10-year dataset and a window size of 7, yielding the lowest MAE (9.793), MSE (189.373), and a high R2 score (0.885).[Figure 6] Medium-range datasets (7-10 years) with appropriately chosen window sizes (around 7) provide the best balance between data volume and model performance. This range captures sufficient historical data to identify patterns without overwhelming the model with too much noise. For very long datasets, consider techniques such as data normalization, regularization, and reducing

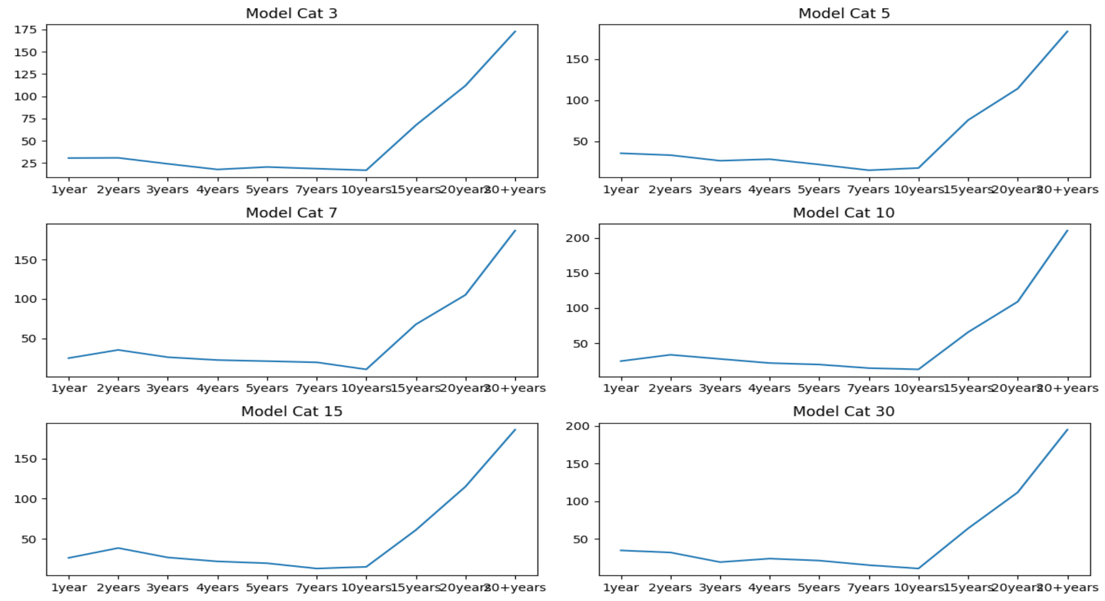

Figure 5: RMSE Difference results variation for different Window size (Model Cat = Window size)

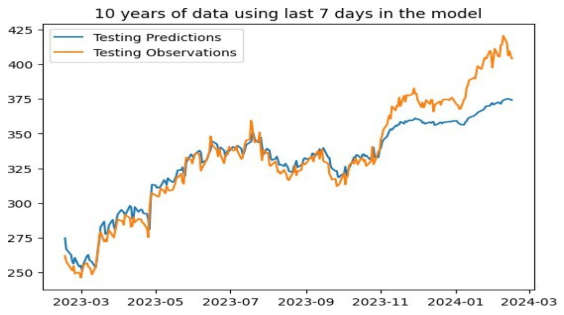

Figure 6: Best performance of Case study 01

complexity to prevent overfitting and improve generalization. Focus on minimizing MAE and MSE while aiming for a higher R2 score to ensure the model accurately tracks the actual values and has good predictive power.

5.2. Sequential LSTM(multi training run)

This analysis focuses on evaluating the performance of Sequential LSTM models using Tensor-Flow across multiple training iterations. This analysis provides insights into the consistency and reliability of the LSTM model's predictive capability. The dataset includes results from 16 different training iterations of the LSTM model. Each iteration exhibits variations in performance metrics, reflecting the model's sensitivity to different training conditions and potential overfitting or underfitting issues.

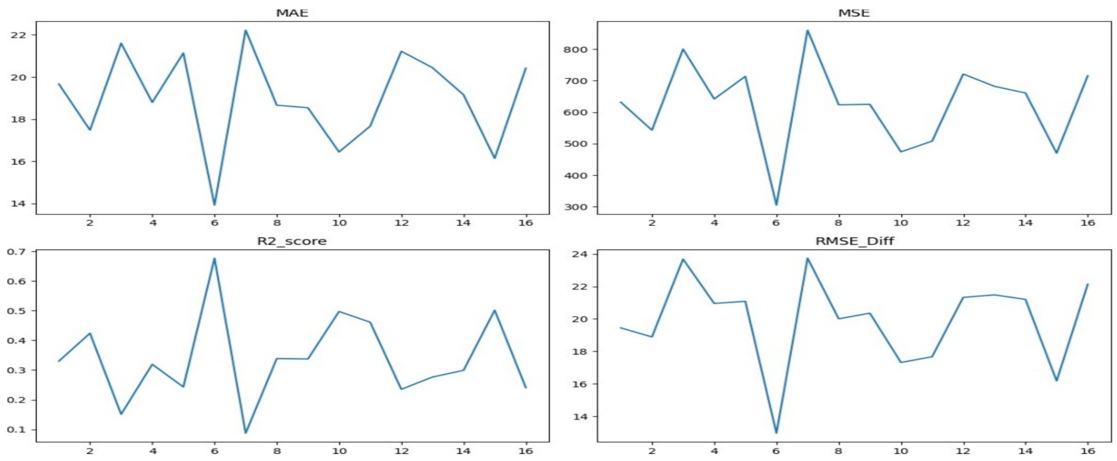

Figure 7: Results of Case study 02

The MAE values range from 13.92 to 22.22, with the lowest (13.92) in the sixth iteration, indicating the highest accuracy, and the highest (22.22) in the seventh iteration, indicating the least accuracy. MSE values range from 304.72 to 860.59, with the best performance in the sixth iteration (304.72) and the worst in the seventh (860.59). R2 scores range from 0.09 to 0.68, with the highest in the sixth iteration (0.68) and the lowest in the seventh (0.09). Training RMSE values range from 4.39 to 5.69, and testing RMSE values range from 17.46 to 29.34, with the smallest difference (12.96) in the sixth iteration, indicating less overfitting, and the largest difference (23.74) in the seventh iteration.[Figure 7]

Insights and Recommendations: The well-trained model and generalized effectively to the test data was observed in the sixth iteration, which had the lowest MAE (13.92), lowest MSE (304.72), highest R2 score (0.68), low Train RMSE (4.5), lowest Test RMSE (17.46), and the smallest RMSE Difference (12.96). [Figure 8] The LSTM model's performance varies significantly

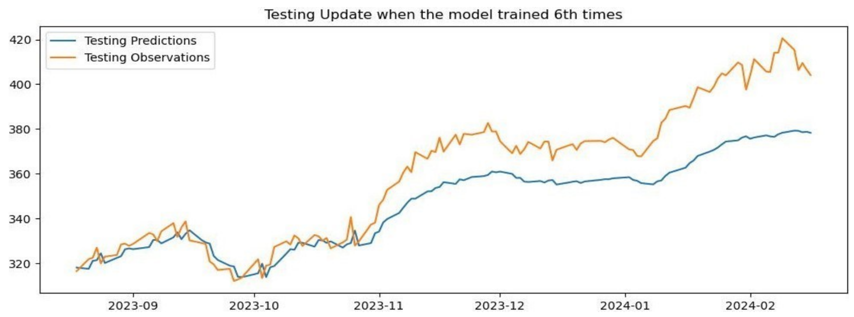

Figure 8: Best performance of Case study 02

across different training iterations, indicating sensitivity to initialization, hyperparameters, and the training process. Choosing the best-performing iteration (the sixth) is beneficial for applications requiring high accuracy and generalization, as this model shows robust performance across all key metrics. Mitigate performance variability with strategies like ensemble learning, hyperparameter tuning, regularization techniques, and consistent data preprocessing. Enhance generalization from training to test data with cross-validation, dropout, and early stopping to avoid overfitting and ensure reliable performance on unseen data.

5.3. Functional LSTM

This analysis explores the performance of a Functional LSTM model in TensorFlow for varying numbers of data inputs. The performance metric analyzed here is accuracy, which indicates the percentage of correct predictions made by the model. The dataset comprises results from 14 different training instances of the LSTM model, with the number of data points increasing with each instance. The window size is consistently set at 60 data points. The accuracy of the model shows significant variation across different training instances.

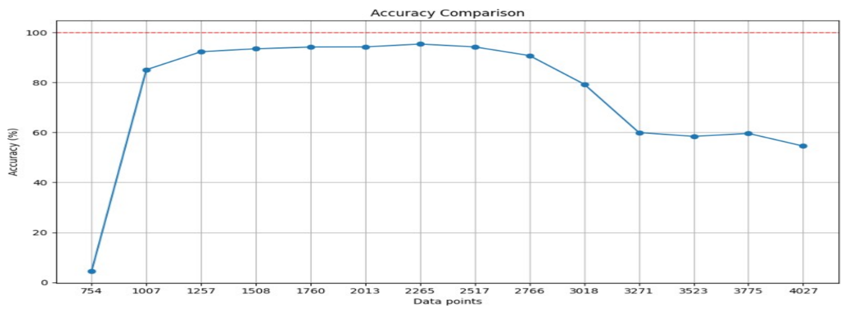

Figure 9: Prediction accuracy of LSTM Model

The accuracy of the model improves as the number of data points increases initially but starts to decline after reaching a peak.[Figure 9] With 754 data points, the model accuracy is notably low at 4.55% indicating poor performance due to insufficient data. As the number of data points increases from 754 to 2265 (years 3 to 9), there is a significant improvement in accuracy, peaking at 95.4%. After reaching 95.4% accuracy with 2265 data points, the performance starts to decline, reaching 54.57% with 4027 data points.

Insights and Recommendations: Accuracy starts low at 4.55% with 754 data points, indicating insufficient information for accurate predictions. It dramatically increases to 85.13% with 1007 data points (year 4) and 92.32% with 1257 data points (year 5), suggesting effective learning with more data. Accuracy continues to improve, reaching 93.51% with 1508 data points (year 6) and peaking at 95.4% with 2265 data points (year 9). This indicates sufficient data for generalization without overfitting. After peaking, accuracy declines to 94.26% with 2517 data points (year 10) and further to 79.32% with 3018 data points (year 12). It drops to 54.57% with 4027 data points (year 16), indicating overfitting and poor generalization.

The highest accuracy (95.4%) was achieved with 2265 data points (year 9). This suggests that the model performed optimally with this amount of data, striking a balance between having enough data for training and avoiding overfitting or saturation.[Figure 10]

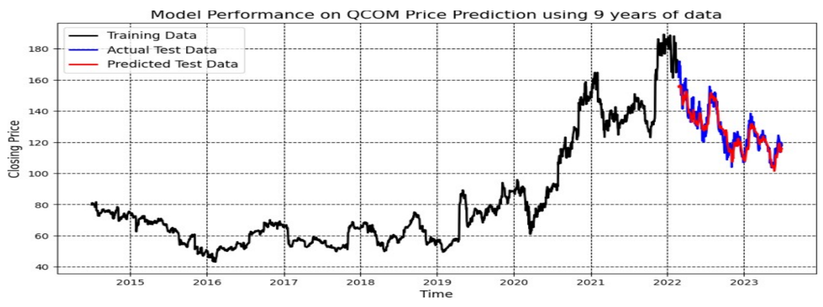

Figure 10: Best performance of Case study 03

Optimal data utilization shows that the best LSTM model performance was around 2265 data points. Beyond this, adding more data led to diminishing returns and a decline in performance. The decline suggests potential overfitting, which can be mitigated by techniques such as regularization, dropout, or reducing model complexity. Cross-validation can ensure the model generalizes well across different data subsets. Further analysis of the performance drop after year 9 could provide valuable insights, including examining data quality and model capacity. Experimenting with different window sizes and hyperparameters may reveal better configurations. For practical applications, balancing data points is crucial—too few lead to poor accuracy, while too many can cause overfitting. Regular monitoring and adjusting training practices based on performance metrics are essential for maintaining optimal model accuracy.

5.4. PyTorch LSTM

In this case, the dataset includes results for 11 different training instances of the LSTM model, with the number of data points increasing for each instance. MAE generally decreases as the number of data points increases, indicating that the model's predictions become more accurate with more data. The lowest MAE (1.65) occurs with 250 data points (1 year). Both MSE and RMSE show a trend of decreasing with an increasing number of data points, with some fluctuations. The lowest MSE (4.8) and RMSE (2.19) are also observed with 250 data points (1 year). MAPE shows a generally decreasing trend with more data, suggesting improved model accuracy in percentage terms. The lowest MAPE (1.29) is again observed with 250 data points (1 year). The accuracy remains relatively high across all datasets, consistently above 97%.[Figure 11] The highest accuracy (98.71%) is observed with 250 data points (1 year).[Figure 11]

Insights and Recommendations: With 123 data points, the model shows poor performance, with the highest MAE, MSE, and RMSE values, and the lowest accuracy (97.14%). A significant improvement is observed with 250 data points, where metrics are optimal, showing substantial benefit from the increased data. As the dataset grows from 250 to 1508 data points (1 to 6 years), the model maintains high accuracy (above 98%) and lower error metrics, indicating effective learning and generalization. Beyond 1508 data points (7 to 10 years), metrics stabilize with minor fluctuations, and accuracy remains high (around 98.1% to 98.4%), showing marginal benefits from further data.

The best overall performance in terms of all metrics (lowest MAE, MSE, RMSE, MAPE, and

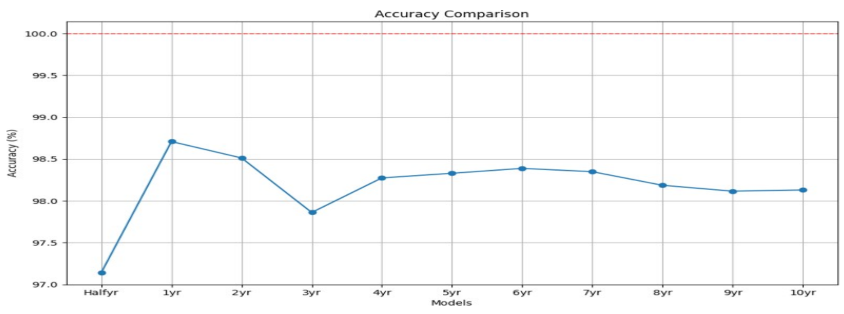

Figure 11: Prediction accuracy of PyTorch LSTM

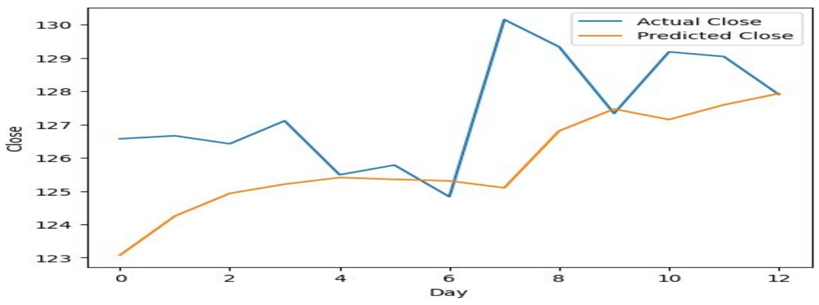

Figure 12: Best result of Case study 04

highest accuracy) is observed with 250 data points (1 year). This indicates that the model performs optimally with this amount of data, achieving an accuracy of 98.71%. [Figure 12]

Model generalization was high and stable (above 97%) across varying data inputs, indicating the LSTM model generalizes well. However, a slight decrease in performance with very large datasets suggests potential overfitting or data quality issues. The decline in performance metrics beyond a certain point implies that additional data might not always improve model performance. Investigating the quality and relevance of added data points could provide insights. Techniques like regularization, dropout, or model tuning might help mitigate overfitting and ensure consistent performance with larger datasets. Practically, identifying the optimal data range is crucial to ensure high performance without unnecessary computational complexity. In this case, 250 data points provide a sweet spot for training the model effectively. Continuous monitoring and validation against a holdout dataset can ensure the model remains robust in real-world scenarios. The model achieves optimal performance with 250 data points, with diminishing returns beyond this initial boost.

6. Conclusion

I analyzed the performance of various LSTM neural network architectures by varying the number of data inputs. My research included sequential, functional, and PyTorch LSTM models, which were evaluated based on different metrics. Sequential LSTM models showed optimal performance with data in the 1-10 year range, with a 10-year dataset and window size of 10 achieving the highest accuracy and lowest error metrics. Functional LSTM models in TensorFlow peaked in accuracy with about 9 years of data (2265 data points), with performance declining beyond this threshold. PyTorch LSTM models performed best with 1-4 years of data. Our research goals included investigating LSTM configurations under varying input data lengths to determine the most effective setups. We found that the number of layers and units significantly impacts model performance, with more layers and units beneficial for longer sequences but risking overfitting for shorter ones. Dropout layers were crucial in preventing overfitting. Key performance metrics like MAE, MSE, RMSE, R^2, MAPE, and accuracy varied with input data lengths, revealing different aspects of model performance. This analysis provides valuable insights for real-world applications, such as stock prediction and demand planning, by identifying model limitations and highlighting the importance of hyperparameter optimization tailored to specific sequence lengths. Overall, increasing data generally improved performance up to the point of diminishing returns, typically around 10 years for TensorFlow models and slightly less for PyTorch. Window size significantly influenced results, highlighting the importance of tailoring model architecture and training parameters to specific datasets. These insights are crucial for optimizing LSTM models in time-series prediction and other sequence modelling tasks.

7. Acknowledgments

I am deeply grateful to my esteemed supervisor, Prof. dr hab. inż. Marcin Woźniak, for his invaluable guidance, support, and encouragement throughout this research. I would also like to extend my sincere thanks to my colleagues for their insightful discussions and collaborative spirit.

References

[1] J. Siłka, M. Wieczorek, M. Woźniak, Recurrent neural network model for high-speed train vibration prediction from time series, Neural Computing and Applications 34 (2022) 13305–13318.
[2] J. D. Hamilton, Time Series Analysis, Princeton University Press, 1994.
[3] C. Chatfield, The Analysis of Time Series: An Introduction, CRC Press, 2004.
[4] P. J. Brockwell, R. A. Davis, Introduction to Time Series and Forecasting, Springer, 2016.
[5] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (1997) 1735–1780.
[6] M. Woźniak, J. Siłka, M. Wieczorek, M. Alrashoud, Recurrent neural network model for iot and networking malware threat detection, IEEE Transactions on Industrial Informatics 17 (2020) 5583–5594.
[7] Q. Ke, X. Jing, M. Woźniak, S. Xu, Y. Liang, J. Zheng, Apgvae: Adaptive disentangled representation learning with the graph-based structure information, Information Sciences 657 (2024) 119903.
[8] M. Woźniak, M. Wieczorek, J. Siłka, Bilstm deep neural network model for imbalanced medical data of iot systems, Future Generation Computer Systems 141 (2023) 489–499.

[9] J. Brownlee, Long Short-Term Memory Networks With Python, Machine Learning Mastery, 2017.
[10] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[11] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with lstm, Neural Computation 12 (2000) 2451–2471.
[12] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: NIPS '14: Proceedings of the 27th International Conference on Neural Information Processing Systems, 2014, pp. 3104–3112.
[13] C. Olah, Understanding lstm networks, 2015. URL: https://colah.github.io/posts/ 2015-08-Understanding-LSTMs/.
[14] F. Chollet, Deep Learning with Python, Manning Publications, 2018.