Received Date: Sept. 2025 Revised: Oct. 2025 Accepted: Dec. 2025

Stock Price Prediction Using LSTM GRU and BiLSTM

Paras Mishra
Affiliation of Central Department of MIT, Tribhuvan University, Kritipur
Correspond Author: Parasmishra428@gmail.com

Doi: https://doi.org/10.3126/ppj.v5i2.92904

Abstract

This research investigates the comparative performance of Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (BiLSTM) models for stock price prediction using Tesla dataset. The primary objective was to evaluate the predictive accuracy of these models by minimizing the root mean square error (RMSE) through rigorous hyperparameter tuning. Parameters like number of units (50 and 100), number of layers (1 and 2), L2 regularization (0.001 and 0.01), dropout rate, and batch size was applied to optimize each model's architecture. This research revealed that GRU consistently achieved the lowest RMSE outperforming both LSTM and BiLSTM with hyperparameters batch size of 32, dropout rate of 0.2, L2 regularization of 0.001, one layer, and 50 units.

Key words: Stock Price Prediction, LSTM, GRU, BILSTM, and Hyper parameter Tuning

1. Introduction

Stock price prediction is one of the most challenging and intriguing areas of research in financial markets. Accurate predictions can provide investors with a significant edge, enabling them to make informed decisions and maximize returns. However, stock markets are inherently volatile and influenced by a multitude of factors, including economic indicators, geopolitical events, and investor sentiment. Traditional statistical methods, such as ARIMA and GARCH, have been widely used for stock price forecasting, but they often struggle to capture the non-linear and complex patterns inherent in financial data.

With the advent of deep learning, models like Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Bidirectional LSTM (BiLSTM) have emerged as powerful tools for time-series forecasting. These models excel at handling sequential data and capturing long-term dependencies, making them particularly suitable for stock price prediction. Furthermore, hyperparameter optimization techniques, such as Grid Search, Random Search, and Bayesian Optimization, have been shown to significantly enhance model performance by fine-tuning critical parameters. This study aims to explore the effect

of hyperparameter optimization for LSTM, GRU, and BiLSTM models, combined with hyperparameter optimization, for stock price prediction.

The stock market is highly volatile and influenced by various factors, making accurate price prediction a challenging task. Traditional models struggle to capture complex patterns in stock data, while deep learning approaches like LSTM, GRU, and BiLSTM offer promising results and hyperparameter also affect their accuracy. Their performance heavily depends on selecting optimal hyperparameters as it plays crucial role for model performance. This study aims to compare these models for Tesla stock price prediction, focusing on improving accuracy through hyperparameter optimization and evaluating their effectiveness using performance metrics RMSE.

2. Objectives

The objective of this research is to evaluate the performance of LSTM, GRU, and BiLSTM model with hyperparameter optimization.

3. Literature Review

Stock price prediction has been a key area of research, with traditional statistical methods often struggling to capture complex financial time-series dependencies. The rise of deep learning, particularly LSTM, GRU, and BiLSTM, has significantly improved predictive accuracy due to their ability to model long-term dependencies. Hyperparameter optimization techniques like Grid Search and Bayesian Optimization further enhance performance. While global markets have widely adopted these models, Nepal's stock market (NEPSE) remains in the early stages of leveraging deep learning for financial forecasting. Preliminary studies on LSTM show promise, but advanced techniques such as GRU, BiLSTM, and hyperparameter tuning remain underexplored in the Nepali context. Hyperparameter optimization has further enhanced the performance of these models. Techniques like Grid Search, Random Search, and Bayesian Optimization have been widely adopted to fine-tune parameters such as learning rates, batch sizes, and the number of layers. These advancements have led to significant improvements in prediction accuracy, making deep learning models indispensable tools for financial analysts and investors. Despite these challenges, there is a growing interest in applying machine learning and deep learning techniques to the Nepali stock market. Preliminary studies, such as those by Thapa et al. (2021), have explored the use of LSTM for predicting NEPSE index movements.

LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to handle long-range dependencies in sequential data. It uses memory cells with input, forget,

and output gates to control information flow and mitigate the vanishing gradient problem. LSTMs are widely used in tasks like speech recognition, language modeling, and time-series forecasting like stock market prediction.

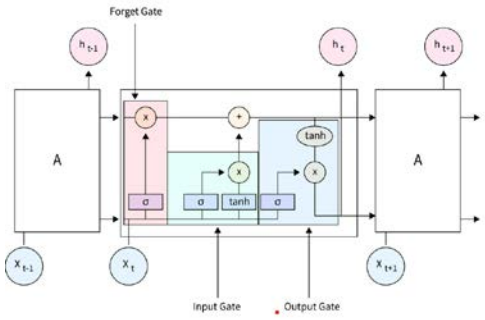


Figure 2.1.1 : Architecture of LSTM Model

The equation for the representation of various gates and cell state is given below;

Forget gate:

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

Input gate:

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

Cell state update:

$$\tilde{c}_t = \tanh(w_c.[h_{t-1}.x_t] + b_c)$$

Final cell state:

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

Output gate:

$$o_t = \sigma(w_o.[h_{t-1}, x_t] + b_o)$$

Hidden state:

$$h_t = o_t.\tanh(C_t)$$

GRU

Gated Recurrent Unit (GRU) is a variant of recurrent neural networks (RNNs) designed to efficiently capture long-range dependencies in sequential data. It uses an update gate and a

reset gate to control information flow, making it simpler and faster than LSTM. GRUs are widely used in speech recognition, machine translation, and time-series forecasting like stock price prediction.

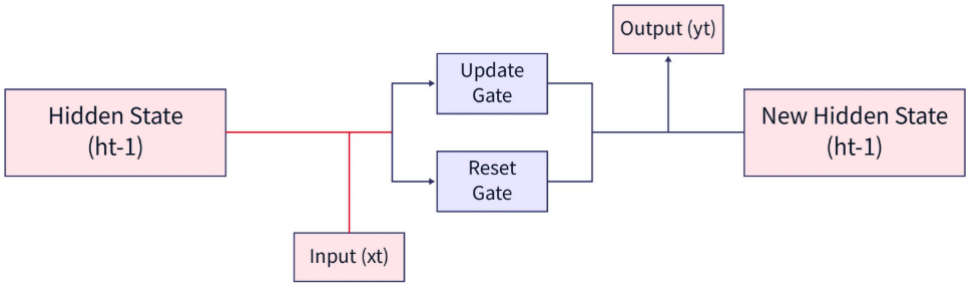


Figure 2.1.2 : Architecture of GRU Model

The equation for the representation of various gate and cell state is given below;

Update gate:

$$z_t = \sigma(w_z.[h_{t-1}, x_t] + b_z)$$

Reset gate:

$$r_t = \sigma(w_r.[h_{t-1}, x_t] + b_r)$$

Candidate hidden state:

$$\tilde{h_t} \tanh(w.[r_t.h_{t-1}, x_t] + b)$$

Final hidden state:

$$h_t = (1 - z_t).h_{t-1} + z_t.\tilde{h}_t$$

BiLSTM

Bidirectional Long Short-Term Memory (BiLSTM) is an extension of LSTM that processes input sequences in both forward and backward directions. This allows the model to capture past and future context, improving performance in tasks requiring deep contextual understanding. BiLSTM is widely used in natural language processing, speech recognition, and time-series analysis.Here's a simple diagram illustrating BiLSTM:

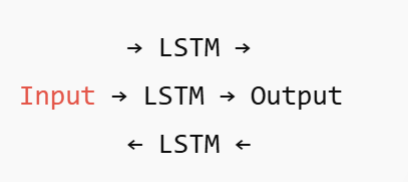


Figure 2.1.3: Representation of BiLSTM Model

This structure shows how BiLSTM processes data in both forward and backward directions before combining the results resulting in learning from both previous and future elements in a sequence, enhancing accuracy in context-dependent tasks.

Forward pass:

$$\overrightarrow{h}_t = \text{LSTM}(x_t, \overrightarrow{h}_{t-1})$$

Backward pass:

$$\overleftarrow{h}_t = \text{LSTM}(x_t, \overleftarrow{h}_{t-1})$$

Final output:

$$h_t = [\overrightarrow{h}_t ; \overleftarrow{h}_t]$$

Stock price prediction has long been one of the central challenges in financial research, with machine learning and deep learning methods gaining significant traction in recent years. Among these, recurrent neural network (RNN) variants such as Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional LSTM (BiLSTM) have been widely explored.

LSTM networks, first proposed by Hochreiter and Schmid Huber [1], remain foundational in sequential data analysis. Fischer and Krauss [2] later demonstrated the superiority of LSTM over traditional statistical models like ARIMA in stock market forecasting. Their experiments on the S&P 500 dataset revealed that LSTM can effectively capture long-term dependencies in time-series data, leading to improved predictive accuracy.

GRU, introduced as a simplified alternative to LSTM, has also shown promise. Bao et al. [3] reported that GRU models not only maintain strong predictive accuracy but also achieve faster computation, making them suitable for volatile market environments. On the other hand, BiLSTM models process data in both forward and backward directions, thereby enhancing feature extraction. Zhang et al. [4] illustrated how BiLSTM can capture complex temporal patterns more effectively than unidirectional LSTM, particularly in highly dynamic datasets.

The role of hyperparameter optimization in improving model performance cannot be overstated. Bergstra and Bengio [5] were among the first to stress its importance, highlighting how tuning learning rates, batch sizes, and network depth can dramatically alter results. Feurer and Hutter [6] later expanded on this, showing how techniques such as Bayesian Optimization and Grid Search enable more efficient exploration of the parameter space.

Several comparative studies have examined LSTM, GRU, and BiLSTM head-to-head. Siami-Namini and Siami-Namini [7] found that BiLSTM typically achieves higher accuracy, whereas GRU offers faster training, making it more appropriate for real-time forecasting applications. Importantly, they emphasized that model performance is highly sensitive to hyperparameter choices, reinforcing the need for systematic tuning.

In the Nepali context, Pandey and Shrestha [8] argued that combining hyperparameter optimization with deep learning architectures could provide powerful forecasting tools for local investors. This integration, they noted, could help democratize financial analytics in developing markets where access to sophisticated tools has traditionally been limited.

Collectively, these studies suggest that while LSTM, GRU, and BiLSTM each bring unique strengths, their effectiveness is closely tied to how well their hyperparameters are tuned. Without proper optimization, even state-of-the-art architectures may underperform, underscoring the need for systematic approaches to model selection and tuning in stock price prediction.

Mishra and Pandey [9] further demonstrated this by comparing multiple linear regression models with L1 and L2 regularization, emphasizing that careful parameter tuning can significantly enhance predictive accuracy in stock price forecasting.

4. Methodology and Implementation details

4.1. Methodology

Corchado, J. M., & Lees, the general approach for this project is illustrated in the given figure. The process involves several steps which are discussed below;

Figure 3.1: Flowchart for the Representation of the project

4.1.1. Data Collection

Dataset from Kaggle for TSLA dataset is used for analysis. Historical stock data from kaggle (Varpit94, 2021) is collected for the study and the dataset used for this study is loaded and cleaned to ensure there are no missing or inconsistent values. The independent variables and the dependent variable are identified and separated. For this study, the features include 'Total Transactions', 'Total Traded Shares', 'Total Traded Amount', 'Max. Price', and 'Min. Price', while the target variable is 'Close Price'.

The simple architecture of collected dataset that has been used for analysis is shown in the table below;

Table 3.1 : Structure of Dataset

S.N.	Date	Total Transactions	Total Traded Shares	Total Traded Amount	Max. Price	Min. Price	Close Price
1	2021-12-29	34	696	842596	1227	1205	1227
2	2021-12-28	48	1322	1575896.8	1227	1180.1	1227
3	2021-12-27	45	1023	1256329	1285.2	1204	1204
4	2021-12-26	43	2051	2510045	1239.8	1194.2	1238.8
5	2021-12-23	41	1153	1390142	1221	1181.1	1200
6	2021-12-22	45	1929	2303884.6	1215.8	1170	1200
7	2021-12-21	38	1325	1499667	1149	1110	1148
8	2021-12-20	27	470	535034.5	1156	1125	1125

4.1.2. Data Preprocessing

After the data is collected, it is loaded into the environment and examined for any missing or inconsistent values. Next, feature selection is performed, where the independent variables chosen for analysis include Total Transactions, Total Traded Shares, Total Traded Amount, Max. Price, and Min. Price, while the target variable is identified as Close Price. To standardize the data, feature scaling is applied using either Min-Max Scaling, ensuring that all features are on a similar scale for better model performance. Additionally, missing values, if present, are addressed using imputation techniques like mean substitution or forward-fill methods. After completing these preprocessing steps, the dataset is well-structured and ready for training the LSTM, GRU, and BiLSTM models for comparative analysis.

4.1.3. Model Selection

LSTM, GRU, and BiLSTM variants of RNN model is used as they are well-suited for sequential data and can capture dependencies in stock price trends. LSTM, GRU, and BiLSTM models were evaluated to determine the most suitable architecture for stock price prediction. Performance metric RMSE is used to compare model accuracy. After training, the model with different hyperparameter, the model with optimal value of performance metrics with particular value of hyperparameter is selected. Finally, Visualization of actual vs. predicted stock prices further validated the effectiveness of the chosen model.

4.1.4. Hyperparameter Optimization

A Grid Search was performed to optimize learning rate, batch size, number of layers, and units. The following are the various hyperparameter which are tuned;

- Learning Rate
- Batch Size
- Number of Neurons/Units in Layers
- Dropout Rate
- Number of Epochs

4.1.5. Model Evaluation

RMSE (Root Mean Square Error) is used to evaluate the model's performance, as it penalizes larger errors more significantly, making it suitable for stock price prediction.

RMSE is an evaluation parameter for the analysis of model. RMSE is the square root of the MSE. It provides an error metric on the same scale as the original data, making it more interpretable in the context of the data.

Root Mean Squared Error $(\mathbf{RMSE}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

where yi are the actual values and $\hat{y}_i$ are the predicted values, and n is the number of observations.

4.2. Implementation Details

4.2.1. Implementation Tools

Programming Language:

Python is used as the primary programming language to implement and run the program due to its simplicity, wide adoption in the machine learning community, and the availability of a rich ecosystem of libraries.

Integrated Development Environment (IDE):

Google Colab is used as the IDE for coding and experimentation. It provides a cloud-based platform that supports GPU acceleration and allows easy collaboration and sharing of notebooks.

Libraries and Frameworks:

- **Scikit-learn:**
  Scikit-learn is used to build and train various machine learning models. It also supports tasks such as feature engineering, model evaluation, and hyperparameter tuning.
- **TensorFlow:**
  TensorFlow is an open-source deep learning framework developed by Google. It is used for building and training deep neural network models. In this project, TensorFlow is utilized to implement advanced models such as LSTM, GRU, BiLSTM.
- **GridSearchCV:**
  GridSearchCV from Scikit-learn is used to perform hyperparameter tuning. It exhaustively searches through a specified set of hyperparameters to find the optimal combination that provides the best model performance. This method ensures that the model is well-tuned and performs effectively on unseen data by evaluating different configurations using cross-validation.
- **NumPy:**
  NumPy is used for efficient storage and manipulation of numerical data, including

training and testing datasets. It provides support for large, multi-dimensional arrays and matrices.

- **Pandas:**
  Pandas is used to load, clean, and preprocess the training and testing data. Its DataFrame structure makes data manipulation intuitive and effective.
- **Matplotlib:**
  Matplotlib is used to create visualizations for analyzing model performance. It helps in plotting graphs and charts that illustrate the results clearly.

4.2.2 Implementation Details

Data preprocessing

This step involves cleaning the data, handling missing values, and normalizing the data to ensure consistency. First, the dataset has been loaded and extracted the relevant and the target variable. Split the data into training and testing sets. The dataset is split into training and testing sets using 70-30 ratio. This ensures that the model can be trained on one portion of the data and evaluated on another to assess its performance on unseen data.

```
# Check for missing values
print("\nMissing values:")
print(df.isnull().sum())
# Fill or drop missing values if necessary (optional)
df = df.dropna()

# Select features and target
# Example: Use 'Open', 'High', 'Low', 'Volume' to predict 'Close'
X = df[['Open', 'High', 'Low', 'Volume']]
y = df['Close']

# Normalize the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets (70-30 ratio)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

print("\nShape of training set:", X_train.shape)
print("Shape of testing set:", X_test.shape)
```

Figure 3.2.2 : Data Preprocessing Implementation

4.3. Hyper parameter optimization

GridSearchCV is employed to perform hyper parameter tuning for lstm, gru and bilstm models. This involves specifying a range of hyper parameters values and using cross-validation to determine the best hyper parameter value that minimizes the error metrics.

```
def hyperparameter_search(model_type, param_grid, X_train, y_train, X_test, y_test, epochs=10):
    best_rmse = float("inf")
    best_params = None
    best_model = None
    best_pred = None

    for params in ParameterGrid(param_grid):
        print(f"Testing params: {params}")
        model = build_model(model_type, units=params['units'], layers=params['layers'],
                            dropout_rate=params['dropout_rate'], l2_value=params['l2_value'])
        train_rmse, test_rmse, train_mae, test_mae, test_pred = train_and_evaluate(
            model, X_train, y_train, X_test, y_test, epochs=epochs, batch_size=params['batch_size'])

        if test_rmse < best_rmse:
            best_rmse = test_rmse
            best_params = params
            best_model = model
            best_pred = test_pred

        print(f"Train RMSE: {train_rmse}, Test RMSE: {test_rmse}")

    print(f"Best RMSE for {model_type}: {best_rmse} with params: {best_params}")
    return best_model, best_pred, best_rmse, best_params
```

Figure 3.3 : Hyperparameter Optimization Implementation

4.4. Model Training

LSTM, GRU, BILSTM model is trained with tesla dataset with different hyperparameter value to find the optimal model for analysis.

4.4.1. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to handle long-term dependencies in sequential data, making it well-suited for stock price prediction. During training, the Adam optimizer is used to update weights efficiently, and the Mean Squared Error (MSE) loss function minimizes prediction errors. Hyperparameters such as batch size, number of epochs, LSTM units, and learning rate are optimized to improve model accuracy.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(time_steps, features)))
model.add(LSTM(units=50))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

Figure 3.4.1 : LSTM Model Implementation

4.4.2. Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is an advanced variant of the recurrent neural network (RNN) that simplifies the architecture of Long Short-Term Memory (LSTM) by combining the forget and input gates into a single update gate. GRUs are computationally efficient and can capture long-term dependencies in time series data while requiring fewer parameters than LSTMs.

During training, the Adam optimizer is used to optimize weights, and the Mean Squared Error (MSE) loss function minimizes the prediction error. The model's hyper parameters, including batch size, number of epochs, GRU units, and learning rate, are fine-tuned to improve performance.

```
from tensorflow.keras.layers import GRU

model = Sequential()
model.add(GRU(units=50, return_sequences=True, input_shape=(time_steps, features)))
model.add(GRU(units=50))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

Figure 3.5 : GRU Model Implementation

4.4.3. Bidirectional Long Short-Term Memory(BiLSTM)

BiLSTMs extend LSTM by making the model bi-directional. It processes data in both forward and backward directions, allowing it to capture patterns more comprehensively, which can be beneficial in understanding stock price movements. RMSE loss function and adam optimizer has been used during training. Hyper parameters such as batch size, number of epochs, number of LSTM units, learning rate has been tuned for analysis. In BiLSTM, forward and backward LSTM cells are computed separately, and then their outputs are concatenated.

```
from tensorflow.keras.layers import Bidirectional
model = Sequential()
model.add(Bidirectional(LSTM(units=50, return_sequences=True), input_shape=(time_steps, features)))
model.add(Bidirectional(LSTM(units=50)))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

Figure 3.6 : BiLSTM Model Implementation

4.5 Model Evaluation

To evaluate the model's performance Root Mean Squared Error (RMSE) has been calculated for both training and testing datasets. The predictions are first inverse-transformed to their original scale using the same scaler before calculating these error metrics.

```
def train_and_evaluate(model, X_train, y_train, X_test, y_test, epochs=50, batch_size=64):
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size, verbose=1,
                        callbacks=[early_stopping])

    # Predictions
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    # Inverse transform predictions to original scale
    train_predict = scaler.inverse_transform(train_predict)
    test_predict = scaler.inverse_transform(test_predict)

    # Calculate RMSE, MSE, MAE
    train_rmse = math.sqrt(mean_squared_error(scaler.inverse_transform(y_train.reshape(-1, 1)), train_predict))
    test_rmse = math.sqrt(mean_squared_error(scaler.inverse_transform(y_test.reshape(-1, 1)), test_predict))
    train_mae = mean_absolute_error(scaler.inverse_transform(y_train.reshape(-1, 1)), train_predict)
    test_mae = mean_absolute_error(scaler.inverse_transform(y_test.reshape(-1, 1)), test_predict)

    print(f"Train RMSE: {train_rmse}, Test RMSE: {test_rmse}")
    return train_rmse, test_rmse, train_mae, test_mae, test_predict
```

Figure 3.7 : Model Evaluation Implementation

5. Results and Findings

5.1. Findings and Observations

This section presents the evaluation results of LSTM, GRU, and BiLSTM models on the TSLA stock dataset using various hyperparameter combinations. Table below highlights the performance in terms of Train RMSE and Test RMSE for each configuration.

Table 5.1: Evaluation parameter for different hyper parameter value for TSLA

Model	Hyperparameters	Train RMSE	Test RMSE
LSTM	‘batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 50	21.91	26.88
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 100	19.73	24.32
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 50	27.72	36.72

	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 100	29.39	35.64
GRU	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 50	17.06	20.45
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 100	17.60	20.84
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 50	26.12	30.95
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 100	35.09	40.00
BiLSTM	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 50	21.95	27.74
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 1, 'units': 100	21.23	25.86
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 50	25.84	33.31
	'batch_size': 32, 'dropout_rate': 0.2, 'l2_value': 0.001, 'layers': 2, 'units': 100	22.99	29.96

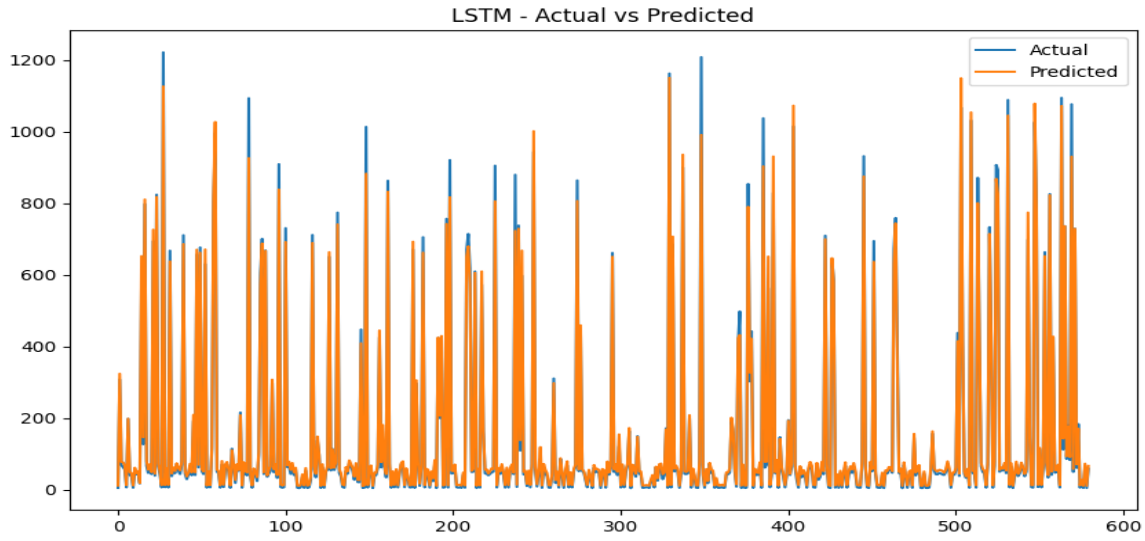
From the results, several key observations can be drawn:

1. **Impact on Number of Units**
    - Increasing the number of units generally improved performance across all models for single-layer architectures. For example, LSTM with 1 layer and 100 units achieved a **Test RMSE of 24.32**, which is lower than 26.88 from the same setup with 50 units.
    - A similar trend is seen in GRU and BiLSTM. GRU with 1 layer and 100 units gave **Test RMSE of 20.84**, while 50 units gave 20.45 (slightly better).
2. **Effect of Number of Layers**
    - Increasing the number of layers led to **worse performance** in most cases. For instance, LSTM with 2 layers and 100 units resulted in a **Test RMSE of 35.64**, compared to 24.32 with 1 layer and 100 units.
    - This trend suggests that a deeper architecture may lead to overfitting on the training data, especially when the dataset is not very large.

3. **Model Comparison**
    - Among all models and configurations, **GRU with 1 layer and 50 units** yielded the **best Test RMSE of 20.45**, indicating that GRU might be better suited for TSLA stock prediction on this dataset.
    - BiLSTM generally performed slightly worse than GRU and similarly to LSTM in terms of RMSE, though its performance was more consistent across different unit values.
4. **Overall Best Configuration**
    - The optimal result was achieved with **GRU (1 layer, 50 units)** using dropout and L2 regularization, emphasizing that simpler architectures with proper regularization can outperform deeper models.

These findings underline the importance of hyperparameter tuning and model selection in time series forecasting tasks, especially when dealing with real-world financial data.

Actual vs predicted stock price for LSTM model



Index

Figure 3 : Actual vs predicted price for TSLA dataset for LSTM model

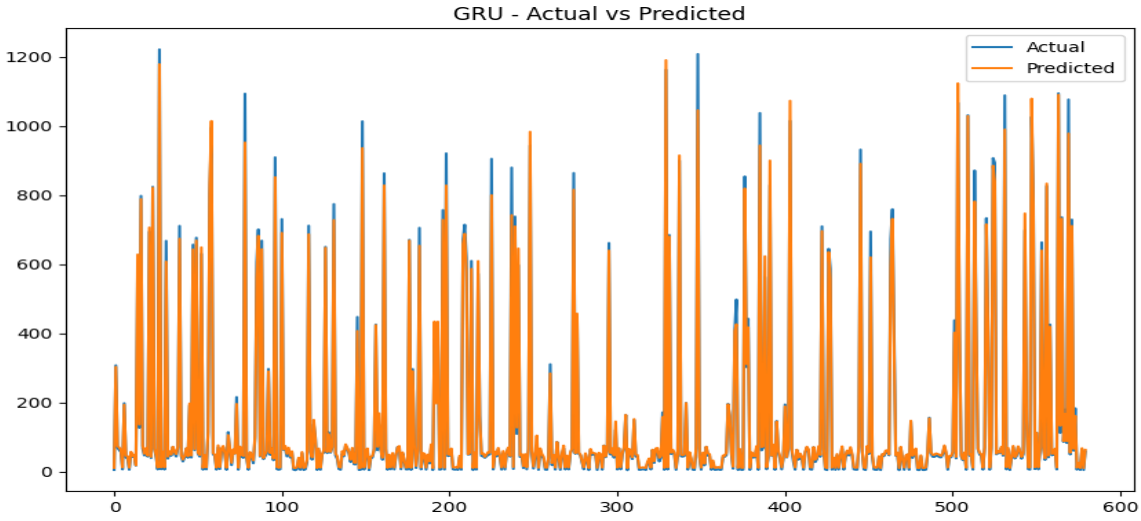


Figure 42 : Actual vs predicted price for TSLA dataset for GRU model

Actual vs predicted for TSLA stock price for BILSTM model

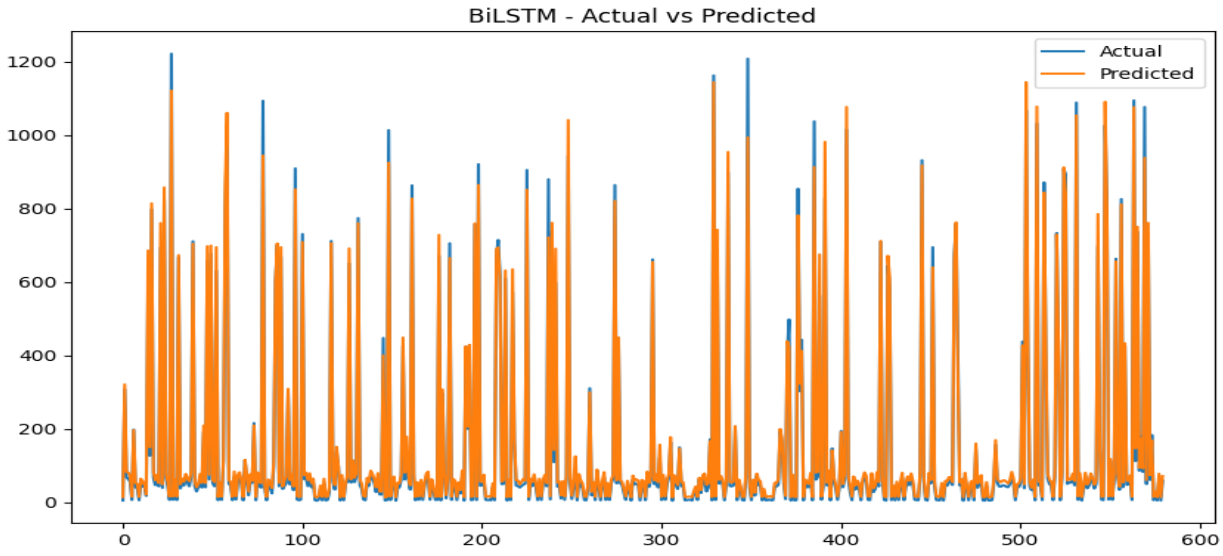


Figure 4.3: Actual vs predicted price for TSLA dataset for BILSTM model

GRU

Again, performed best among the models, achieving a minimum RMSE of 20.46 with a batch size of 32, dropout rate of 0.2, L2 regularization of 0.001, one layer, and 50 units. This significantly improved performance on the TSLA dataset suggests that GRU's architecture is well-suited for the TSLA data's inherent patterns and volatility.

LSTM

Recorded an RMSE of 24.33 with the same hyperparameter configuration as used for BiLSTM and GRU. The improved performance compared to the NEPSE dataset shows that LSTM's capability for sequence prediction aligns better with the TSLA dataset, though it still trailed behind GRU.

BiLSTM

Achieved an RMSE of 25.87 using the same parameter settings, though it remained the least accurate on the TSLA dataset compared to GRU and LSTM. Despite bidirectional processing, BiLSTM's performance was limited, suggesting that additional computational complexity might not be beneficial for this particular dataset's temporal characteristics.

GRU Model Training and Validation Loss over Epochs

Figure 4.4: GRU Model Training and Validation Loss over Epochs

4.2. Result Analysis

The analysis reveals that GRU consistently outperformed both LSTM and BiLSTM across both NEPSE and TSLA datasets, with the best RMSE scores. For NEPSE, GRU's best

RMSE was 27.60, which was substantially lower than LSTM's (35.72) and BiLSTM's (36.88) RMSE scores. For TSLA, GRU again demonstrated superior performance, achieving an RMSE of 20.46, compared to LSTM's 24.33 and BiLSTM's 25.87. These findings underscore that GRU's architecture can handle sequential data efficiently, even with fewer units, as seen in the TSLA dataset where GRU used only 50 units but achieved the best RMSE. The study also highlights the impact of hyperparameter tuning in optimizing model performance, as evidenced by the consistent use of a dropout rate of 0.2 and L2 regularization of 0.001 across all models. These settings helped mitigate overfitting and added regularization, which was critical in refining the predictive accuracy of each model.

5. Conclusion

The results suggest that GRU is more effective for stock price prediction within these datasets, particularly in terms of RMSE. While LSTM and BiLSTM possess robust architectures for capturing long-term dependencies, GRU's comparatively simpler structure may provide computational efficiency benefits and superior predictive accuracy. With optimal hyperparameters set to batch_size: 32, dropout_rate: 0.2, l2_value: 0.001, layers: 1, and units: 50, the results demonstrate that hyperparameter selection significantly impacts model performance compared to other hyperparameter values.

6. Future Work

Future research can focus on exploring advanced architectures like Transformers and Informer models to better capture long-term dependencies in financial data. Additionally, implementing ensemble models by combining LSTM, GRU, and BiLSTM using techniques like bagging and boosting may enhance predictive robustness. More sophisticated hyperparameter optimization methods, such as Bayesian Optimization or Genetic Algorithms, could further refine model performance. Expanding feature engineering by incorporating additional indicators, such as sentiment scores from financial news, and applying feature selection techniques may also improve accuracy. These advancements collectively aim to optimize predictive capabilities in financial modeling.

Conflict of Interest

I declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[3] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long–short term memory," *PLoS ONE*, vol. 12, no. 7, pp. 1–24, 2017.

[4] Y. Zhang, J. Aggarwal, and H. Zhang, "Stock price prediction using Bidirectional LSTM networks," *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[6] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning: Methods, Systems, Challenges*, Springer, 2019, pp. 3–33.

[7] H. Siami-Namini and A. Siami-Namini, "Forecasting economics and financial time series: ARIMA vs. LSTM," *arXiv preprint arXiv:1803.06386*, 2018.

[8] R. Pandey and S. Shrestha, "Deep learning and hyperparameter optimization for financial forecasting in Nepal," *Nepal Journal of Finance and Economics*, vol. 5, no. 1, pp. 45–54, 2022.

[9] P. Mishra and R. Pandey, "Comparative analysis of multiple linear regression with L1 and L2 regularization for stock price prediction," *Aadim Journal of Multidisciplinary Research*, vol. 1, no. 1, pp. 13–24, Jul. 2025.

[10] V. Varpit, "TESLA Stock Data (Updated till 28 Jun 2021)," Kaggle, 2021. [Online]. Available://www.kaggle.com/datasets/varpit94/tesla-stock-data-updated-till-28jun2021
rovince. *Journal of Cultural Studies, 8*(3), 33-44.