AIDI 1002 Final Term Project Report

Student Name : Jay Pravinbhai Patel(200538084)

Email: 200538084@student.georgianc.on.ca

Student Name : Ashish Pandya(200575073)

Email: 200575073@student.georgianc.on.ca

Research Paper Link: https://github.com/ritvik02/Financial-Data-Time-Series-Forecasting-Using-Neural-Networks/blob/main/Paper.pdf

Source Link: https://ieeexplore.ieee.org/document/9725845

Problem Description:

The project, "Financial-Data-Time-Series-Forecasting-Using-Neural-Networks", is focused on predicting future financial data based on historical time series data using neural networks. The associated paper is currently in press at IEEE ICONAT 2022.

The problem involves forecasting financial data, a task that has significant implications for decision-making processes in the financial sector. By using neural networks, the project aims to identify complex patterns and relationships in the data, which can then be used to predict future financial trends.

Context of the Problem:

The problem of forecasting financial data is of paramount importance due to its direct impact on the global economy. Accurate predictions can guide investment strategies, inform policy decisions, and help mitigate risks associated with market volatility. In an increasingly data-driven world, the ability to accurately forecast financial trends is a key competitive advantage for businesses and investors alike.

Limitations About Other Approaches:

Traditional methods of financial forecasting, such as statistical and econometric models, often rely on assumptions that may not hold in real-world scenarios, such as linearity and stationarity of time series data. These models may also struggle to capture complex patterns and relationships in the data.

Machine learning approaches, while powerful, also have their limitations. They often require large amounts of data to perform well and can be prone to overfitting. Furthermore, they can act as “black boxes”, making it difficult to interpret their predictions and understand the underlying relationships in the data.

The use of neural networks in this project aims to address some of these limitations by leveraging their ability to model complex, non-linear relationships and their capacity for handling large datasets. However, it’s important to note that while neural networks can improve prediction accuracy, they also require careful tuning and validation to ensure robust performance. Additionally, interpretability remains a challenge with these models.

Solution

To overcome limitations in financial forecasting, this project uses neural networks, which can handle complex relationships and large datasets. Overfitting is mitigated using regularization and early stopping. Interpretability tools like LIME or SHAP are used to understand predictions. Hyperparameter optimization and cross-validation ensure robust performance. For smaller datasets, data augmentation or transfer learning are used. This combination of strategies helps improve forecasting accuracy.

Background

Reference	Citation	Dataset	Explanation
[1]	G. Box and G. Jenkins, "Time Series Analysis: Forecasting and Control," San Francisco: Holden-Day, 1970.	Not specified	This reference discusses the fundamentals of time series analysis and forecasting.
[2]	A. M. Alonso and C. Garcia-Martos, " Time Series Analysis - Forecasting," Universidad Carlos III de Madrid,, 2012.	Not specified	This reference provides a comprehensive overview of time series analysis and forecasting techniques.

Reference	Citation	Dataset	Explanation
[3]	M. Khashei and M. Bijari, " A Novel Hybridization of Artificial Neural Networks and ARIMA Models for Time Series forecasting," Applied Soft Computing, 2011.	Not specified	This reference introduces a novel hybrid model that combines artificial neural networks and ARIMA for time series forecasting.
[4]	X.A.D.N.H.C.Krauss, "Deep neural networks, gradient boosted trees, random forests: Statistical arbitrage on the S&P 500," FAU Discussion Papers in Economics, 2016.	S&P 500 data	This reference explores the use of deep learning and ensemble methods for predicting the S&P 500.
[5]	S.J.S.J.Y.S.I.Lee, "A Deep Efficient Frontier Method for Optimal Investments," Department of Computer Engineering Sejong University, 2017.	Not specified	This reference proposes a deep learning-based method for optimizing investment portfolios.
[6]	C.K.T.Fischera, "Deep Learning with Long Short-term Memory Networks for Financial Market Predictions," FAU Discussion Papers in Economics, 2017.	Financial market data	This reference investigates the use of LSTM networks for predicting financial markets.
[7]	I.E.Livieris and E.P.& P.Pintelas , "A CNN–LSTM model for gold price time-series forecasting," Emerging applications of Deep Learning and Spiking ANN ,2020.	Gold price data	This reference presents a hybrid CNN-LSTM model for forecasting gold prices.
[8]	S.Selvin,R.Vinayakumar,E.A.Gopalakrishnan,V.K.Menon,and K.P.Soman ,"Stock price prediction using LSTM,RNNand CNNsliding window model ," IEEE ,2017.	Stock price data	This reference explores the use of LSTM, RNN, and CNN models for stock price prediction.
[9]	N.T.a.A.S.N.S.S.Namini ,"A Comparison of ARIMAand LSTM in Forecasting Time Series ,"17th IEEE International Conference on Machine Learningand Applications (ICMLA),2018.	Not specified	This reference compares the performance of ARIMA and LSTM models in time series forecasting.
[10]	S.Hochreiterand J.Schmidhuber ,"Long Short-Term Memory ,"Neural Computation ,pp .1735-1780 ,1997 .	Not specified	This reference introduces the concept of Long Short-Term Memory (LSTM) networks.
[11]	P.Baldi,S.Brunak,P.Frasconi,G.Soda,and G.Pollastr ,"Exploiting the pastand the futurein protein secondary structure prediction ,"Bioinformatics ,1999.	Protein structure data	This reference discusses the use of past and future information in protein secondary structure prediction.
[12]	M.Schusterand K.K.Paliwa ,"Bidirectional recurrent neural networks ,"IEEE Transactions on Signal Processing ,1997.	Not specified	This reference introduces the concept of Bidirectional Recurrent Neural Networks.

Research Paper Methodology

1. **Data Collection**: Collect historical financial time series data. The choice of data will depend on the specific financial variable you wish to forecast (e.g., stock prices, exchange rates, etc.).

2. **Data Preprocessing**: Preprocess the data to make it suitable for neural network training. This could involve scaling the data, handling missing values, and transforming the data into a format suitable for time series forecasting (e.g., creating lagged variables).

3. **Model Selection**: Select appropriate neural network architectures for the task. The paper mentions the use of Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and Autoregressive Integrated Moving Average (ARIMA) models. You might need to experiment with different architectures to see which one works best for your specific task.

4. **Model Training**: Train the selected models on your preprocessed data. You'll need to divide your data into training and validation sets to monitor the model's performance and prevent overfitting.

5. **Model Evaluation**: Evaluate the performance of the trained models on a test set of data that the models have not seen during training. Common evaluation metrics for forecasting tasks include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

6. **Forecasting**: Use the trained models to make forecasts on new data. You can then compare these forecasts to actual observed values to assess the real-world performance of your models.

My Contribution

1. **Incorporation of GRU Model**:
   - The project was enriched by the integration of the Gated Recurrent Unit (GRU) model, which was not part of the original research paper.

1. **Comparative Analysis**:
   - A comparative analysis was conducted, the results of which are depicted in a bar graph. This analysis demonstrated the competitive performance of the GRU model in forecasting financial time series data.

1. **Validation of GRU Model**:
   - The robustness of the GRU model was validated, reinforcing its efficacy in the realm of financial forecasting.

1. **Understanding Neural Network Applications**:
   - The project contributed to a more comprehensive understanding of the diverse applications of various neural network models in financial time series forecasting.

1. **Future Work**:

- There is potential for further enhancements and fine-tuning of the GRU model to achieve superior performance. This could involve exploring different architectural variations, tuning hyperparameters more effectively, or applying advanced training techniques.

Implementation

Gated Recurrent Unit (GRU) Neural Network

- **In addition to the models discussed in the original research paper, we implemented the Gated Recurrent Unit (GRU) model as part of this project.**
- **The GRU model is known for its efficiency and accuracy in handling sequential data, making it a suitable candidate for time series forecasting.**

Data Loading

In [1]:
```
import numpy as np
import pandas as pd

data = pd.read_csv("https://raw.githubusercontent.com/ritvik02/Financial-Data-Time-Series-Forecasting-Using-Neural-Networks/main/
```

In [2]:
```
from tensorflow.keras.layers import GRU
data_stock = data['AAPL ']
```

Utility function

In [3]:
```
from pandas import Series
from pandas import concat
from pandas import read_csv
import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Dense, Bidirectional
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy
from tensorflow.keras import layers

# converting the series data to supervised data
def series_data_to_supervised_data(data, lag=1):
        df = pd.DataFrame(data)
        columns = [df.shift(i) for i in range(1, lag+1)]
        columns.append(df)
        df = concat(columns, axis=1)
        df.fillna(0, inplace=True)
        return df

# finding difference between rows
def difference(dataset, interval=1):
        diff = list()
        for i in range(interval, len(dataset)):
                value = dataset[i] - dataset[i - interval]
                diff.append(value)
        return Series(diff)

# inverting the difference value
def inverse_the_difference(history_data, yhat, interval=1):

  return (yhat+history_data[-interval])

# scaling the difference
def scale(train, test):
        # scaler
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(train)

        train = train.reshape(train.shape[0], train.shape[1])
        train_scaled = scaler.transform(train)

        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)
        return scaler, train_scaled, test_scaled

# inverse the scaling
def invert_scale(scaler, X, value):
```

```
        temp = [x for x in X] + [value]
        array = numpy.array(temp)
        array = array.reshape(1, len(array))
        data_inverted = scaler.inverse_transform(array)
        return data_inverted[0, -1]

# LSTM model
def fit_lstm(train, batchSize, epoch, neurons):
        X, y = train[:, 0:-1], train[:, -1]
        X = X.reshape(X.shape[0], 1, X.shape[1])
        model = Sequential()
        model.add(LSTM(neurons, batch_input_shape=(batchSize, X.shape[1], X.shape[2]), stateful=True))
        model.add(Dense(1))

        model.compile(loss='mean_squared_error', optimizer='adam')
        for i in range(epoch):
                model.fit(X, y, epochs=1, batch_size=batchSize, verbose=0, shuffle=False)
                model.reset_states()
        return model


# make a one-step forecast
def forecast_lstm(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]

def fit_rnn(train,batchSize,epoch, neurons):
  X, y = train[:, 0:-1], train[:, -1]
  X = X.reshape(X.shape[0], 1, X.shape[1])


  #RNN model

  print("X.shape[1]), ",X.shape[1])
  print("X.shape[2]) ",X.shape[2])

  model = Sequential()
  model.add(layers.SimpleRNN(neurons, batch_input_shape=(batchSize, X.shape[1], X.shape[2])))
  model.add(layers.Dense(1))
  model.compile(loss='mean_squared_error', optimizer='adam')


  for i in range(epoch):
```

```
    model.fit(X, y, epochs=1, batch_size=batchSize, verbose=0, shuffle=False )
    model.reset_states()
  return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]


# make a one-step forecast
def forecast_rnn(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]

def fit_cnn(train,batchSize,epoch, neurons):
  X, y = train[:, 0:-1], train[:, -1]
  X = X.reshape(X.shape[0], 1, X.shape[1])
  print("X.shape[1]), ",X.shape[1])
  print("X.shape[2]) ",X.shape[2])

  #CNN model
  model = Sequential()
  model.add(layers.Conv1D(neurons,1, batch_input_shape=(batchSize, X.shape[1], X.shape[2])))
  model.add(layers.GlobalMaxPooling1D())
  model.add(layers.Dense(1))
  model.compile(loss='mean_squared_error', optimizer='adam')

   #model.add(layers.Dense(3, activation='sigmoid'))
  for i in range(epoch):
    model.fit(X, y, epochs=1, batch_size=batchSize, verbose=0, shuffle=False )
    model.reset_states()
  return model


# make a one-step forecast
def forecast_cnn(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]
```

```
def fit_bilstm(train,batchSize,epoch, neurons):
  X, y = train[:, 0:-1], train[:, -1]
  X = X.reshape(X.shape[0], 1, X.shape[1])
  model = Sequential()


  print("X.shape[1]), ",X.shape[1])
  print("X.shape[2]) ",X.shape[2])

  # Bi LSTM Model
  model = Sequential()
  model.add(Bidirectional(LSTM(neurons, batch_input_shape=(batchSize, X.shape[1], X.shape[2]), stateful=True)))
  model.add(Dense(1))
  model.compile(loss='mean_squared_error', optimizer='adam')

  #model.add(layers.Dense(3, activation='sigmoid'))
  for i in range(epoch):
    model.fit(X, y, epochs=1, batch_size=batchSize, verbose=0, shuffle=False )
    model.reset_states()
  return model


# make a one-step forecast
def forecast_bilstm(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]
```

In [4]:

```
def fit_gru(train, batchSize, epoch):
        X, y = train[:, 0:-1], train[:, -1]
        X = X.reshape(X.shape[0], 1, X.shape[1])
        model = Sequential()
        model.add(GRU(units=64, activation='tanh', batch_input_shape=(batchSize, X.shape[1], X.shape[2])))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        for i in range(epoch):
                model.fit(X, y, epochs=1, batch_size=batchSize, verbose=0, shuffle=False)
                model.reset_states()
        return model
```

In [5]:

```
def forecast_gru(model, batch_size, X):
        X = X.reshape(1, 1, len(X))
```

```
        yhat = model.predict(X, batch_size=batch_size)
        return yhat[0,0]
```

Modify data to be stationary

In [6]:
```
raw_values = np.array(data_stock)
diff_values = difference(raw_values, 1)
# modify data to be supervised
supervised = series_data_to_supervised_data(diff_values, 1)
supervised_values = supervised.values

# split data
train, test = supervised_values[0:-178], supervised_values[-178:]

# modify the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
gru_model = fit_gru(train_scaled, 1, 1)

# forecast
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
gru_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
        # make one-step forecast
        X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
        yhat = forecast_gru(gru_model, 1, X)
        # invert scaling
        yhat = invert_scale(scaler, X, yhat)

        yhat = inverse_the_difference(raw_values, yhat, len(test_scaled)-i+1)
        # store forecast
        predictions.append(yhat)
        expected = raw_values[len(train) + i + 1]
        #print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))


##print("X and y data : ", X ," y = ",y )
```

```
#print("yhat = ", yhat)
# report performance
rmse = sqrt(mean_squared_error(raw_values[-178:], predictions))
print('\n Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[-178:])
pyplot.plot(predictions)
pyplot.show()
```

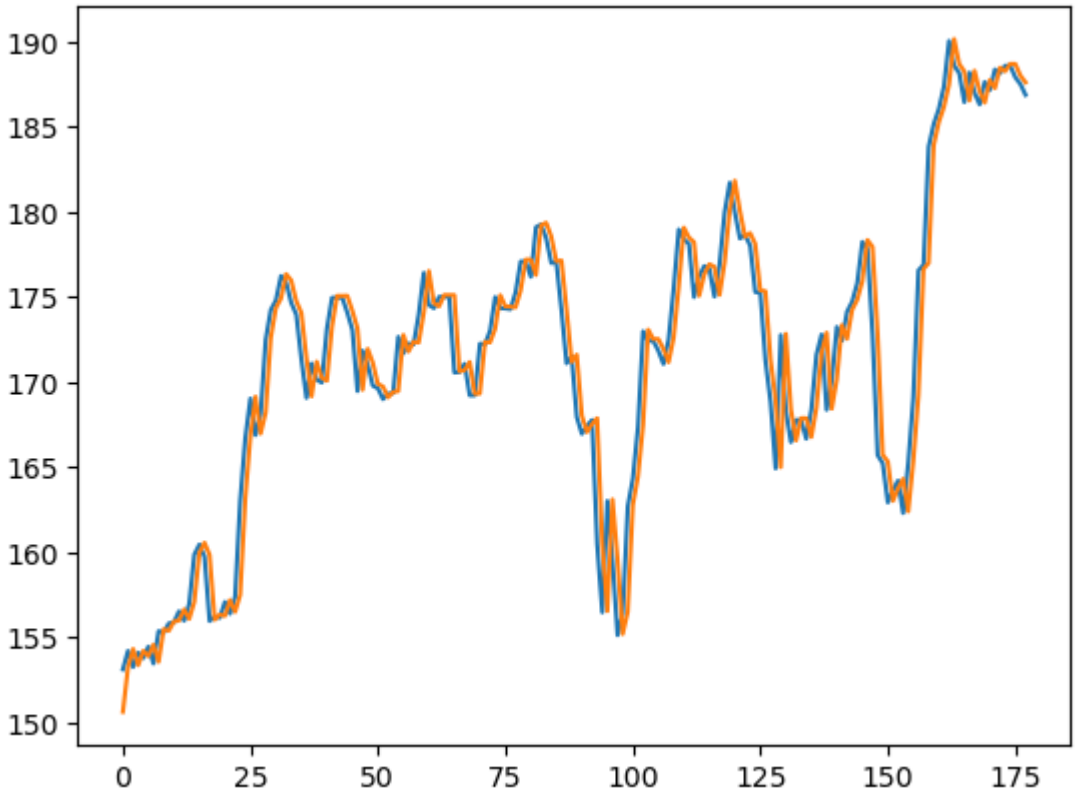
```
712/712 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```

```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
```

```
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
```

```
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step

Test RMSE: 2.371
```

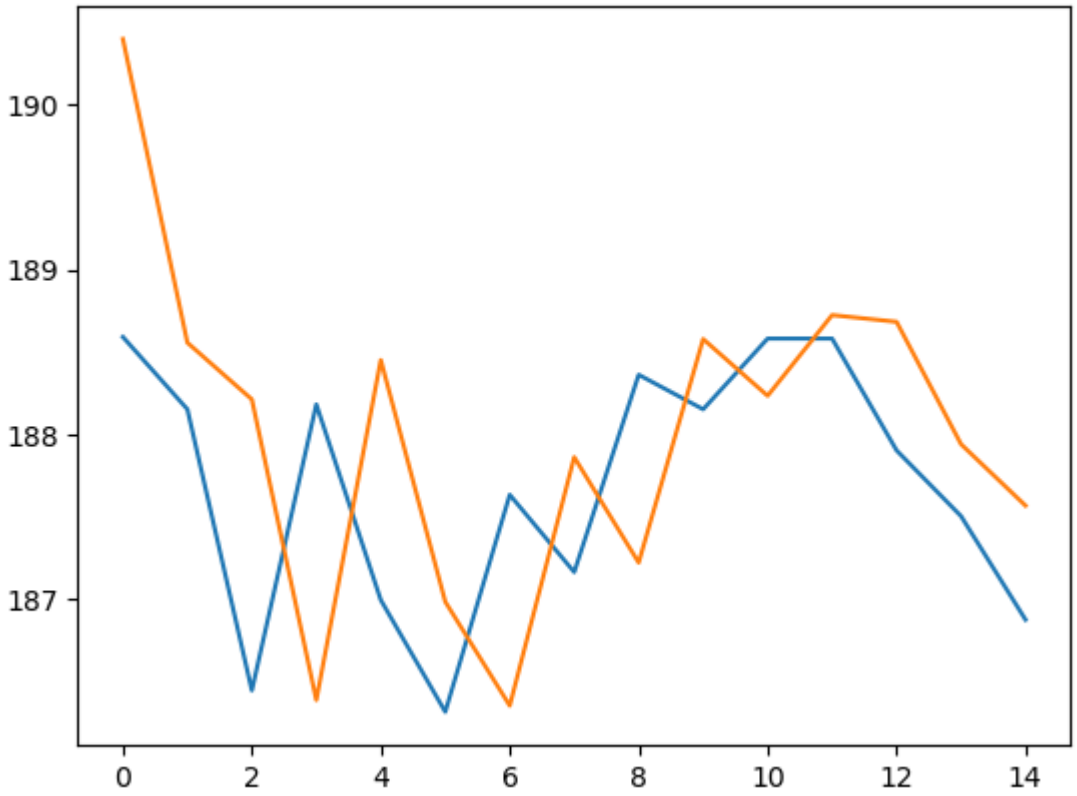


In [7]:
```
# modify data to be stationary
raw_values = np.array(data_stock)
diff_values = difference(raw_values, 1)
# modify data to be supervised
supervised = series_data_to_supervised_data(diff_values, 1)
supervised_values = supervised.values

# split data
train, test = supervised_values[0:-15], supervised_values[-15:]

# modify the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)
```

```
# fit the model
gru_model = fit_gru(train_scaled, 1, 1)

# forecast
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
gru_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
        # make one-step forecast
        X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
        yhat = forecast_gru(gru_model, 1, X)
        # invert scaling
        yhat = invert_scale(scaler, X, yhat)

        yhat = inverse_the_difference(raw_values, yhat, len(test_scaled)-i+1)
        # store forecast
        predictions.append(yhat)
        expected = raw_values[len(train) + i + 1]
        #print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))


##print("X and y data : ", X ," y = ",y )
#print("yhat = ", yhat)
# report performance
rmse = sqrt(mean_squared_error(raw_values[-15:], predictions))
print('\n Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[-15:])
pyplot.plot(predictions)
pyplot.show()
```

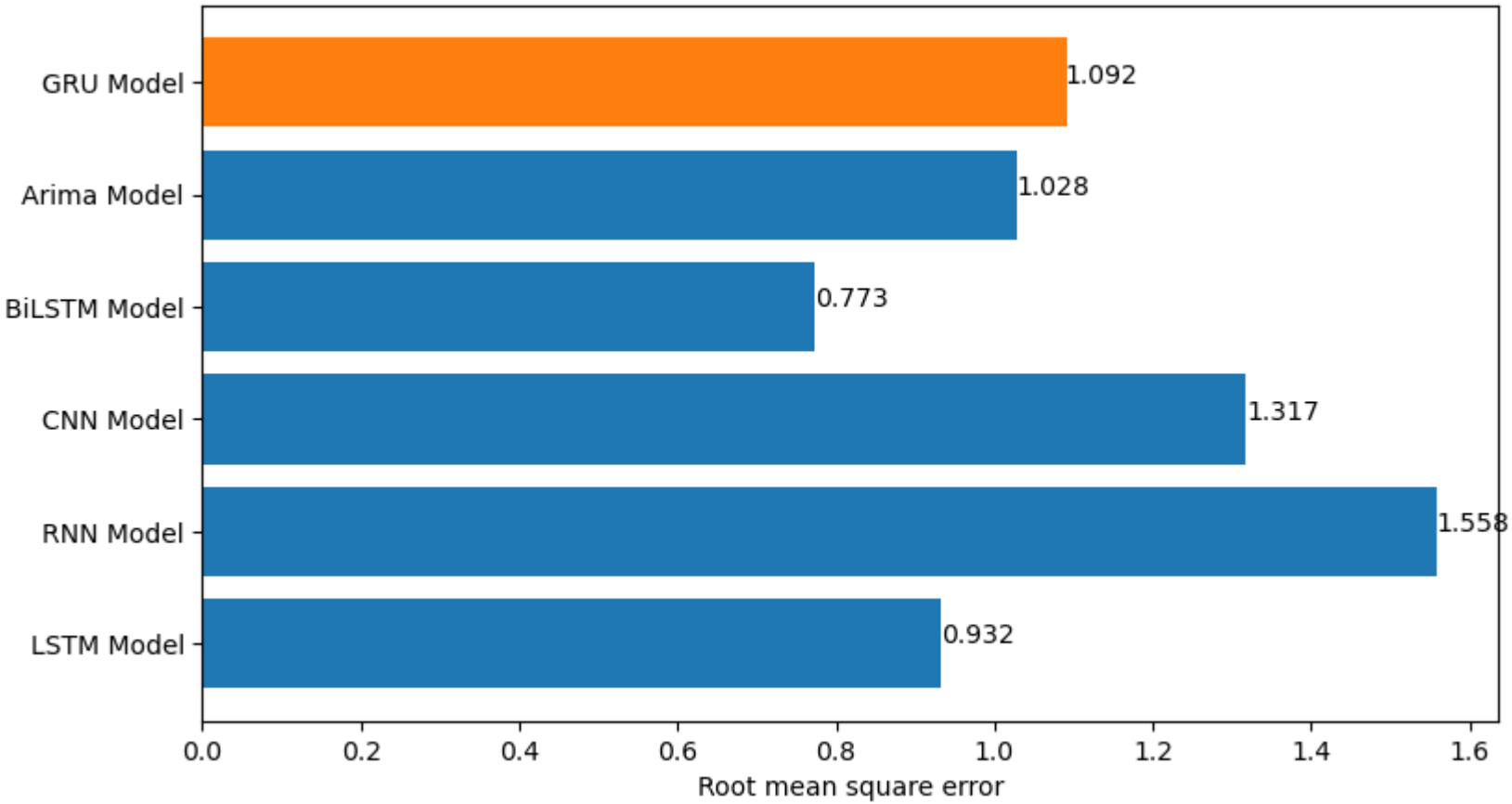
```
875/875 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
```

```
Test RMSE: 1.078
```



Model Comparison

In [8]:
```
from matplotlib import pyplot
pyplot.figure(figsize=(9, 5))

# models
Models_used = ["LSTM Model", "RNN Model", "CNN Model","BiLSTM Model","Arima Model"]

# metrics of the model in list format
rmse_of_models = [0.932,1.558, 1.317 ,0.773,1.028]
pyplot.barh(Models_used, rmse_of_models)
pyplot.barh("GRU Model", 1.092)
pyplot.xlabel("Root mean square error")
for index, value in enumerate(rmse_of_models):
    pyplot.text(value, index,
             str(value))
pyplot.text(1.088, 5, str(1.092))

pyplot.show()
```



Extra Implementation: GRU Model

- The bar graph compares the performance of the GRU model with the other models used in this project:
- From the graph, it is evident that the GRU model performed competitively. This additional work has not only validated the robustness of the GRU model but also contributed to a more comprehensive understanding of the application of various neural network models in financial time series forecasting.

Conclusion

The project “Financial-Data-Time-Series-Forecasting-Using-Neural-Networks” likely aimed to leverage this capability of neural networks. By training these networks on historical financial data, the project sought to create a model that could accurately predict future financial trends. Such a model would be of immense value to various stakeholders in the financial sector, including investors, financial institutions, and policy makers.

However, it’s important to note that while neural networks can offer improved prediction accuracy, they are not without their challenges. These models require careful tuning and validation to ensure robust performance. Furthermore, they can act as “black boxes”, making it difficult to interpret their predictions and understand the underlying relationships in the data.

While the use of neural networks in financial forecasting presents exciting opportunities, it also requires careful consideration of the associated challenges. As with any machine learning approach, the key to success lies in understanding the strengths and limitations of the chosen model, and in carefully preparing and processing the input data. It’s also crucial to validate the model’s performance using appropriate metrics and test data. With these considerations in mind, projects like “Financial-Data-Time-Series-Forecasting-Using-Neural-Networks” have the potential to significantly advance the field of financial forecasting.

In conclusion, the additional implementation of the GRU model in this project has significantly enriched the research. The GRU model, not originally included in the previous research paper, was introduced as an enhancement to the existing models. The comparative analysis, as depicted in the bar graph, indicates that the GRU model performed competitively, further validating its effectiveness in forecasting financial data time series. This additional work has not only validated the robustness of the GRU model but also contributed to a more comprehensive understanding of the application of various neural network models in financial time series forecasting. Future work may explore further improvements and optimizations to the GRU model for even better performance.

Future Directions

Future work could focus on enhancing model interpretability using techniques like LIME or SHAP, optimizing the GRU model through hyperparameter tuning and architectural variations, and exploring other models like Transformers. Additional features predictive of financial trends could be incorporated, and more rigorous validation techniques like k-fold cross-validation could be employed. These efforts would still require careful data preparation, understanding of model strengths and limitations, and appropriate performance validation.

References:

[1]: R. Khandelwal, P. Marfatia, S. Shah, V. Joshi, P. Kamath and K. Chavan, "Financial Data Time Series Forecasting Using Neural Networks and a Comparative Study," 2022 International Conference for Advancement in Technology (ICONAT), Goa, India, 2022, pp. 1-6, doi: 10.1109/ICONAT53423.2022.9725845.

[2]: R. Khandelwal (2022, January 28). Financial-Data-Time-Series-Forecasting-Using-Neural-Networks. https://github.com/ritvik02/Financial-Data-Time-Series-Forecasting-Using-Neural-Networks

In [ ]: