Netflix Stock Price Prediction Using LSTM

1. Importing Libraries :

- `numpy`, `pandas`, `matplotlib.pyplot`: Standard libraries for numerical computing, data manipulation, and plotting respectively.
- `MinMaxScaler` from `sklearn.preprocessing`: Used for feature scaling to normalize the dataset.
- `Sequential`, `LSTM`, `Dense` from `keras.models`: Components of the neural network model.
- `os`, `sys`, `warnings`: System and warning related utilities.

2.Suppressing Warnings:

- `warnings.filterwarnings("ignore")`: Suppresses all warnings to avoid cluttering the output.

3. Loading the Dataset :

- The code reads a CSV file ('NFLX.csv') containing historical stock data for Netflix.
- It extracts the 'Close' prices as these will be used for prediction.

4.Preprocessing Data :

- The 'Close' prices are converted into a numpy array and then scaled using `MinMaxScaler` to normalize the values between 0 and 1.

5.Splitting Dataset :

- The dataset is split into training and testing sets. Typically, 67% of the data is used for training (`train_size`) and the remaining for testing.

6.Creating Time Series Dataset :

- The function `create_dataset` is defined to create a time series dataset suitable for LSTM training. It takes a sequence of 'Close' prices and creates input-output pairs.

7. Reshaping Data:

- The input data is reshaped to fit the LSTM model's input requirements. LSTM expects input data in the shape `(samples, time steps, features)`.

8. Building the LSTM Model:

- A Sequential model is created.
- An LSTM layer with 4 memory units is added as the first layer. The `input_shape` parameter specifies the input shape.
- A Dense layer with one neuron is added as the output layer.
- The model is compiled with Mean Squared Error loss and Adam optimizer.

9. Training the Model :

- The model is trained using the training data (`trainX`, `trainY`) for 100 epochs with a batch size of 1.
- `verbose=0` parameter is used to suppress epoch-wise training progress output.

10. Making Predictions :

- The trained model is used to make predictions on both training and testing data.

11.Inverting Predictions :

- The scaled predictions are inverted back to the original scale using `scaler.inverse_transform`.

12. Plotting Results :

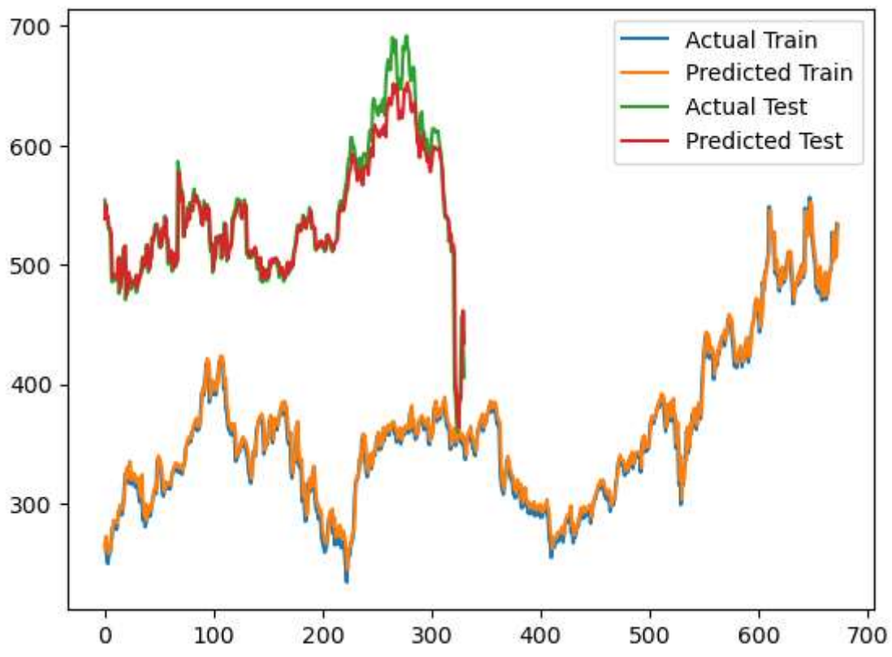
- The actual and predicted values for both training and testing data are plotted using `matplotlib.pyplot`.

This code demonstrates a simple LSTM-based time series forecasting model for predicting stock prices. The model is trained on historical data and then used to predict future stock prices. The visualization helps to assess the performance of the model by comparing actual and predicted values.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
import os
import sys
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")

# Load the dataset
data = pd.read_csv('NFLX.csv')

# Take only the closing prices for prediction
data = data[['Close']]
dataset = data.values.astype('float32')

# Normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# Split the data into training and testing sets
train_size = int(len(dataset) * 0.67)
train, test = dataset[:train_size], dataset[train_size:]

# Function to create the dataset with look back
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

```
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# Reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Build the LSTM model
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=0)  # Set verbose=0 to hide epoch output

# Make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# Invert predictions to original scale
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# Plot the results
plt.plot(trainY[0], label='Actual Train')
plt.plot(trainPredict[:,0], label='Predicted Train')
plt.plot(testY[0], label='Actual Test')
plt.plot(testPredict[:,0], label='Predicted Test')
plt.legend()
plt.show()
```

```
22/22 [==============================] - 1s 3ms/step
11/11 [==============================] - 0s 2ms/step
```



In [ ]: