
TDS3851 - Machine Learning

# Univariate Time-Series Forecasting on Netflix Closing Stock price

Group ID: ML7_T2120

Lecturer: Ts. Dr. Ramakrishnan Kannan

**Project Members:**


| Student Name | Student ID | Specialisation | Model |
|---|---|---|---|
| Nurina Izzati Binti Mohamad Jiwa | 1171103938 | Data Science | Linear Regression |
| Azib Jazman Bin Azmawi | 1181100438 | Data Science | Multilayer Perceptron |
| Muhammad Najmi Arif Bin Mohd Rosley | 1181100589 | Data Science | Long Short-Term Memory |

# Abstract

The stock market has always been a vital part of the economy because it allows individuals and companies to trade stocks, which helps the economy thrive. With the rise in stock market investment in recent years, it's more important than ever to look ahead to the future stock market as forecasting time-series models may assist buyers to decide when is the best time to invest. This study proposed univariate time-series forecasting on the Netflix closing price from year range of 2002 to 2021 using different neural network model and deep learning model such as Linear Regression with Neural Network, Multi-Layered Perceptron (MLP), and Long Short-Term Memory (LSTM) using Keras architecture. Hyperparameter optimization was handled to improve the performance of the models. The results showed that the final tuned model improves all models by reducing the number of errors, with Linear Regression having the best performance for both the original dataset (2002 - 2021) with MAE and MSE of 0.0141 and 0.0004 respectively and the 2016 dataset (2016 - 2021) with 0.0256 and 0.0012 respectively. The learning curve also shows better results after tuning the models as the validation loss starts to drop and shows stability while reducing the gap with the training loss.

# Table of Contents

# List of Figures

# List of Tables

# Abbreviations

| Abbreviation | Meaning |
|---|---|
| DNN | Deep Neural Network |
| SET | Stock Exchange of Thailand |
| CNN | Convolutional Neural Networks |
| LR | Linear Regression |
| MLP | Multi-Layered Perceptron |
| LSTM | Long Short-Term Memory |
| MAE | Mean Absolute Error |
| MSE | Mean Squared Error |
| CSV | Comma-Separated Values |
| kB | kiloBytes |
| EDA | Exploratory Data Analysis |
| ReLU | Rectified Linear Activation |
| RNN | Recurrent Neural Network |
| SGD | Stochastic Gradient Descent |
| Adam | Adaptive Moment Estimation |
| AdaGrad | Adaptative Gradient |
| RMSProp | Root Mean Squared Propagation |

# Section 1: Introduction

A stock market, sometimes known as a share market, is a loose network of economic transactions involving stocks (also known as shares), which reflect ownership claims on enterprises including securities listed on a public or private stock exchange. Predicting the future behaviour of stock prices has long been one of the most challenging tasks in this domain as it is affected by various factors (Mehtab & Sen, 2020). It is also deemed to be very important as the right expectation of stocks can assist investors to best allocate their assets and prompt colossal benefits for the vendor and the dealer (Budiharto, 2021; Puja, Parghania, & Bhawnani, 2022). According to *Efficient Market Hypothesis*, stock market prices behave in a 'random walk,' and it is impossible to forecast where they will end up in the future. Since it had no significant proof of the validity of the assumption, this has caused researchers and investors to come up with a method for stock price forecasting.

Traditional strategies for predicting the stock price of a company have been introduced and are being utilised by other investors to anticipate the stock price of the company. Using a line chart as a guide, the technical analysis approach predicts the direction of prices by studying and examining historical market data such as closing and opening prices, trading volume, and the adjacent close values in order to predict future prices. The fundamental analysis emphasises the fundamental value of stocks and conducts a qualitative examination of external variables affecting the stock, such as the business profile and market situation. However, these existing methods have their limitations. Fundamental analysis is a tedious and time-consuming technique as multiple factors are taken into account for making predictions. Although technical analysis is a good method to forecast the stock price, it only helps traders make short term profits (Kumar & Ningombam, 2018). In addition to the nature of stock market data being huge in size and dimensionality, current models are not efficient enough to identify complex relationships in large data sets (Vijh, Chandola, Tikkiwal, & Kumar, 2020).

This is where machine learning, specifically deep learning models, comes into place. In the past few years, many machine learning techniques including Long Short-Term Memory (LSTM), Multi-Layered Perceptron (MLP) and Convolutional Neural Networks (CNN) have been utilised to successfully predict the stock price of an organisation. Shah, Campbell, and Zulkernine (2018) implemented a state-of-the-art LSTM model to successfully predict the closing price of the Indian BSE Sensex index based on 20 years' worth of past data (Shah, Campbell, & Zulkernine, 2018). When compared to a Deep neural network (DNN), LSTM was able to outperform DNN in making weekly predictions and able to identify underlying trends in the stock market. A study by Werawithayaset and Tritilanunt (2019), estimates the closing price of the Stock Exchange of Thailand (SET) stock by using MLP (Werawithayaset & Tritilanunt, 2019). They found out that their model had a high Mean Absolute Error (MAE) of 0.8 and concluded that it was not as efficient as they had thought. Mehtab and Sen (2020) implement various machine learning models and a CNN model to successfully predict the closing price index of the NIFTY 50 index values of the National Stock Exchange of India (Mehtab & Sen, 2020). They discovered that the performance of CNN was way higher than all the other machine learning models as deep learning models are much more capable of extracting and learning features of a huge training dataset.

As a result, in this project, we are interested in forecasting the closing stock price of a well-known entertainment company, Netflix, by using deep learning models such as Linear Regression (LR), MLP and LSTM to observe as well as compare the performance of every

model. Netflix is a subscription-based streaming service founded in 1997 that allows the public to stream television shows and movies on any device as long as it is connected to the internet. The company started with DVD rentals in 1999 and proceeded to provide streaming services through the Internet in 2007. Although it had been popular in America and Canada since 2010, the service started to widely expand to more than 190 countries in 2016 and had more than 200 million subscribers globally in 2021 (Hosch, 2022). Our Netflix Stock price dataset was taken from the Kaggle website.

The remainder of this paper is organised as follows: Section 2 describes the features, the cleaning techniques involved and an initial exploratory of our dataset. Section 3 provides the methodology of the three deep learning models used while Section 4 shows the implementation of the described models. Section 5 discusses the performance of those models with Section 6 concluding the paper and its future enhancements.

# Section 2: Dataset description, Pre-processing and Early Analysis

## Dataset Description

In this section, we would be describing the dataset that was used for this univariate time series forecasting project as well as some pre-processing techniques applied to the dataset for preparation in training the model.

For this project, we would be using the Netflix Stock dataset that is made publicly available to the people on the Kaggle website. The data was collected from Yahoo Finance, which is a website that provides financial information like stocks for the people. Our dataset was provided in a comma-separated values (CSV) file containing 4874 rows of stock data from 23rd May 2002 to 30th September 2021 with a total size file of 335.65 kiloBytes (kB). The dataset has 7 stock-related attributes which are Date, Open price, Close price, Adjusted Closing price, High, Low, and volume of the Netflix stock.

'Date' is essentially the date that the stock market operates. The 'Open' column represents the price at which the financial security opens in the market when trading begins. On the other hand, 'Close' refers to the last price of an individual stock when the stock exchange closed its shop for the day. It is also called the final buy-sell order executed between two traders. This usually occurs in the final seconds before the market is closed (Smigel, 2022). 'Adj Close' represents the adjusted closing price that amends a stock's closing price which is to reflect that stock's value after accounting for any corporate actions. Moreover, the 'High' attribute is the highest price at which a stock is traded during a certain period while the 'Low' attribute is the lowest price of that period. The details of attributes in the dataset are listed in Table 1 together with its type and possible range values.

Table 1: Attributes Details

| Attribute | Data Type | Range of Values |
|---|---|---|

| Date | String | 23th May 2002 to 30th September 2021 |
|---|---|---|
| Open | Continuous | 0.38 to 608 |
| Close | Continuous | 0.37 to 610 |
| Adj Close | Continuous | 0.37 to 610 |
| High | Continuous | 0.41 to 619 |
| Low | Continuous | 0.35 to 608 |

## Data Pre-Processing

This section describes the pre-processing steps that were taken to ensure that the data was ready for model training. The first step would be to switch from string to datetime as the date data type. This is done to ensure that the data type is more suitable for plotting time series data and is easier to forecast time series data. We also check to see if there are any null values in our dataset for each column. There were no missing values in any of the features in the dataset that we used.

For this project, we are performing univariate time-series forecasting, which means we will only be using one feature to predict the future value of the closing price of a stock. Therefore, we decided to remove the 'Open', 'Adj Close', 'High' and 'Low' columns from our dataset as we will not use these features in our project. The close price data is then normalised using a Min-Max scaler to ensure that the data has a uniform scale. This is to ensure that there are no biases formed towards larger values during model training. The aforementioned steps are illustrated in Figure 1.

```
#Normalize our feature
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler(feature_range = (0, 1))

#Original dataset
close = netflixData[['Close']] #As we're only doing univariate, our only feature would be the Close col
close = scaler.fit_transform(close)
netflixData['Close'] = close

close_price = netflixData['Close']
close_price.shape

#Dataset starting from 2016
close2016 = netflixData2016[['Close']]
close2016 = scaler.fit_transform(close2016)
netflixData2016['Close'] = close2016

close_price2016 = netflixData2016['Close']
close_price2016.shape
```

Figure 1 Pre-Processing techniques implemented on the Netflix Stock dataset

## Exploratory Data Analysis

The next step would be to perform an Exploratory Data Analysis (EDA) on the features in our dataset. Since this project only focuses on forecasting the close price of Netflix stock data, we did an initial exploration of the 'Close' attribute.

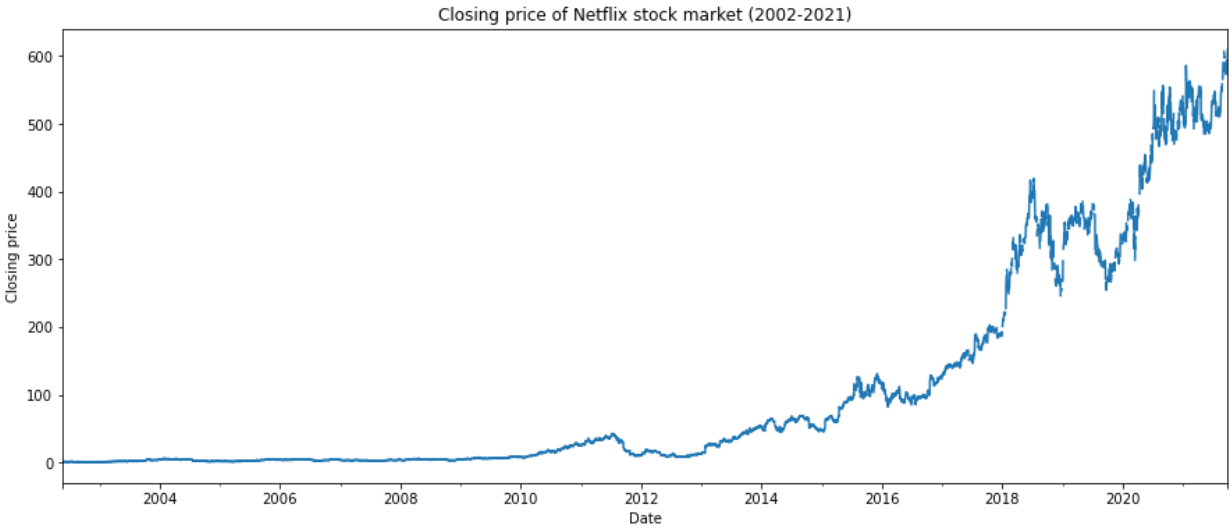


Figure 2 Netflix Stock Closing Price (2002 - 2021)

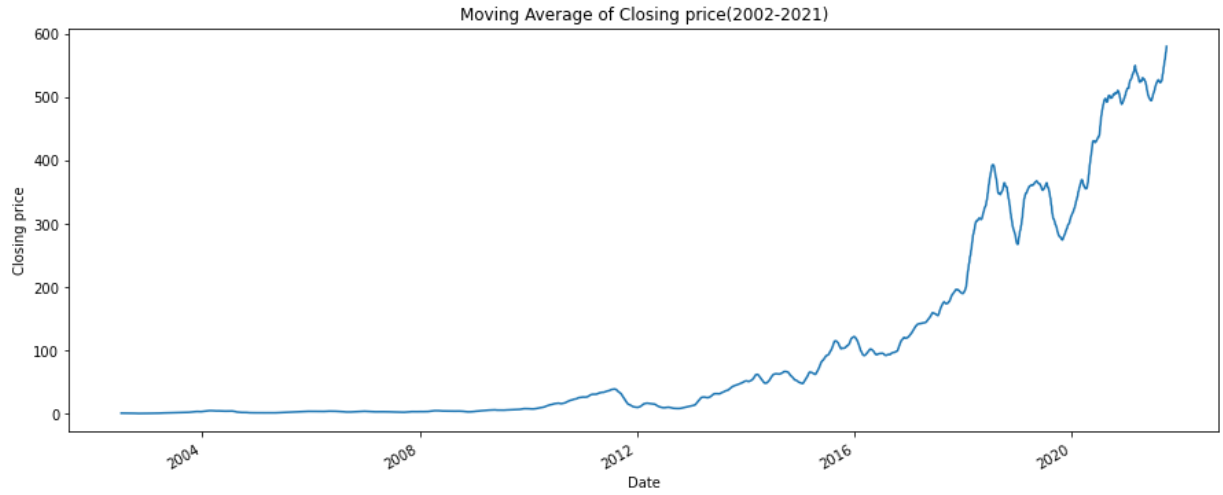


Figure 3 Netflix Stock Moving Average Closing Price (2002 - 2021)

Figure 2 shows the distribution of the closing price of Netflix stock while Figure 3 shows the moving average of the closing price of Netflix stock price for every 30 days. In the early years of Netflix, the closing price of its stock market was lower than $200. Starting in 2016, its stock price gradually increased up until 2021. This is due to the booming of streaming services which makes Netflix expand its services to 130 countries worldwide (VILLARREAL, 2016). In addition to the massive original movies and series being released such as 'Stranger Things' and 'Orange is the New Black', Netflix had seen an increase in stock market price between 2017 to 2018. With the Covid-19 pandemic in 2020, Netflix had a spike in their stock market as more

people were being forced to stay at home and craved for home entertainment (Forbes, 2020). Upon further analysis, we also noticed that there were gaps in the data which indicates no data was recorded on that day. This is due to the stock market being closed on the weekends. Thus, no data was recorded.

# Section 3: Research Methodology and Models

## Workflow Diagram

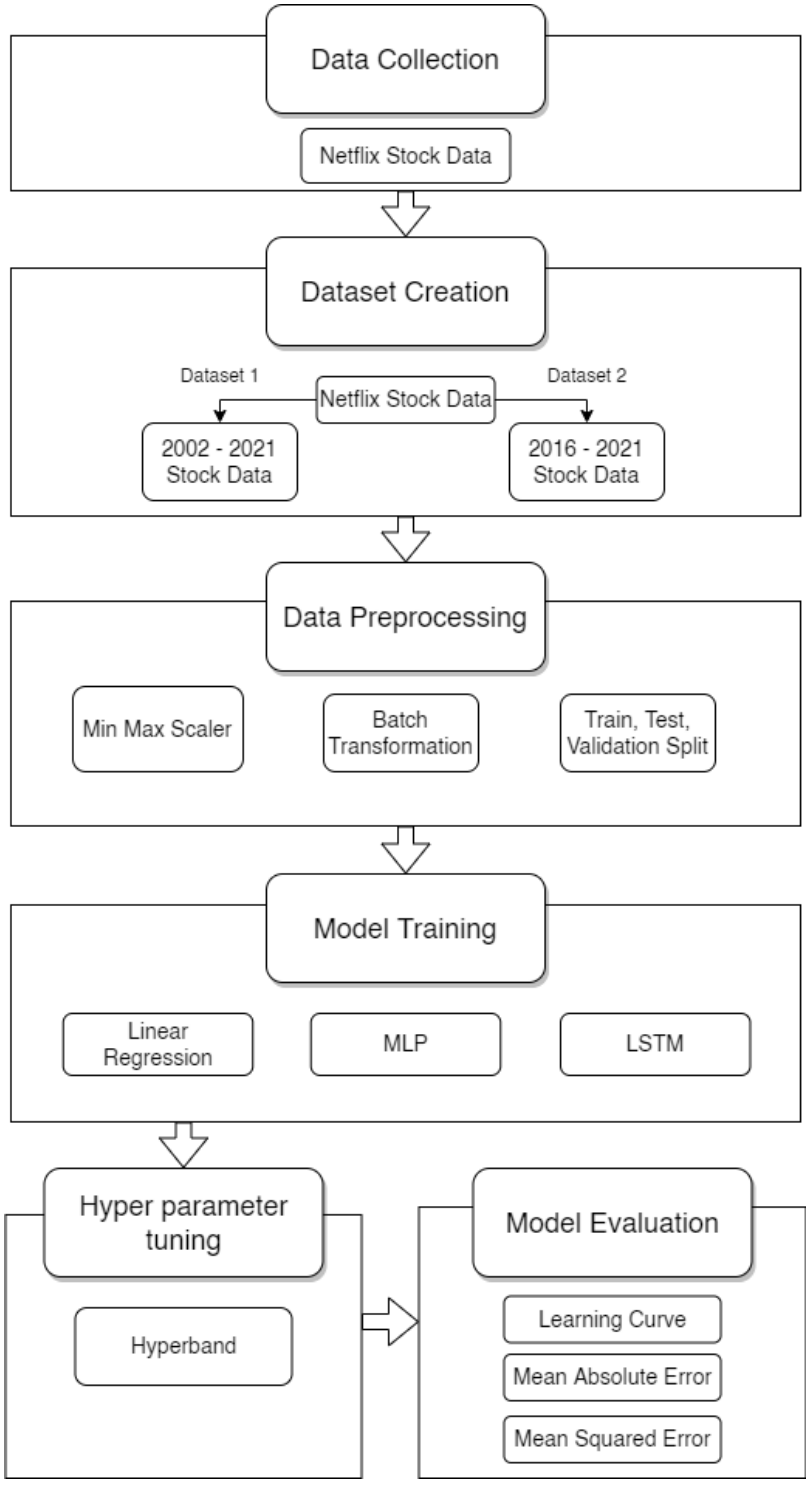


Figure 4 Research Framework

## Machine Learning Models

The first model used in this project is Linear Regression (LR) which is a regression machine learning model. Linear Regression was implemented using Keras architecture for this assignment as we want to utilise the neural network algorithm. Linear Regression with Neural Network architecture works by connecting a single neuron as the input layer and a single neuron as the output layer. The perceptron function accepts a single input only for linear regression. The model does not need any hidden layers to be input. Figure 5 below shows the linear activation function that would be set to the linear regression perceptron (Donthi, 2019).

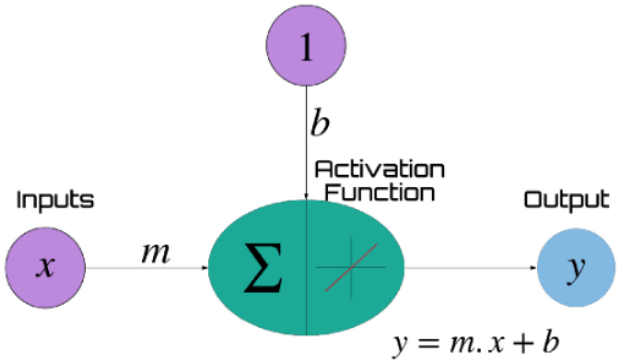


Figure 5 Linear Activation Function

The second deep learning model that we will be using in this project is a type of Neural Network called Multi-Layered Perceptron (MLP). MLP consists of a finite number of continuous layers and at least three layers are needed which are an input layer, hidden layer and output layer. An example of a MLP model is illustrated in Figure 6. Typically, MLP is trained using a supervised learning method called the backpropagation algorithm. The layers of MLP are fully connected, which means, every neuron in each layer is connected with all the neurons in the previous layer, and this connection represents a weight addition and sum. With backpropagation, it allows MLP to iteratively adjust the weights in the network reversely layer by layer until the error is reduced to an acceptable level (Gao, Zhang, & Yang, 2020; Bento, 2021). The curve of the Mean Squared Error (MSE) is calculated across all input and output pairs in every iteration. The first hidden layer's weights will then be modified with the gradient's value. This process will continue until the gradient for each input-output pair has converged, that is, the new computed gradient should only have a minor difference from the previous iteration. The nodes in MLP are neurons with a nonlinear activation function. The activation function specifies how the weighted sum of the input is non-linearly transformed into an output from a node or nodes in a network layer. As a result, the model can learn and perform more complex tasks (GeeksforGeeks, 2020). Typically, all hidden layers have the same activation function, whereas the output layer has a different activation function. Some of the common activation functions that have been used are Sigmoid, Softmax and Rectified Linear Activation (ReLU).

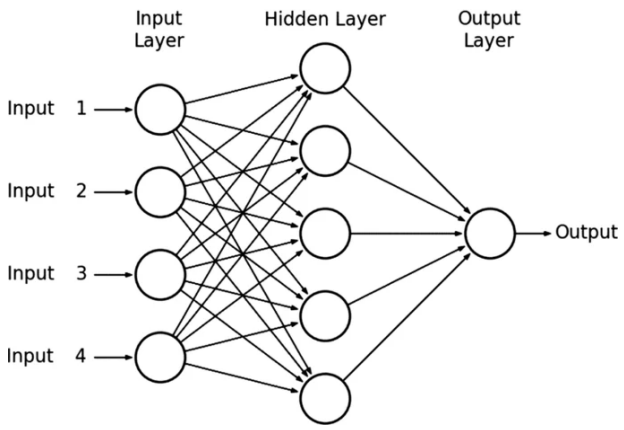


Figure 6 Sample Neural Network structure (Aghelpour & Varshavian, 2020)

The third deep learning model we will be using in this research is Long Short-Term Memory (LSTM). LSTM is a type of recurrent neural network (RNN). RNN is derived from a neural network in which a RNN has an internal memory of the previous inputs. The output of the previous inputs is cycled and used as inputs for future data. This makes RNN especially useful for forecasting sequential data. Although RNN is good at forecasting sequential data, it has its limitations. One such limitation is that RNN due to having only short-term memory makes it difficult to forecast data that has a very long sequence. To overcome this problem researchers have created LSTM. RNN modifies all its memory when a new input is available whereas LSTM chooses which part of the memory to remember and forget. LSTM flows information in a mechanism called a cell state which allows it to selectively manage the memory. A typical LSTM is made out of cells which are represented by the rectangles in Figure 7. LSTM managed its memory by using three mechanisms in the cells called gates. The three gates are the forget gate, the input gate and the output gate. The forget gate is responsible for removing data from the memory. It filters the input by the sigmoid function which returns a 0 or 1. If the output is 0 the input will be forgotten but if the output is 1 the input will be remembered in the memory. The input gate is responsible for adding new input to the memory by using a combination of tanh and sigmoid functions. Lastly, the output gate is responsible for filtering relevant information in the cell to be displayed as the outputs. This is done using a sigmoid function.

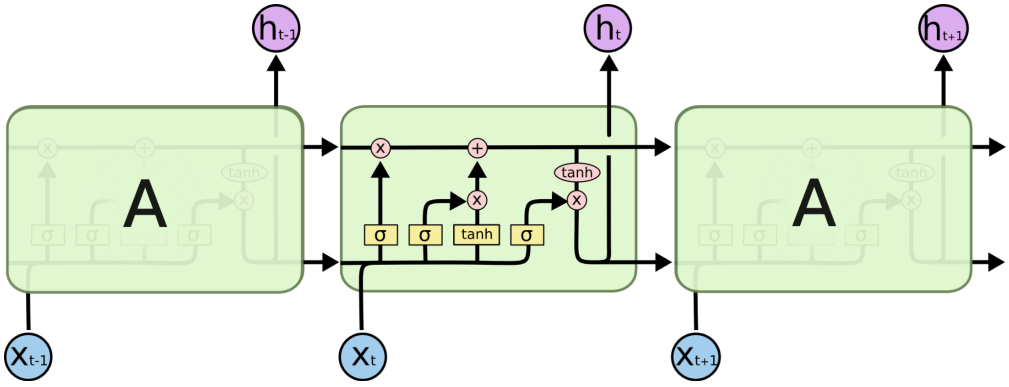


Figure 7 LSTM network structure

# Section 4: Initialization, Training and Tuning of Hyperparameters

## Data Splitting

For univariate time series forecasting, the data must be split into batches before the data can be used for training. In this project, we decided to use 5 days of past closing stock price to determine the closing price of the following day. Therefore, we split the data into batches of 5 days i.e., the window size to be 5 with the target being to predict the closing price of the following day. We created a function that will convert our dataset to a NumPy array and divide the Close price column into dependent (X) and independent (y) variables based on the window size. As such the data is transformed from a one-dimensional array to a three-dimensional array of size (4869, 5, 1). The code implementation is shown in Figure 8.

```
def df_to_X_y(df, window_size=5):
  df_as_np = df.to_numpy()
  X = []
  y = []
  for i in range(len(df_as_np)-window_size):
    row = [[a] for a in df_as_np[i:i+window_size]]
    X.append(row)
    label = df_as_np[i+window_size]
    y.append(label)
  return np.array(X), np.array(y)

WINDOW_SIZE = 5
X1, y1 = df_to_X_y(close_price, WINDOW_SIZE)
X2, y2 = df_to_X_y(close_price2016, WINDOW_SIZE)
X1.shape, y1.shape
```

Figure 8 Separate dataset into dependent and independent variables

From there, the array is further split into training, validating and testing datasets using the 70:15:15 ratio. The training dataset consists of the first 70% of the data while the validation and test datasets each contain 15% of the remaining data respectively. The data splitting was then applied to both 2002 Netflix stock data and 2016 Netflix stock data shown in Figure 9 and Figure 10.

```
# closePrice 2002
p70 = int(X1.shape[0] *70/100)
p15 = int(X1.shape[0] *15/100)

X_train, y_train = X1[:p70], y1[:p70]
X_val, y_val = X1[p70:p70+p15], y1[p70:p70+p15]
X_test, y_test = X1[p70+p15:], y1[p70+p15:]

X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape
```

```
((3408, 5, 1), (3408,), (730, 5, 1), (730,), (731, 5, 1), (731,))
```

Figure 9 Netflix Close Price 2002 Data Splitting

```
# closePrice 2016
p70 = int(X2.shape[0] *70/100)
p15 = int(X2.shape[0] *15/100)

X_train2, y_train2 = X2[:p70], y2[:p70]
X_val2, y_val2 = X2[p70:p70+p15], y2[p70:p70+p15]
X_test2, y_test2 = X2[p70+p15:], y2[p70+p15:]

X_train2.shape, y_train2.shape, X_val2.shape, y_val2.shape, X_test2.shape, y_test2.shape
```

```
((938, 5, 1), (938,), (201, 5, 1), (201,), (201, 5, 1), (201,))
```

Figure 10 Netflix Close Price 2016 Data Splitting

## Initialization & Training of models

As mentioned in Section 3, in this project we will be implementing 2 neural network models which are Linear Regression (LR), Multilayer Perceptron (MLP) and 1 deep learning model, Long Short-Term Memory (LSTM). For each model, we decided to create an initial model and compare its performance with the tuned models. The summary of our baseline models is illustrated in Table 2. All models were trained with different initial parameters and executed over 100 epochs to ensure the same number of times the training data passed. Then, we fit our models with the 2002 and 2016 Netflix datasets.

For linear regression, the initialization model was set by defining the Sequential class with one input layer with one node and one output layer with one node. The activation function was applied to both layers using a linear activation function to specify the model to produce a linear output. A flattening layer is inserted between the input and output layer to flatten the multi-dimensional by reshaping it into one dimensional. The initial optimizer chosen for the linear regression model is the SGD optimizer or Stochastic Gradient Descent, however, when inputting into the Adaptive Moment Estimation (Adam) optimizer, the model shows an improvement. Adam is a back-propagation algorithm that combines the advantages of AdaGrad and RMSProp to deal with sparse gradients and non-stationary objectives making its learning speed and convergence rate quite fast and efficient (Kumar & Ningombam, 2018). Hence, the baseline model of Linear Regression uses the Adam optimizer and the number of epochs is set to 100 epochs. The creation of our baseline LR model is shown in **Figure 11**.

```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

model = keras.models.Sequential([
              keras.layers.Dense(1, activation = "linear", input_shape = (5,1)), #input layer
              keras.layers.Flatten(), # flatten layer
              keras.layers.Dense(1, activation = "linear") #output layer
        ])
optimizer = keras.optimizers.Adam(lr=1e-5)
# Compile the model, optimizer use Adam
model.compile(loss=keras.losses.Huber(),
              optimizer = optimizer,
              metrics = ["mae"])
history = model.fit(X_train, y_train, epochs = 100) # train on 100 epochs
```

Figure 11 Linear Regression Baseline Model

Our initial MLP model is a Naive MLP model according to (PONRAJ, 2021). The naive MLP model has four dense layers with 50 neurons each, one layer with 10 neurons, a flattened layer and one fully connected layer as the output layer. The first four layers use the sigmoid activation while the fifth layer uses the softmax activation function. The results will then be flattened at the flatten layer in order to produce a single closing stock price output. For this model, we decided to use the Adam optimizer for the initial experiment with a learning rate of 10^-7 because it provides both the smart learning rate annealing and momentum behaviours of the algorithms (BILOGURE, 2018). Optimizers are search techniques used to update the weights in our model and are critical for achieving the highest possible accuracy or minimising loss. Moreover, the model is set to run for 100 epochs. The described model is created with keras.models as shown in **Figure 12**.

```
model = keras.models.Sequential([
  keras.layers.Dense(50, input_shape=(5,1)), #Dense layers
  keras.layers.Activation('sigmoid'),
  keras.layers.Dense(50),
  keras.layers.Activation('sigmoid'),
  keras.layers.Dense(50),
  keras.layers.Activation('sigmoid'),
  keras.layers.Dense(50),
  keras.layers.Activation('sigmoid'),
  keras.layers.Dense(10),
  keras.layers.Activation('softmax'),
  keras.layers.Flatten(),
  keras.layers.Dense(1)
])

lr_schedule = keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-7 * 10**(epoch / 20))
optimizer = keras.optimizers.Adam(lr=1e-7)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, callbacks=[lr_schedule])
```

Figure 12 MLP Baseline Model

The last model that we will be using to do univariate time series forecasting on Netflix stock data is to use LSTM. The architecture of our LSTM baseline model is made up of four LSTM layers and a dense layer as the output layer. Then, it will be trained and we will be comparing it to the hyperparameter tuned model. The code implementation to create our base LSTM model is illustrated in **Figure 13**. The baseline model consists of 5 layers. The first layer consists of a LSTM layer with 100 neurons which will serve as the input layer. The model has 3 hidden layers each with 100 neurons. Lastly, the fifth layer is the output layer which consists of a dense layer with 1 neuron. The dropout rate for the input and hidden layer is set to 0.3 to avoid overfitting. Adam optimizer was used and the model will run on a total of 100 epochs and with a batch size of 32.

```
# Baseline Model Definition
baseline = Sequential()

#add 1st lstm layer
baseline.add(LSTM(units = 100, return_sequences = True, input_shape = (X_train1.shape[1], 1)))
baseline.add(Dropout(rate = 0.3))

##add 2nd lstm layer: 50 neurons
baseline.add(LSTM(units = 100, return_sequences = True))
baseline.add(Dropout(rate = 0.3))

##add 3rd lstm layer
baseline.add(LSTM(units = 100, return_sequences = True))
baseline.add(Dropout(rate = 0.3))

##add 4th lstm layer
baseline.add(LSTM(units = 100, return_sequences = False))
baseline.add(Dropout(rate = 0.3))

##add output layer
baseline.add(Dense(units = 1))

baseline.compile(optimizer = 'adam', loss = 'mean_squared_error')

baseline.summary()

# Fit the whole data
history = baseline.fit(x = X_train1, y = y_train1,validation_data=(X_val1, y_val1), batch_size = 32, epochs = 100)
```

Figure 13 LSTM Baseline Model

Table 2: Baseline Model Summary

| Model | Number of neurons in a layer | Activation Function | Learning Rate | Optimizer | Epochs |
|---|---|---|---|---|---|
| Linear Regression | 1, 1 | Linear | 1e-5 | Adam | 100 |
| MLP | 50, 50, 50, 50, 10, 1 | Sigmoid, Softmax | 1e-7 | Adam | 100 |
| LSTM | 100,100,100, 100,1 | Linear | 1e-5 | Adam | 100 |

## Hyperparameter Tuning

In order to improve the performance of our models, we will perform hyperparameter tuning to all of our models. Hyperparameter tuning is usually implemented in order to identify the optimal values for every parameter, resulting in a better performance model. The chosen hyperparameter tuning method for this project is called Hyperband provided by Keras Tuner. Hyperband lets us set a range of values of each parameter by choosing the data types such as hp.Int which is a set range of integer values, hp.Float which is a set range of float values, and hp.Choice is a set range of selected values. A minimum value and maximum value of the

parameter's value were set for hp.Int and hp.Float while a set of string values was set for hp.Choice. The metric chosen for measuring the performance of the tuning will be using Mean Absolute Error (MAE) and Mean Squared Error (MSE) with MSE for the loss evaluation. In this project, each machine learning algorithm has its respective parameters that would go through the Hyperband tuning.

Aside from the learning rate of the optimizer, which we set to 1e-2, 1e-2, and 1e-4, there are not many parameters that can be tuned for Linear Regression. The details of the parameters to be tuned in the Linear Regression model are tabulated in Table 3.

Table 3: Linear Regression Tuned Parameter

| **Parameters** | **Method** | **Range Values** |
|---|---|---|
| Learning Rate | hp.Choice | 1e-2, 1e-3, 1e-4 |

For the MLP model, there are four parameters that we have decided to tune, which are the activation function type, the number of neurons in each layer, the number of hidden layers, the learning rate of the optimizer and the type of activation function consisting Sigmoid, Softmax, and ReLu. The number of neurons for each layer was tuned with range values of 50 to 200. The number of layers for MLP was set from 1 to 5 layers and the learning rate of the optimizer is tested using 1e-5, 1e-6, and 1e-7. Table 4 displays the MLP parameters details to be tuned.

Table 4: MLP Tuned Parameter

| **Parameters** | **Method** | **Range Values** |
|---|---|---|
| Activation Function | hp.Choice | sigmoid, softmax, relu |
| Number of neurons | hp.Int | 50 - 200 |
| Number of layers | hp.Int | 1 - 5 |
| Learning Rate | hp.Choice | 1e-5, 1e-6, 1e-7 |

For the LSTM model, there are three parameters that can be tuned which are the number of neurons in a layer, the dropout range, and the learning rate of the optimizer. Similar to MLP, we would want to know the suitable range for the number of neurons for each layer where we set the range from 10 to 100 number of neurons. The dropout values range from 0.0 to 0.05 and the learning rate function is tested using 1e-2, 1e-3, and 1e-4. Table 5 tabulates the LSTM parameters to be tuned.

Table 5: LSTM Tuned Parameter

| **Parameters** | **Method** | **Range Values** |
|---|---|---|
| Number of neurons | hp.Int | 10 - 100 |

| | | |
|---|---|---|
| Dropout | hp.Float | 0.0 - 0.5 |
| Learning Rate | hp.Choice | 1e-2, 1e-3, 1e-4 |

Following on is the process of Hyperband hyperparameter tuning where firstly, we created a "model_builder()" function where it will create a model for every possible combination of hyperparameters. The parameters to be tuned are specified by using the Hyperband or "hp" followed by its Hyperband type and the range of values that wanted to be tested for every parameter. Then, the list of values of the tuned parameters will be inserted into the model back in "model.compile" where it will be evaluated on MSE for loss and MAE and MSE for the evaluation metrics. Figure 14 shows an example of the Hyperband hyperparameter model builder for Linear Regression.

```
# Hyperparameter Tuning
def model_builder(hp):
    model = Sequential()

    model.add(layers.Dense(1, activation = "linear", input_shape = (5,1)))
    model.add(layers.Flatten())
    model.add(layers.Dense(1, activation = "linear"))

    #Tune learning rate from 1e-5, 1e-6, 1e-7 for optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-5, 1e-6, 1e-7])
    # Choose Adam Optimizer
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate),loss='mse', metrics=['mae', 'mse'])
    return model
```

Figure 14 Linear Regression Hyperparameter Model Builder

Afterwards, we created the Hyperband tuner which contains the model builder function to process the tuning shown in Figure 15. The tuner needs other necessary arguments for it to run such as the objective to optimise where we choose validation loss, the maximum epochs number of 250 which is higher than our expected number of epochs and the factor of 4 which is the reduction factor of the epochs number. The tuner also lets us set the directory to save the logs of every number of trials the model runs and assign the project name.

```
# Instantiate the tuner
tuner = kt.Hyperband(model_builder, # the hypermodel
                     objective='val_loss', # objective to optimize
                      max_epochs=250,
                      factor=4,
                      directory='dir',
                      overwrite=True,# directory to save logs
                      project_name='khyperband2')

tuner.search_space_summary()
```

Figure 15 Instantiate the Tuner

We can even look back at the summary of our tuner search space by using the "search_space_summary" function as shown in Figure 16.

```
Search space summary
Default search space size: 1
learning_rate (Choice)
{'default': 1e-05, 'conditions': [], 'values': [1e-05, 1e-06, 1e-07], 'ordered': True}
```

Figure 16 Search Space Summary for Linear Regression

Next, we set an early stopping function to monitor the model based on the MAE result and stop running on the train set to avoid overtraining issues. We then let the tuner start the search by inputting the train set and validating on the validation set displayed in Figure 17.

```
stop_early = tf.keras.callbacks.EarlyStopping(monitor='mae', patience=20)
# Perform hypertuning
tuner.search(X_train, y_train, epochs=200, validation_data=(X_val,y_val), callbacks=[stop_early])
```

Figure 17 Tuner Search Function

We used the "get_best_hyperparameters()" function to determine the optimal configuration value for each parameter generated by the Hyperband. We will then use the "build" function to create a model with the desired value, as illustrated in Figure 18.

```
best_hp=tuner.get_best_hyperparameters()[0]
h_model = tuner.hypermodel.build(best_hp)
h_model.get_config()
```

Figure 18 Get Optimal Hyperparameters Values

# Section 5: Performance evaluation, comparison and discussion

For this assignment, we decided to evaluate the performance of the models by using error metrics. Error metric is a metric evaluation to measure and evaluate error values from a forecast model. We decided to choose common error metrics that were used by many researchers which are the Mean Absolute Error (MAE) and Mean Squared Error (MSE) to evaluate all of our three models, (Aijaz & Agarwal, 2020; Jedox, 2021).

MAE is one of the most preferred metrics for forecasting where it calculates the absolute difference value between the forecast error and the actual values. The MAE equation can be observed in Figure 19 where it represents the error value as average. The yi represents the actual data and the $\tilde{y}_i$ represents the predicted data, i represents the instantaneous time and n is the total number of samples.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Figure 19 Equation of Mean Average Error (MAE)

MSE is the metric that shows the average squared difference between the forecast error and the actual values. This metric removes any small error such as negative values as the equation squared the errors, resulting in the value always staying positive. The MSE equation can be described below in Figure 20.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Figure 20 Equation of Mean Squared Error (MSE)

Table 6: MAE and MSE of every model trained on different datasets

| Dataset | Model | Mean Average Error (MAE) | Mean Squared Error (MSE) |
|---|---|---|---|
| 2002 - 2021 | Linear | 0.0226 | 0.0009 |
| | **Tuned Linear** | **0.0141** | **0.0004** |
| | MLP | 0.3668 | 0.1663 |
| | Tuned MLP | 0.2127 | 0.0541 |
| | LSTM | 0.1849 | 0.0496 |
| | Tuned LSTM | 0.0974 | 0.0129 |
| 2016 - 2021 | Linear | 0.1442 | 0.0217 |
| | **Tuned Linear** | **0.0256** | **0.0012** |
| | MLP | 0.2293 | 0.0522 |
| | Tuned MLP | 0.0684 | 0.0055 |
| | LSTM | 0.1009 | 0.0113 |
| | Tuned LSTM | 0.0380 | 0.0020 |

The MAE and MSE scores of the baseline and tuned models, trained and tested using the original and 2016 datasets are shown in Table 6. In general, we found that all of our models were capable of delivering exceptional results, with MAE and MSE scores of around 0. Linear Regression fared the best among the baseline models, while MLP performed the worst, with MAE scores of 0.0226 and 0.3668, respectively, as well as MSE scores of 0.0009 and 0.1663. When using the original dataset, LSTM performed significantly better than MLP, with an MAE of 0.1849 and an MSE of 0.0496. We were able to significantly lower the score of the error metrics using hyperparameter optimization. In comparison to the baseline model, the tuned MLP had a 0.1 lower MAE and 0.11 lower MSE while the modified LSTM had a 0.0974 and 0.0129 reduction in MAE and MSE, respectively. Although our neural network and deep learning models already have low MAE and MSE, Linear Regression beats them with MAE and MSE of 0.0141 and 0.0004, correspondingly.

For the 2016 - 2021 dataset, all of the tuned models performed better compared to the baseline models with MLP showing the most improvement with a decrease in MAE from 0.2293 to 0.0684 and a decrease in MSE from 0.0522 to 0.0055. The best performing model for the 2016-2021 dataset is the tuned Linear Regression model with a MAE and MSE score of 0.0256 and 0.0012 respectively. For both datasets, it seems that the Linear Regression model performed the best with LSTM being a close second.

The performance of the Linear Regression model trained using the 2002 - 2021 dataset performed slightly better than the Linear Regression model trained using the 2016 - 2021 dataset. One possibility is that the amount of training data used was insufficient. Since the 2016 dataset only covers the period from 2016 to quarter 2020, it contained less data than the original dataset, which has 13 years of data from 2002 to 2015. When the training data set is small, the model is unable to fully absorb the information, reducing its performance in forecasting a stock's closing price and resulting in a higher error rate.

We also decided to utilise the training and validation loss learning curve as a metric to diagnose our models. A learning curve is usually used to observe the learning progress of our model over time by using a specific metric, which in our case, is a loss metric. The training loss indicates how well the model is fitting the training data while the validation loss indicates how well the model fits with the new data (Baeldung, 2020). Typically, a high loss value means the model is producing erroneous predictions while a low one indicates that the model forecasts it correctly. The shape and dynamics of a learning curve can be used to diagnose the behaviour of machine learning and helps determine if our models are underfitted, overfit or a good fit (Brownlee, 2019). Moreover, we can identify whether the training dataset is a good representative in relation to our validation dataset.

The learning curve of our Linear Regression model was displayed in Figure 21 for both the baseline model and tuned model for both the 2002 dataset and 2016 dataset. The validation loss is higher than the training loss in the baseline model with the 2002 dataset, with the validation loss decreasing as the number of epochs increased. The validation curve got closer to the training loss at 80 epochs, with a tiny gap between the two losses indicating slight overfitting. When tuning the Linear Regression model, the losses were reduced from 0.004 to 0.0005. However the validation loss is still higher with several spikes as the number of epochs increases. For the 2016 dataset, the learning curve of the validation loss is also higher than the training loss where both losses decreased as the number of epochs increased. However, the gap between the losses is wider than in the original dataset. This can also be observed based on the MAE and MSE result between the two datasets where the 2016 dataset have 0.1442 and 0.0217 results

respectively that are higher than the original dataset due to the small training data set. After tuning, the validation loss started to drop where during 20 epochs, the validation loss began to converge with the training loss. In terms of the number of losses, it seems that the Linear Regression model works better with the original dataset after we tuned it as the learning loss reduced compared to the 2016 dataset. However, for the learning curve, the tuned model with the 2016 dataset has a better learning curve fit as validation loss is almost as close as the training loss.

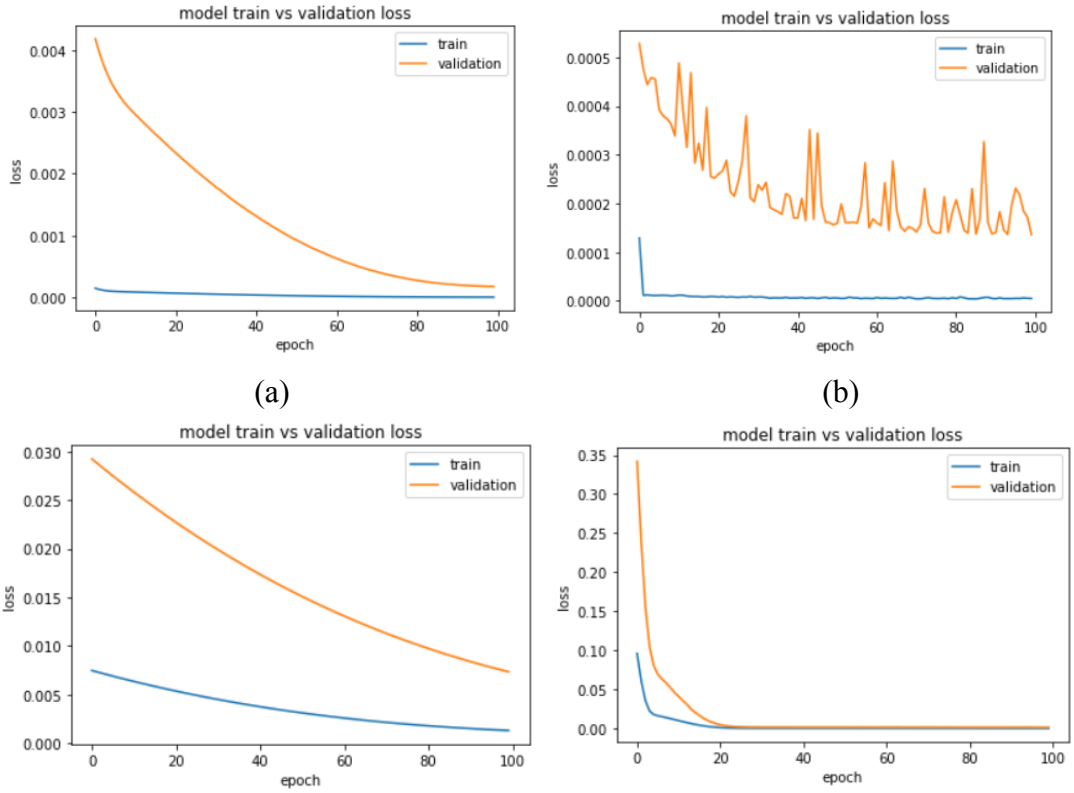


Figure 21 Training and Validation Loss for (a) Baseline Linear Regression (b) Tuned Linear Regression with original dataset and (c) Baseline Linear Regression (d) Tuned Linear Regression with 2016 dataset

The learning curve of our MLP model varies. We discovered the training loss of our baseline model with the original dataset in Figure 22 had very small to no changes through the number of epochs while its validation loss gradually decreased and went unstable after 60 epochs. Moreover, its validation loss is always higher than its training loss through the number of epochs indicating that our model is overfitting our data and may predict inaccurately on future data. This is in line with its high MAE and MSE of 0.3668 and 0.1663 respectively. The loss of the tuned version of the same model yields differently as the validation loss decreases at around 10 epochs and steadily follows the training loss with having smaller gap between losses indicating it is slightly overfit. However, its validation loss increased from 0.007 from baseline to 0.07 after tuning our model. In the case of our MLP that was trained with the 2016 dataset, the validation loss is still higher than training loss, but when at around 80 epochs, the validation loss started to get closer to train loss with still a gap in between. When tuned, our model shows

promising results as the validation loss gradually dropped up to a point of stability after 20 epochs and nearer to the training loss. Therefore, we can conclude from the learning curve that the tuned MLP model trained with the 2016 dataset fits the data well in comparison with the other MLP trials as the two loss values were able to converge with a minimal generalisation gap (Brownlee, 2019).

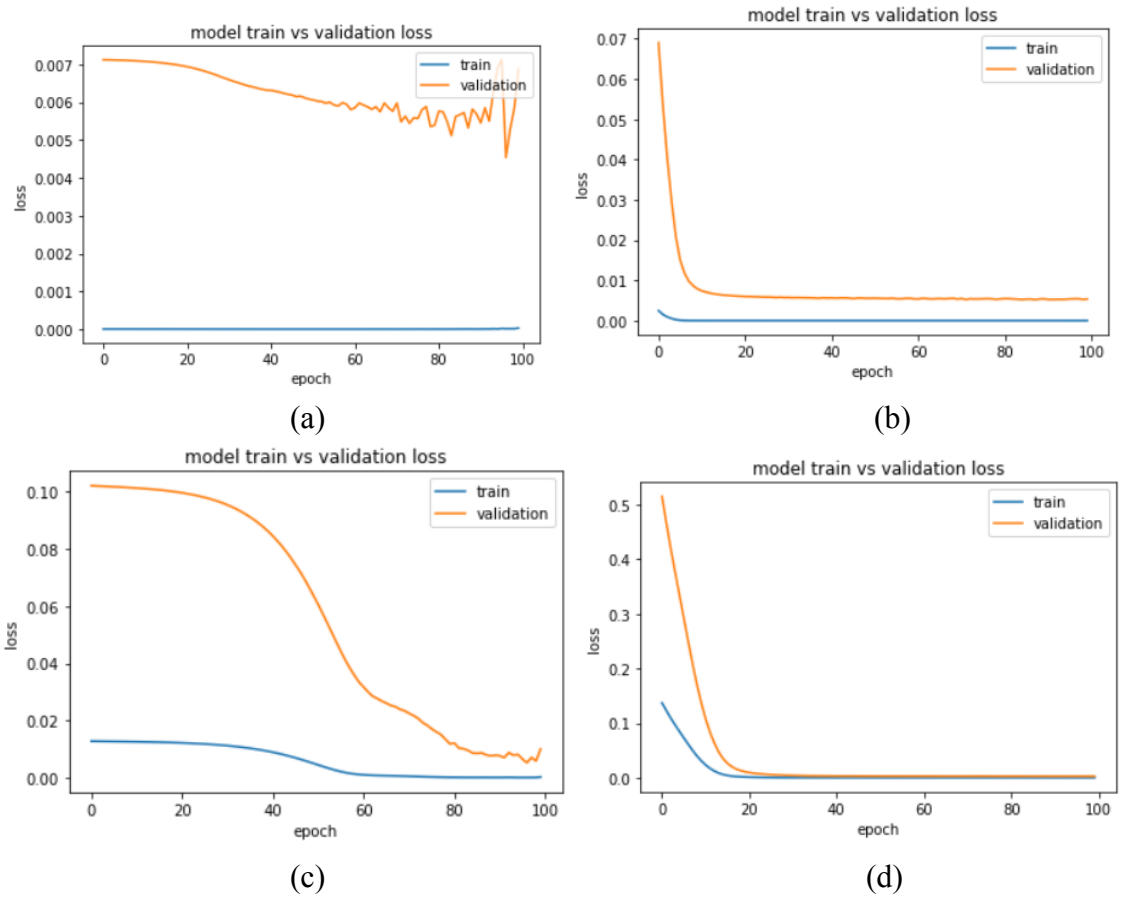


Figure 22 Training and Validation Loss for (a) Baseline MLP (b) Tuned MLP with original dataset and (c) Baseline MLP (d) Tuned MLP with 2016 dataset

Figure 23 illustrates the learning curve for the baseline and tuned LSTM models for both the 2002-2021 dataset and the 2016-2021 dataset. The learning curve for our LSTM models varies. Some of the graphs show the validation loss decreasing as the epochs increase while others show the validation loss increasing with the increase in epochs. For the 2002-2021 dataset, the validation loss for the baseline model shows an increase as the number of epochs increased as the learning curve is not really obvious. The baseline model shows signs of overfitting as validation loss is higher than the training loss. On the other hand, the tuned model showed better performance as the validation and training loss converge to a smaller area compared to the baseline model which shows signs of good fit. This is reflected by the MAE and MSE results where the tuned model shows a lower error with the values for the baseline and tuned models being (0.1849, 0.0496) and (0.0974, 0.0129) respectively. For the 2016-2021 dataset, on the other hand, the validation loss for both the baseline and tuned models decreases as the epochs increase. The baseline model shows signs of overfitting as the validation loss is higher compared

to the training loss. For the tuned model, the training curve shows signs of good fit as both the training and validation loss converge to the same spot. This is reflected in the MAE and MSE scores. The scores for the baseline model and tuned model are (0.1009, 0.0113) and (0.0380, 0.0020) respectively.

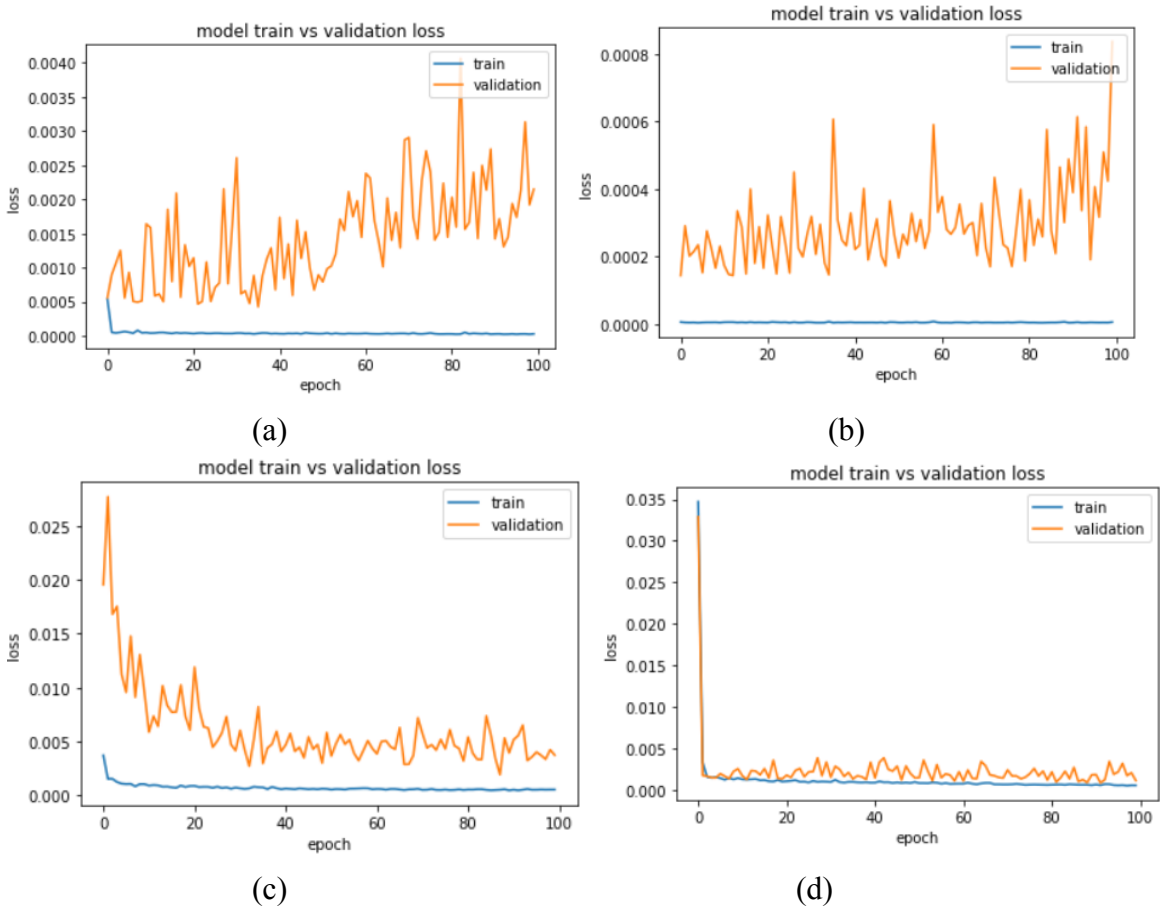


Figure 23 Training and Validation Loss for (a) Baseline LSTM (b) Tuned LSTM with original dataset and (c) Baseline LSTM (d) Tuned LSTM with 2016 dataset

To further support our decision, we decided to plot out the actual closing price of the Netflix dataset with the forecasted values produced by each model that was trained on the original dataset and 2016 as shown in Figure 24 and Figure 25 respectively. We were able to see that the Linear Regression output was generally more reliable and closer to the real value than the other two models when graphing the actual against the anticipated close price of the Netflix stock market on the test set of the original and 2016 dataset. In accordance with its high error rates, MLP's anticipated closing price appears to be the furthest from the actual value for both datasets. We noticed that the gap between real and predicted values produced by adjusted MLP trained on the original dataset (2002-2021) seemed to be increasing over time. If this is true, the error ratings for this model will rise in the future. Thus, producing inaccurate future closing stock prices.

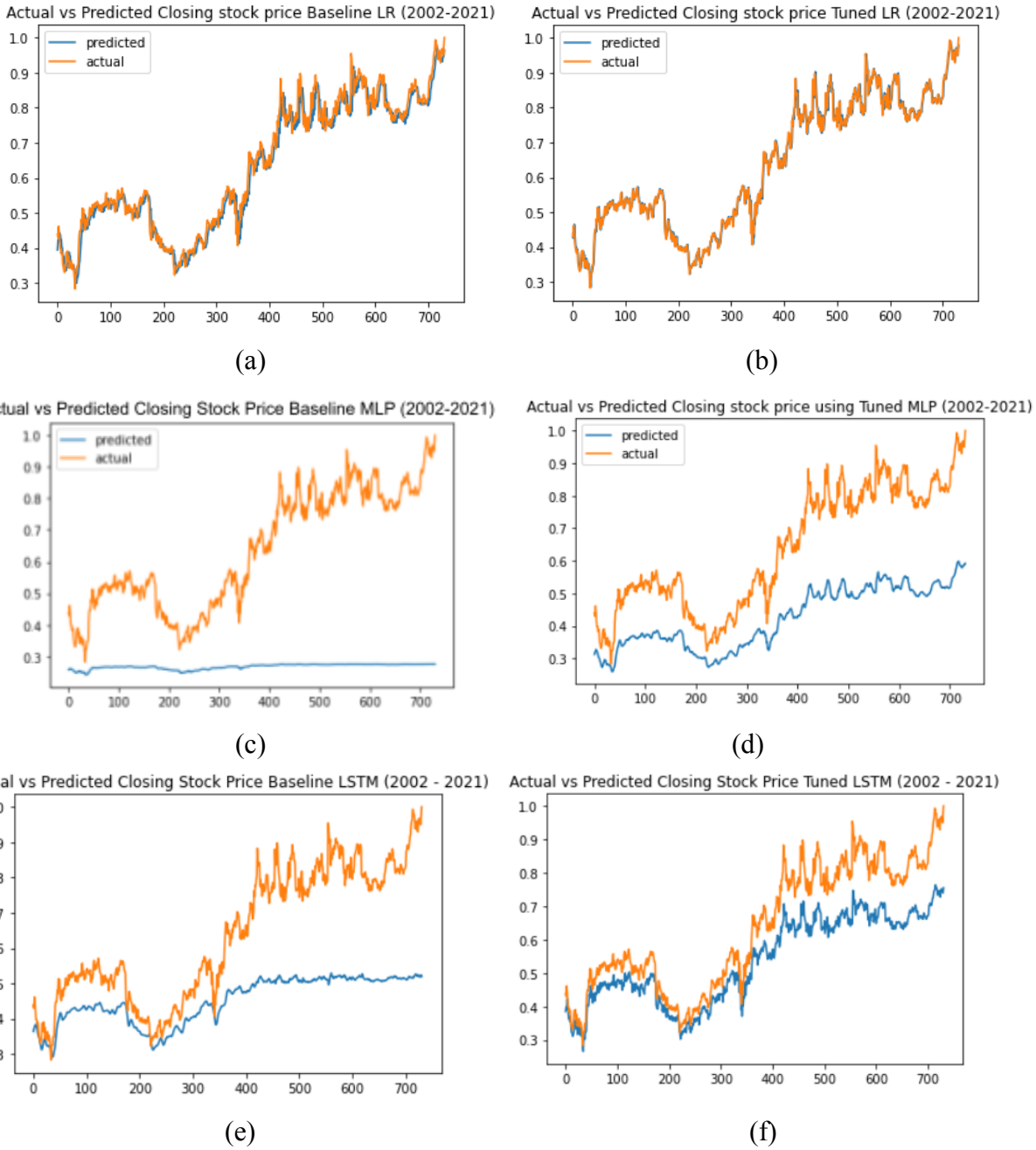


Figure 24 Actual Data versus Predicted Test Set Closing Price (2002 - 2021) with (a) Base LR (b) Tuned LR (c) Base MLP (d) Tuned MLP (e) Base LSTM and (f) Tuned LSTM

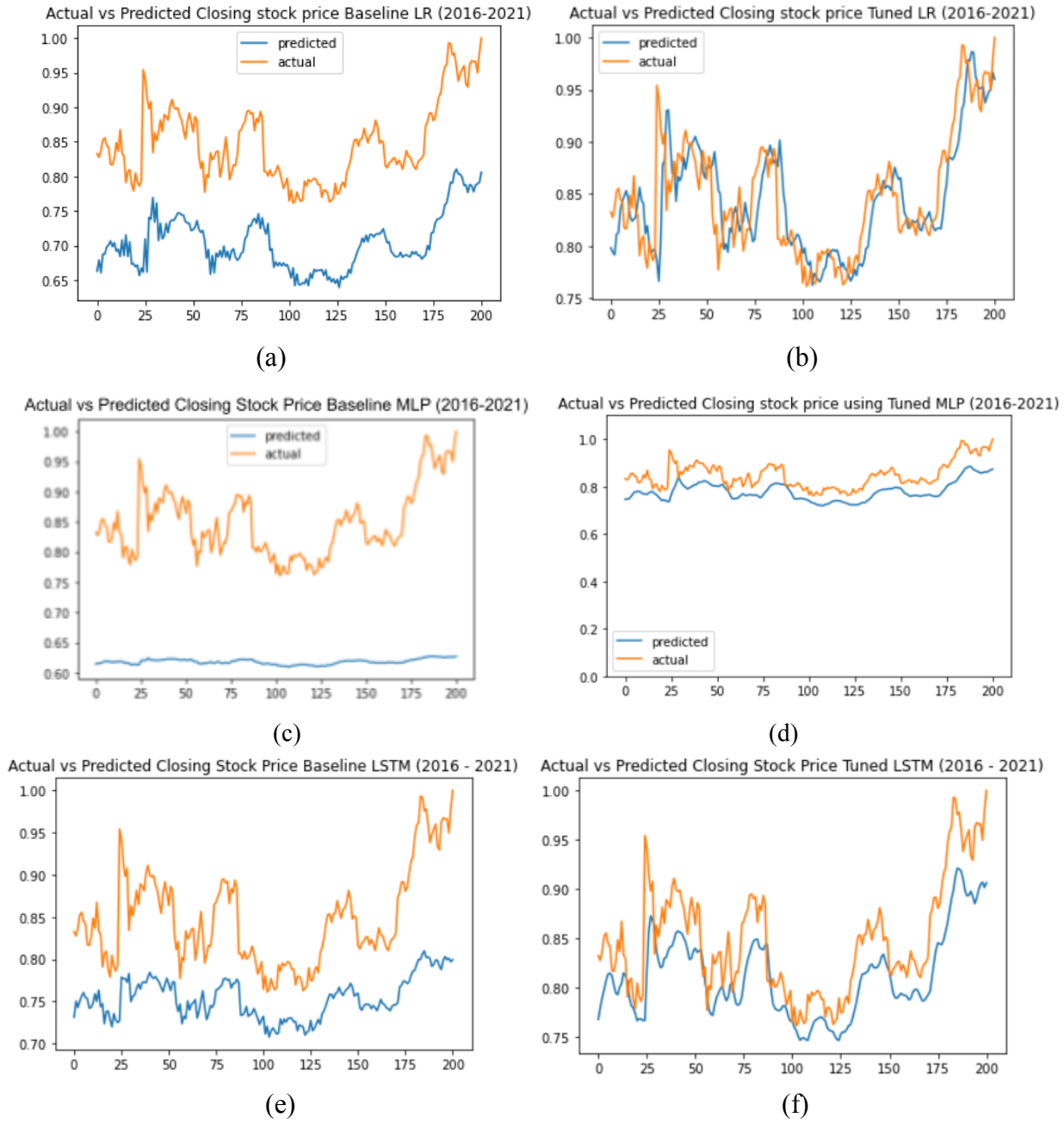


Figure 25 Actual Data versus Predicted Test Set Closing Price (2016 - 2021) with (a) Base LR (b) Tuned LR (c) Base MLP (d) Tuned MLP (e) Base LSTM and (f) Tuned LSTM

In addition, Linear Regression prediction's reliability can also be observed based on the outcome of the linear prediction fit line and the actual data as depicted in Figure 26. We can observe that the Linear Regression model on the original dataset has a better fit with a distinct linear pattern as the prediction line follows the actual data for both baseline model and tuned model; on the other hand, the 2016 dataset does follow the linear pattern, but the data are scattered far from the linear prediction line for both baseline and tuned models.

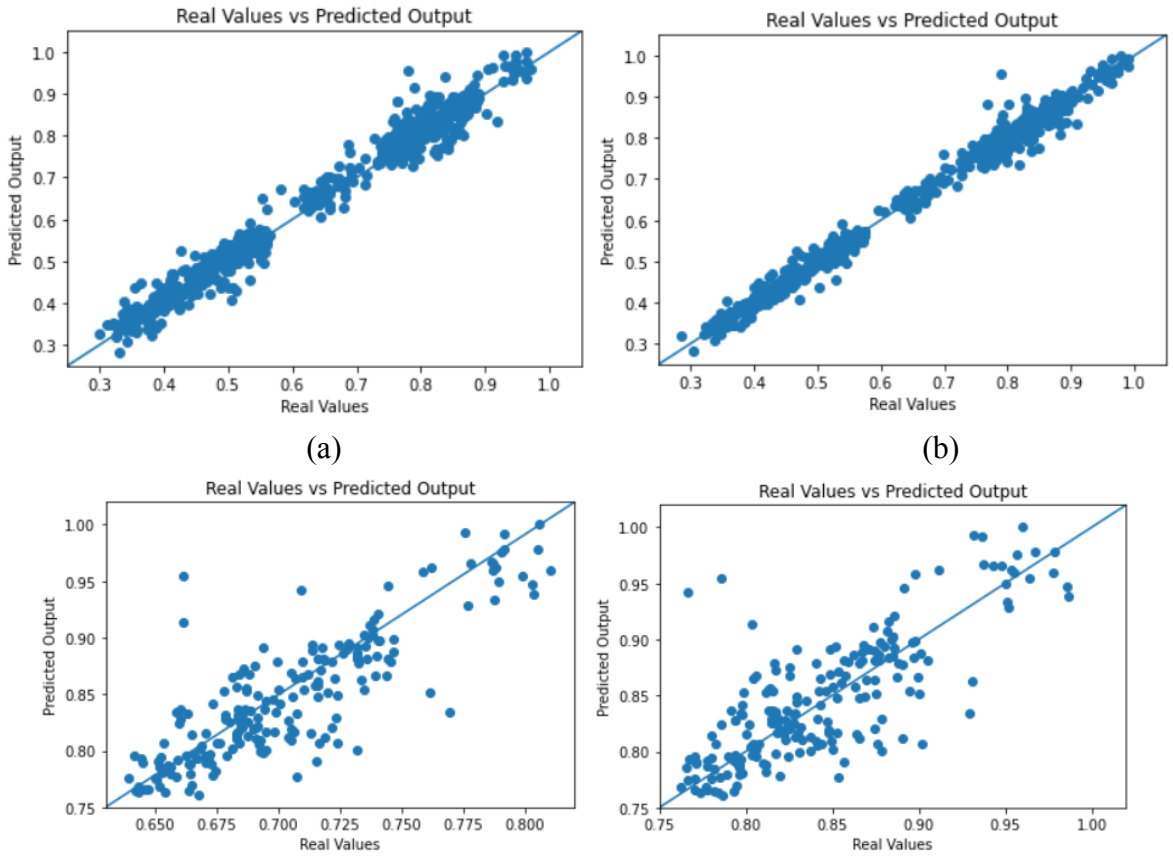


Figure 26 Linear Regression Real Values vs Predicted Output Plot with (a) Base LR (b) Tuned LR Plot (2002 - 2021) and (c) Base LR (d) Tuned LR Plot (2016 - 2021)

# Section 6: Conclusion and Future Work

To summarise, univariate time-series forecasting of the closing price of the stock market using machine learning has been on the rise in recent years. Although there were claims that it is impossible to predict the future stock price, no significant evidence has been put out to support this assumption. This encouraged investors to use statistical approaches to figure out the upcoming closing stock price. However, due to the ever-growing data volume, current models are insufficient to identify complex relationships in large data sets. There have been existing studies that determine the upcoming closing price of an organisation but there was minimal research that focuses on predicting the Netflix stock market. This is where the project is trying to fill in the gap in stock market forecasting.

In this assignment, we implemented Linear Regression, a type of neural network model called Multi-Layer Perceptron and a Long-Short Term Memory deep learning model to predict the closing price of Netflix stock. We trained every model with two different datasets: the original dataset and the 2016 dataset as we believe the stock price during the early years is irrelevant to predict the price in the current years. After performing hyperparameter tuning on the 3 models using Hyperband, we evaluate them by using the standard error metrics, Mean Absolute Error and Mean Squared Error as well as the learning curve of training and validation loss to determine the fitness of our models.

After exhaustive testing, we discovered that the base Linear Regression had already a low MAE and MSE rate when being trained with the original and the 2016 dataset. When we applied hyperparameter optimization, the tuned Linear Regression had the lowest MAE and MSE as compared to the other models with 0.0141 and 0.0004 respectively. In addition to its learning curve, we discovered that the validation loss of the tuned Linear Regression abruptly dropped up to a point of stability after 20 epochs and converged with the training loss as the number of epochs passed. To strengthen our evaluation, we discovered that the tuned Linear Regression model works better when being trained with the original dataset as its predicted values were almost identical to the actual closing price stock data. Here, we noticed that a simple machine learning model is able to outperform the other complex models when performing univariate time-series forecasting on the Netflix closing price.

In future work, we would like to extend our research to perform multivariate time-series forecasting (Iwok & Okpe, 2016). We would like to utilise other features that come together with the dataset to predict the closing price of the Netflix stock and see whether multivariate forecasting is able to outperform our univariate time series models. Moreover, we would like to use a hybrid model to better forecast the closing stock price. One of the model combinations that are well-known to yield accurate predictions were autoregressive integrated moving average (ARIMA) and Multi-Layered Perceptron (MLP) as well as LSTM and Convolutional Neural Network (CNN) (Khashei & Hajirahimi, 2018; Ni, Wang, & Cheng, 2021). By combining multiple single models, the hybrid models may have a lower error rate and produce much more accurate results rather than using each model separately to perform univariate forecasting on the closing price of the Netflix stock market.

# References


Aghelpour, P., & Varshavian, V. (2020). Evaluation of stochastic and artificial intelligence models in modeling and predicting of river daily flow time series. *Stoch Environ Res Risk Assess, 34*, 33-50. doi:https://doi.org/10.1007/s00477-019-01761-4

Aijaz, I., & Agarwal, P. (2020). A Study on Time Series Forecasting using Hybridization of Time Series Models and Neural Networks. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 13*(5), 827-832. Retrieved from https://www.researchgate.net/profile/Parul-Agarwal-7/publication/334155140_A_Study_on_Time_Series_Forecasting_using_Hybridization_of_Time_Series_Models_and_Neural_Networks/links/6001577ba6fdccdcb85512ea/A-Study-on-Time-Series-Forecasting-using-Hybridizati

Baeldung. (2020, July 30). *What is a Learning Curve in Machine Learning?* Retrieved from Baeldung: https://www.baeldung.com/cs/learning-curve-ml#:~:text=The%20training%20loss%20indicates%20how,the%20model%20fits%20new%20data

Bento, C. (2021, September 21). *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis*. Retrieved from Towards Data Science: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

BILOGURE, A. (2018, December 3). *Keras optimizers*. Retrieved from Kaggle: https://www.kaggle.com/code/residentmario/keras-optimizers/notebook

Brownlee, J. (2019, February 27). *How to use Learning Curves to Diagnose Machine Learning Model Performance*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Budiharto, W. (2021). Data science approach to stock prices forecasting in Indonesia during Covid-19 using Long Short-Term Memory (LSTM). *Journal of Big Data, 8*(1), 1-9. Retrieved from https://link.springer.com/article/10.1186/s40537-021-00430-0

Donthi, S. (2019, March 7). *Neural Networks with Numpy for Absolute Beginners—Part 2: Linear Regression*. Retrieved from KDnuggets: https://www.kdnuggets.com/2019/03/neural-networks-numpy-absolute-beginners-part-2-linear-regression.html

Forbes. (2020, April 2). *Netflix Stock Up 14% In 2020 At $375 Despite COVID-19; Is It Sustainable?* Retrieved from Forbes:

https://www.forbes.com/sites/greatspeculations/2020/04/02/netflix-stock-up-14-in-2020-at-375-despite-covid-19-is-it-sustainable/?sh=e9a351d30e41

Gao, P., Zhang, R., & Yang, X. (2020). The Application of Stock Index Price Prediction with Neural Network. *Mathematical and Computational Applications, 25*(3). doi:https://doi.org/10.3390/mca25030053

GeeksforGeeks. (2020, October 8). *Activation functions in Neural Networks*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/activation-functions-neural-networks/

Hosch, W. L. (2022, April 28). *Netflix | Founders, History, Shows, & Facts*. Retrieved from Britannica: https://www.britannica.com/topic/Netflix-Inc

Iwok, I. A., & Okpe, A. S. (2016). A Comparative Study between Univariate and Multivariate Linear Stationary Time Series Models. *American Journal of Mathematics and Statistics , 6*(5), 203-212. doi:10.5923/j.ajms.20160605.02

Jedox. (2021). *Error Metrics: How to Evaluate Your Forecasts*. Retrieved from Jedox: https://www.jedox.com/en/blog/error-metrics-how-to-evaluate-forecasts/

Khashei, M., & Hajirahimi, Z. (2018). A comparative study of series arima/mlp hybrid models for stock price forecasting. *Communications in Statistics - Simulation and Computation, 48*(9), 2625-2640. doi:https://doi.org/10.1080/03610918.2018.1458138

Kumar, S., & Ningombam, D. (2018). Short-Term Forecasting of Stock Prices Using Long Short Term Memory. *2018 International Conference on Information Technology (ICIT)*, (pp. 182-186). doi:10.1109/ICIT.2018.00046

Mehtab, S., & Sen, J. (2020). Stock Price Prediction Using Convolutional Neural. *arXiv preprint arXiv:2001.09769*, 7. Retrieved from https://arxiv.org/ftp/arxiv/papers/2001/2001.09769.pdf

Ni, H., Wang, S., & Cheng, P. (2021). A hybrid approach for stock trend prediction based on tweets embedding and historical prices. *Springer Link*, 849–868. Retrieved from https://link.springer.com/article/10.1007/s11280-021-00880-9

PONRAJ, A. (2021, December 22). *Advanced Hyperparameter Tuning of a Multilayer Perceptron – MLP*. Retrieved from DevSkrol: https://devskrol.com/2021/12/22/advanced-hyperparameter-tuning-of-a-multilayer-perceptron-mlp/

Puja, Y., Parghania, V., & Bhawnani, D. (2022). STOCK MARKET PREDICTION USING MACHINE LEARNING & PYTHON. *www.irjmets.com @International Research Journal of Modernization in Engineering, 4*(1). Retrieved from https://www.irjmets.com/uploadedfiles/paper/issue_1_january_2022/18336/final/fin_irjmets1642364724.pdf

Shah, D., Campbell, W., & Zulkernine, F. H. (2018). A Comparative Study of LSTM and DNN for Stock Market Forecasting. *2018 IEEE International Conference on Big Data (Big Data)* (pp. 4148-4155). IEEE.

Smigel, L. (2022, April 8). *What Is Open High Low Close in Stocks?* Retrieved from Analyzing Alpha: https://analyzingalpha.com/open-high-low-close-stocks/#:~:text=In%20stock%20trading%2C%20the%20high,total%20amount%20of%20trading%20activity

Vijh, M., Chandola, D., Tikkiwal, V. A., & Kumar, A. (2020). Stock Closing Price Prediction using Machine Learning Techniques. *International Conference on Computational Intelligence and Data Science (ICCIDS 2019). 167*, pp. 599-606. Procedia computer science. doi:https://doi.org/10.1016/j.procs.2020.03.326

VILLARREAL, Y. (2016, January 7). *Netflix announces huge global expansion, adding 130 more countries to its service*. Retrieved from Los Angeles Times: https://www.latimes.com/entertainment/envelope/cotown/la-et-ct-netflix-global-expansion-20160107-story.html#:~:text=Netflix%20Chief%20Executive%20Reed%20Hastings,electronics%20show%20in%20Las%20Vegas.&text=Streaming%20services%20giant%20Netflix%20took,the

Werawithayaset, P., & Tritilanunt, S. (2019). Stock Closing Price Prediction Using Machine Learning. *2019 17th International Conference on ICT and Knowledge Engineering (ICT&KE)* (pp. 1-8). IEEE. doi:10.1109/ICTKE47035.2019.8966836