American University of Armenia
College of Science and Engineering

Capstone Project

Comparison of LSTM and Multi-dimensional Convolutional Neural Networks for Time Series Forecasting

Supervisor: Arnak Poghosyan
Author: Arusyak Hakobyan
Spring, 2019

Abstract

This paper discusses time series forecasting, specifically, stock price prediction using neural networks. We first give an introduction to neural networks, time series prediction, financial analysis and give more detailed explanation of our problem setting. Next, we explore the dataset and its features and concentrate on the solutions of the problem. We come up with two neural network architectural solutions, one being convolutional neural networks and another being LSTM. In the end, we present the comparison of results, suggest future plans and provides Python implementations.

Keywords: time series forecasting, financial analysis, stock price prediction, artificial intelligence, machine learning, neural networks, convolutional neural networks(CNN), long-short-term memory(LSTM)

Chapter 1
Theoretical Background

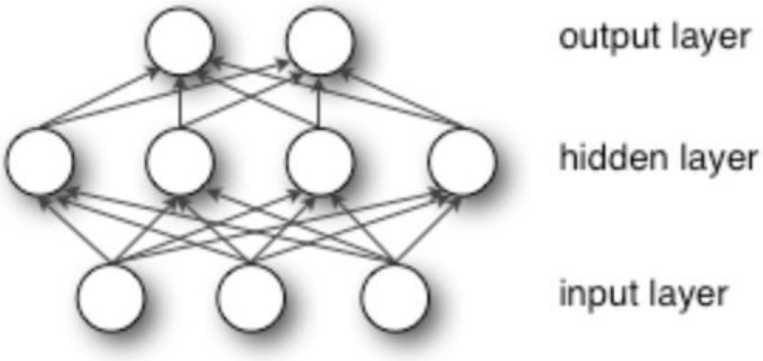
1.1 Neural Networks

There is not any definition fully covering what neural networks are. It is a type of machine learning greatly inspired by biological neural networks. Neural networks are performing deep learning. To make a comparison, while the essential unit of the brain is the neuron, the building block of a neural network is a perceptron**(4)**. They are useful in finding patterns and modeling complex relationships between inputs and outputs in data. In general, it is a network of simple processing elements that show complex behavior based on the connections between the processing elements and element parameters**(9)**.

Now let us introduce some terminologies, so that further discussions will be clear and no questions will arise

- **Feedforward neural network** is one of the first and basic type of neural network invented. Such models are called feedforward as the flow of information goes forward in the neural network, It goes through the input nodes, then follows the hidden layers and finally the output nodes. Here the output of the network is never fed back into itself**(7)**. When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks(6)**. Networks like convolutional neural networks and recurrent neural networks are special cases of feedforward networks.

- **Single-layer perceptron** is a network composed of a single output layer nodes and the inputs are fed directly to the outputs via a series of weights.
- **Multi-layer perceptron(MLP)** is a network consisting of an input layer, an output layer, and in between those two layers, an arbitrary number of hidden layers**(2)**. Hidden layers carry out some computations on the weighted inputs and produce net input.



1.2 Time Series Forecasting

Time series prediction is an important area of machine learning. When tracking business metrics, monitoring industrial processes we are facing with time series data. These data can be described as a sequence of observations stored in time order. So time series analysis is a method to analyze time-series data to get meaningful characteristics of that data and generate other useful insights which can be applied in a business situation**(8)**. Time series forecasting is a technique used across many fields of study not only in business. It is greatly used in geology, economics and many more. The method basically predicts future events by analyzing the trends of the past, having an assumption that future trends will be similar to historical ones. The various

applications of time-series forecasting include weather forecasting, shape detection, earthquake prediction**(14)**.

1.3 Financial Analysis

Financial analysis is the process of assessing a company's performance and making recommendations about how it is going to perform in the future**(5)**. To keep it short, it uses historical data to generate projections about the future. Financial analyses provide insights into the organization's current and future state. Thus it is a critical aspect of the company's success as it highlights both the weaknesses and strengths which directly affect competitiveness**(12)**

The objective of stock price prediction is to find the future value of company stock. Predicting successfully the future price can result in a significant profit. The EMH(efficient-market hypothesis) states that the stock price is unpredictable as it fully reflects all available information. Generally, there are three methods for predicting the stock price. First is fundamental analysis, which is only concerned with the company's fundamentals. The second one is technical analysis, which is the "opposite" of the first one, meaning it is not concerned with the company that underlies the stock. It predicts the future price of a stock based solely on the past trends of the data. And the last one is machine learning, which is using ANNs(common forms: RNNS, TDNN) or GAs**(10)**

.

1.3 Problem Setting

In this project, we implement a regression task, more specifically, stock price prediction using neural networks. We aim to develop convolutional neural networks capable of predicting next close price and compare to the result of LSTM - the main architecture used for solving this problem. For this reason, we implement multi-dimensional convolutional neural networks, using both single step and multi-step time series forecasting technique, give a detailed comparison.

.

Chapter 2
Dataset and features

The dataset we are going to work with is the historical data of Apple Inc, which is an American multinational technology company founded back in 1976. We have taken daily data for 10 years starting from 2009/11/05 to 2019/11/05*. Overall containing 2518 observations and 7 features. The features Open and Close represent the starting and closing price at which the stock is traded on a particular date. High, Low and Adj Close stand for the maximum, minimum, and the last price of the share for the specified date, respectively.

* Data has been taken from https://finance.yahoo.com/quote/AAPL/history?p=AAPL

See the structure of the dataset given below(first and last five observations)

- First five observations:

	Date	Open	High	Low	Close	Adj Close	Volume
Date							
2009-05-11	2009-05-11	18.195715	18.708570	18.160000	18.510000	12.295175	101164700
2009-05-12	2009-05-12	18.508572	18.530001	17.607143	17.774286	11.806481	152370400
2009-05-13	2009-05-13	17.601429	17.717142	17.054285	17.070000	11.338662	148992900
2009-05-14	2009-05-14	17.111429	17.647142	17.100000	17.564285	11.666988	111956600
2009-05-15	2009-05-15	17.474285	17.802856	17.372858	17.488571	11.616698	91891800

- Last five observations:

Date	Date	Open	High	Low	Close	Adj Close	Volume
2019-05-06	2019-05-06	204.289993	208.839996	203.500000	208.479996	207.680222	32443100
2019-05-07	2019-05-07	205.880005	207.419998	200.830002	202.860001	202.081787	38763700
2019-05-08	2019-05-08	201.899994	205.339996	201.750000	202.899994	202.121628	26339500
2019-05-09	2019-05-09	200.399994	201.679993	196.660004	200.720001	199.949997	34908600
2019-05-10	2019-05-10	197.419998	198.850006	192.770004	197.179993	197.179993	41183400

And here is the plot of the Close price feature, that we are interested in.

Chapter 3
Convolutional Neural Networks

3.1 CNN: Explained

Convolutional neural networks are deep artificial neural networks, which are mainly used for classification problems such as image classification(clustering by similarity or performing object recognition). They are many algorithms that are able to identify visual data such as faces, tumors and street signs**(1)**.

The essential part of the convolutional neural network is the convolutional layer, this is where the name of the network comes from. This layer performs convolution, which is a linear operation involving multiplication of a set of weights with the input, similar to what any neural network does. Say, if the convolutional layer was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, sometimes called a filter or kernel. The latter is smaller than the input data and the multiplication type being applied to a filter-sized input and the filter is a dot product(element-wise multiplication resulting in a single value). It is intentionally chosen to have a smaller filter than the input as it allows the same set of weights to be multiplied by the input array many times at different points on the input. The filter is applied systematically to each overlapping part or filter-sized input data, which allows the filter an opportunity to discover that feature without concentrating where it was present, sometimes called translation invariance**(3)**.

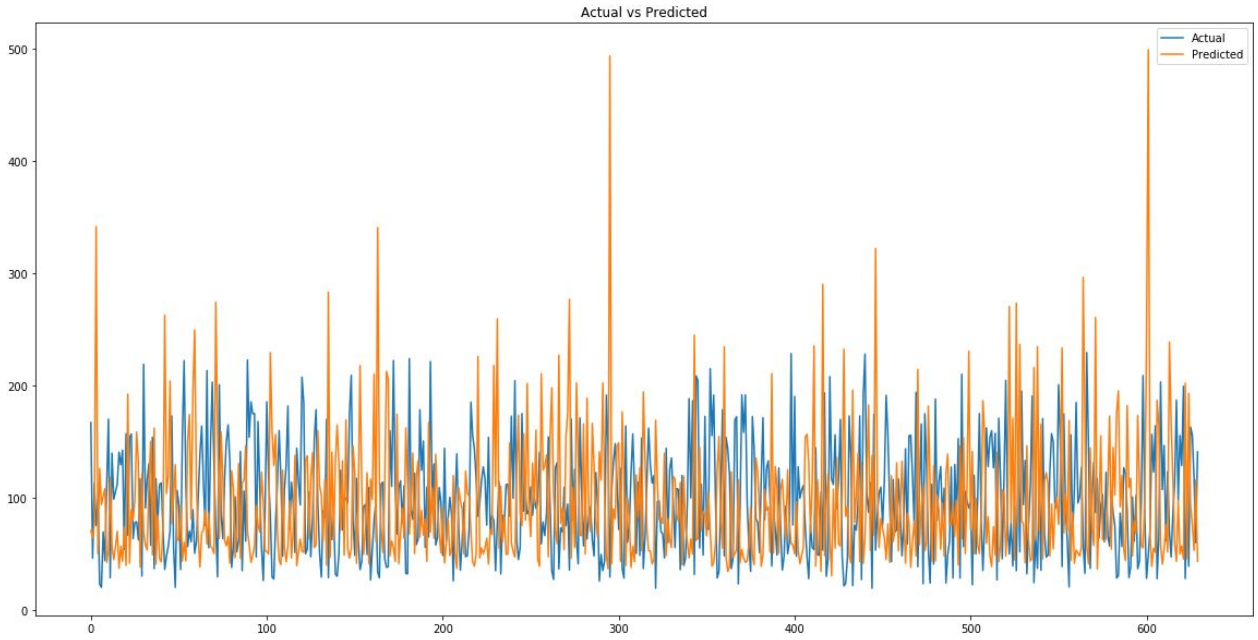
3.2 Why CNN?

Convolutional Neural Networks were initially designed to map image data to an output variable, and they appeared to be so effective that started to be widely used in any prediction problem having image data as an input. Moreover, it works well with data having a spatial relationship. For example, it is used in text analysis as there is an order relationship between words in a document**(15)**. In the same way, there is an ordered relationship in the time steps of a time series. And the main reason for choosing CNN for stock price prediction is for understanding whether it works for financial data or not.

3.3 Multi-dimensional CNN (single-step using one value)

If the aim of time series forecasting is to predict a single observation at the next time step, it is called one-step or single-step forecast. We can do that by providing as an input one value from each feature or by providing multiple values from each feature. Let us first take a look at the multi-dimensional CNN using one value from each metric for predicting the next value of the desired feature.

We want to predict next closing price having the open, high, low, adj close prices and volume. The graph of the actual and predicted value of the close price is given below,



The Root Mean Square Error(RMSE) is

```
print('RMSE = ' + str(rmse))
```

```
RMSE = 97.89326489744205
```

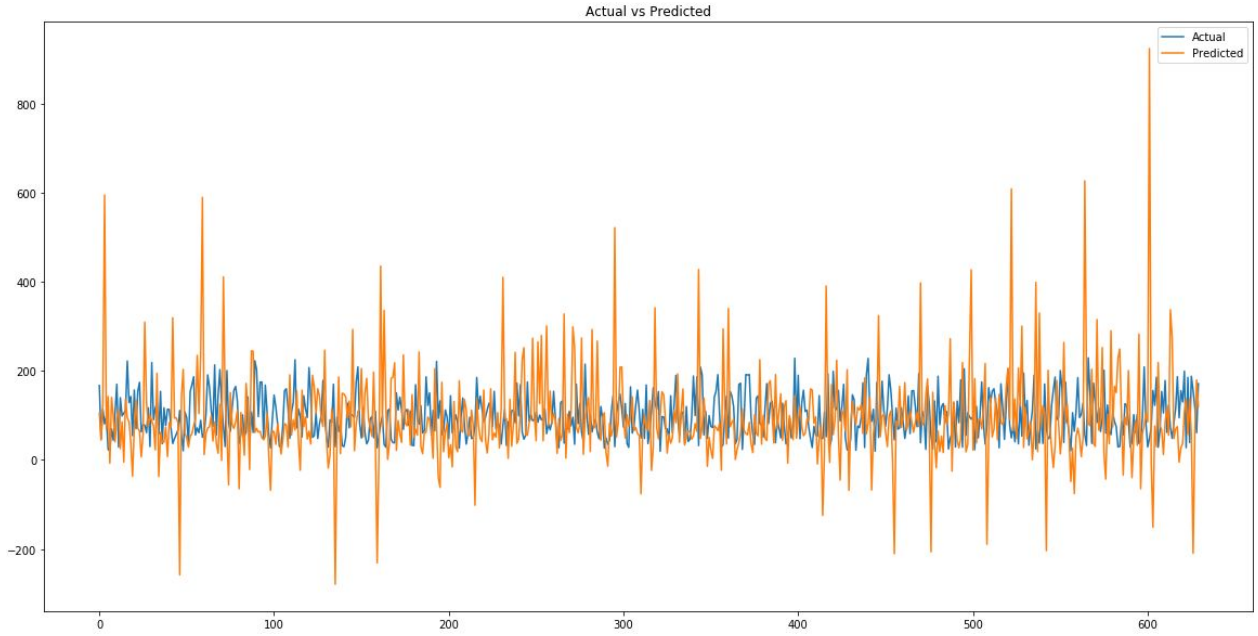
Considering the fact that stock price is unpredictable as EMH is suggesting, our result is not a very bad one, at least it is able to follow the pattern of the actual stock price, and be in the same range with some exceptions.

3.4 Multi-dimensional CNN (single step using multiple values)

Now, as we have already seen that CNNs are more or less working on stock price prediction using single value. Let us also try using multiple values for predicting the close price.

So our problem becomes by having previous two values of five features(high, low, open, adj close prices and volume), to predict next close price.

Below is given the plot of actual and predicted close price values using two value from each metric for predicting the next value of the close price.



The corresponding RMSE is

```
print('RMSE = ' + str(rmse))
```

```
RMSE = 124.96156894422413
```

As we can see from the plot, our model became very sensitive towards an upward or downward movement and is making excessive reaction when observing such behavior.

Chapter 4
Long-Short-Term Memory

4.1 LSTM: Explained

LSTM is a special kind of RNN which is capable of learning long-term dependencies. Practically the model's default behavior is remembering information for long periods**(13)**. For understanding what is LSTM, let us first get the concept of recurrent neural networks(RNNs).

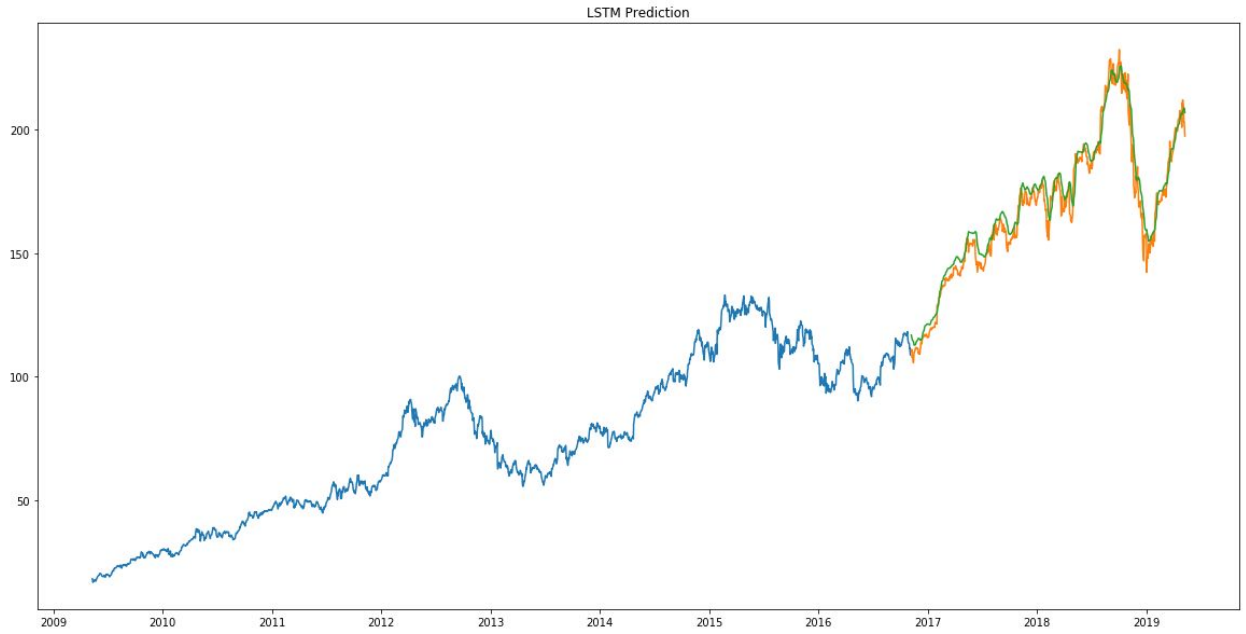
All RNNs have the form of a chain of repeating modules of the neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. Yet, LSTMs has a different structure of the repeating module, instead of having a single neural network layer, there are four interacting ones**(11)**.

3.2 Why LSTM?

LSTM is the most successful recurrent neural network(RNN) as it overcomes the problems of training a recurrent network. It is used on a wide range of applications. It is a great tool for natural language processing, including the work with text or speech data. Even though autoregression methods outperform LSTM on time series forecasting problems, it is a good starting point for time series predictions**(15)**.

3.2 LSTM

The plot below shows that the predictions and actual results are in the same range and have the same shape. Even though the prediction is not a perfect one, the model has reached the state of understanding the movement of the future price fluctuation. The is a work for improving the prediction of the fluctuation rate. But overall the model has RMSE of around 5.



The RMSE of the LSTM model is

```
Loaded model from disk
RMSE: 5.204993154659221
```

which is a very good result compared to our CNN models.

Chapter 5
The Results and Conclusion

Now let us compare the results of three models. The following table shows the models with corresponding RMSEs.

Models	CNN(1-1)	CNN(2-1)	LSTM
RMSE	98	125	5

The results show that LSTM works best. CNN models are not perfect but show interesting result. I was expecting that as the number of values in a time step is increasing, the accuracy of the prediction increases. However, as we can see that the model performs better when it gets one value from each feature and predicts closing price, rather than getting two values from each feature. I have tried increasing the number of time steps and as it increases the RMSE increases as well.

So we can conclude that CNN shows better results if it is based on less number of previous values. That's because it is not concentrated on the next value, instead, it is build for understanding future behavior, not the exact value. And as the number of previous values increases, it starts behaving excessively to the increase or decrease of the next value.

Chapter 6
Future plans

The goal of this paper was understanding whether convolutional neural networks can work for time series prediction problem, specifically for stock price prediction one, and compare the results with LSTM, which has already been proved to work on financial data. It was not concentrated on building a perfect model.

For later analysis, it would be great to improve the CNN model to perform better, maybe use a mixed model of LSTM and CNN and see if it outperforms the LSTM one.

Chapter 7
Appendices

Algorithm Implementations in Python(Keras)

7.0 GitHub URL

https://github.com/arusyak97/capstone

7.1 Necessary Packages

```
from keras.models import Model, Sequential, model_from_json
from keras.layers import Input, Dense, Flatten, Dropout, LSTM
from keras.layers.convolutional import Conv1D, MaxPooling1D
from sklearn.preprocessing import MinMaxScaler
from keras.layers.merge import concatenate
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, mean_squared_error
import matplotlib.pylab as plt
from random import seed
from math import sqrt
import pandas as pd
import numpy as np
from numpy import hstack, array
```

7.2 Defined Functions

```
def setInitalParameters(number_of_steps, number_of_features):
  # choose a number of time steps
  n_steps = number_of_steps

  # choose a number of features
  n_features = number_of_features

  return n_steps, n_features
```

```
def plotClosePriceByDate(dataset):
  # setting date as our data index
  dataset['Date'] = pd.to_datetime(dataset.Date, format = '%Y-%m-%d')
  dataset.index = dataset['Date']

  # plot close price history
  plt.figure(figsize = (16, 8))
  plt.plot(dataset['Close'])
  plt.xlabel("Date")
  plt.ylabel("Close Price")
  plt.title("Close Price history")
  plt.legend()
  plt.show()
```

```
def getDatasetModified(df):
  open = df['Open'].as_matrix()
  high = df['High'].as_matrix()
  low = df['Low'].as_matrix()
  close = df['Close'].as_matrix()
  adj_close = df['Adj Close'].as_matrix()
  volume = df['Volume'].as_matrix()


  open_h = open.reshape((len(open), 1))
  high_h = high.reshape((len(high), 1))
  low_h = low.reshape((len(low), 1))
  close_h = close.reshape((len(close), 1))
  adj_close_h = adj_close.reshape((len(adj_close), 1))
  volume_h = volume.reshape((len(volume), 1))

  dataset = hstack((open_h, high_h, low_h, adj_close_h, volume_h, close_h))
  return dataset
```

```
# split a multivariate sequence into samples
def split_sequences_multi(sequences, n_steps):
  X, y = list(), list()
  for i in range(len(sequences)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the dataset
    if end_ix > len(sequences):
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)
```

```
def getTrainTest(data, number_of_steps, size_of_test):
  # split into samples
  X, Y = split_sequences_multi(data, number_of_steps)

  # let's split X and Y data into test and train datasets, test data is size_of_test * 100% of overall data
  x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = size_of_test, random_state = 42)

  return x_train, x_test, y_train, y_test
```

```
def plotGraph(actual, predicted):
  plt.figure(figsize=(20, 10))
  plt.plot(actual)
  plt.plot(predicted)
  plt.title('Actual vs Predicted')
  plt.legend(['Actual', 'Predicted'], loc='best')
  plt.show()
```

```
def getLoadedModel(modelName):
  # load json and create model
  json_file = open(str(modelName) + '.json', 'r')
  loaded_model_json = json_file.read()
  json_file.close()
  loaded_model = model_from_json(loaded_model_json)
  # load weights into new model
  loaded_model.load_weights(str(modelName) + ".h5")
  print("Loaded model from disk")
  return loaded_model
```

```
def saveModel(model, modelName):
  # serialize model to JSON
  model_json = model.to_json()
  with open(modelName + ".json", "w") as json_file:
    json_file.write(model_json)
  # serialize weights to HDF5
  model.save_weights(modelName + ".h5")
  print("Saved model to disk")
```

```
def printRMSE(y_test, y_pred):
  rmse = sqrt(mean_squared_error(y_test, y_pred))
  print('RMSE = ' + str(rmse))
```

```
def defineFitModel(number_of_steps, number_of_features, size, x_train_reshaped, y_train):
  # define model
  model = Sequential()
  model.add(Conv1D(filters=64, kernel_size=size, activation='relu', input_shape=(number_of_steps, number_of_features)))
  model.add(MaxPooling1D(pool_size=size))
  model.add(Flatten())
  model.add(Dense(50, activation='relu'))
  model.add(Dense(1))
  model.compile(optimizer='adam', loss='mse')
  # fit model
  model.fit(x_train_reshaped, y_train, epochs=1000, verbose=0)
  return model
```

```
def lstmModel(x_train, y_train):
  # create and fit the LSTM
  model = Sequential()
  model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1], 1)))
  model.add(LSTM(units = 50))
  model.add(Dense(1))
  model.compile(loss = 'mean_squared_error', optimizer = 'adam')
  model.fit(x_train, y_train, epochs = 1, batch_size = 1, verbose = 2)

  return model
```

7.3 Dataset and features

```
# import all needed packages
importPackages()

# read data
df = pd.read_csv('AAPL.csv')
```

```
# print data head
df.head()
```

```
# print data tail
df.tail()
```

```
# plot close price by date
plotClosePriceByDate(df)
```

7.4 Multi-dimensional CNN (single-step using one value)

```
# get the dataset modified
dataset = getDatasetModified(df)

# some initial preparation
n_steps, n_features = setInitalParameters(1, 5)
x_train, x_test, y_train, y_test = getTrainTest(dataset, n_steps, 0.25)
```

```
######## I am going to use already saved model
######## if you want to ceate and fit the model open the commented part
# define model
# model1 = defineFitModel(n_steps, n_features, 1, x_train, y_train)
# saveModel(model1, 'model1')

# get loaded model
loaded_model1 = getLoadedModel('model1')
```

```
# reshaping the data
pred_1_1 = loaded_model1.predict(x_test)

# flatten the result
pred_flattened_1_1 = pred_1_1.flatten()
```

```
# plot the graph of predicted and actual values
plotGraph(y_test, pred_flattened_1_1)
```

```
printRMSE(y_test, pred_flattened_1_1)
```

7.5 Multi-dimensional CNN (single step using multiple values)

```
# some initial preparation
n_steps, n_features = setInitalParameters(2, 5)
x_train, x_test, y_train, y_test = getTrainTest(dataset, n_steps, 0.25)
```

```
######## I am going to use already saved model
######## if you want to ceate and fit the model open the commented part
# define model
# model2 = defineFitModel(n_steps, n_features, 1, x_train, y_train)
# saveModel(model2, 'model2')

# get loaded model
loaded_model2 = getLoadedModel('model2')
```

```
# reshaping the data
pred_2_1 = loaded_model2.predict(x_test)

# flatten the result
pred_flattened_2_1 = pred_2_1.flatten()
```

```
# plot the graph of presicted and actual values
plotGraph(y_test, pred_flattened_2_1)
```

```
printRMSE(y_test, pred_flattened_2_1)
```

7.6 LSTM

```
data_length = len(df)
new_data = pd.DataFrame(index = range(0, data_length), columns = ['Date', 'Close'])
for i in range(0, data_length):
    new_data['Date'][i] = df['Date'][i]
    new_data['Close'][i] = df['Close'][i]

# setting index
new_data.index = new_data.Date
new_data.drop('Date', axis = 1, inplace = True)

# creating train and test sets
data = new_data.values

# dividing dataset into train and test ones
index = data_length * 3//4
train = data[0:index,:]
test = data[index:,:]
```

```
######## I am going to use already saved model
######## if you want to ceate and fit the model open the commented part
# #converting dataset into x_train and y_train
# scaler = MinMaxScaler(feature_range = (0, 1))
# scaled_data = scaler.fit_transform(data)
# x_train, y_train = [], []
# for i in range(60, len(train)):
#     x_train.append(scaled_data[i-60:i,0])
#     y_train.append(scaled_data[i,0])
# x_train, y_train = array(x_train), array(y_train)
# x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
# model0 = lstmModel(x_train, y_train)
# saveModel(model0, 'model0')

model0 = getLoadedModel('model0')

# predicting 1/4 of dataset values, using past 60 from the train data
inputs = new_data[data_length - len(test) - 60:].values
inputs = inputs.reshape(-1, 1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i, 0])
X_test = array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
closing_price = model0.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)


# rmse
rmse = np.sqrt(np.mean(np.power((test - closing_price), 2)))
print("RMSE:", rmse)
```

```
# plot the actual vs predicted
train = new_data[:index]
test = new_data[index:]
test['Predictions'] = closing_price
plt.figure(figsize=(20, 10))
plt.plot(train['Close'])
plt.plot(test[['Close','Predictions']])
plt.title("LSTM Prediction")
plt.show()
```

References

1. A Beginner's Guide to Convolutional Neural Networks (CNNs). (n.d.). Retrieved from https://skymind.ai/wiki/convolutional-network
2. A Beginner's Guide to Multilayer Perceptrons (MLP). (n.d.). Retrieved from https://skymind.ai/wiki/multilayer-perceptron
3. A Gentle Introduction to Convolutional Layers for Deep Learning Neural Networks. (2019, April 02). Retrieved from https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
4. DeMuro, J., & DeMuro, J. (2018, August 11). What is a neural network? Retrieved from https://www.techradar.com/news/what-is-a-neural-network
5. Financial Analysis - Overview, Guide, Types of Financial Analysis. (n.d.). Retrieved from https://corporatefinanceinstitute.com/resources/knowledge/finance/types-of-financial-analysis/
6. Gupta, T., & Gupta, T. (2017, January 05). Deep Learning: Feedforward Neural Network. Retrieved from https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7
7. Kumar, N., & Kumar, N. (2019, April 01). Deep Learning: Feedforward Neural Networks Explained. Retrieved from

https://hackernoon.com/deep-learning-feedforward-neural-networks-explained-c34ae3f084f1

8. Lyla, Y., & Lyla, Y. (2019, January 02). A Quick Start of Time Series Forecasting with a Practical Example using FB Prophet. Retrieved from https://towardsdatascience.com/a-quick-start-of-time-series-forecasting-with-a-practical-example-using-fb-prophet-31c4447a2274
9. Sandeep. (2019, April 06). Neural Networks and Data Mining. Retrieved from https://www.vskills.in/certification/tutorial/data-mining-and-warehousing/neural-networks-and-data-mining/
10. Stock market prediction. (2019, May 10). Retrieved from https://en.wikipedia.org/wiki/Stock_market_prediction
11. Understanding LSTM Networks. (n.d.). Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/
12. What is Financial Analysis? (n.d.). Retrieved from https://www.financialplannerworld.com/what-is-financial-analysis/
13. What is LSTM? (2018, April 19). Retrieved from https://hub.packtpub.com/what-is-lstm/
14. What is time series forecasting? - Definition from WhatIs.com. (n.d.). Retrieved from https://whatis.techtarget.com/definition/time-series-forecasting
15. When to Use MLP, CNN, and RNN Neural Networks. (2018, April 25). Retrieved from https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/