ANALYSES OF STOCK PRICE PREDICTIONS USING DEEP LEARNING TECHNIQUES

by

Abrar Ul Alam

A THESIS

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE – ELECTRICAL ENGINEERING

_____
Abrar Ul Alam, Candidate

ADVISORY COMMITTEE

_____
Dr. Kevin Lu, Co-Advisor Date

_____
Dr. Min Song, Co-Advisor Date

_____
Dr. Serban Sabau, Reader Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2020

ANALYSIS OF STOCK PRICE PREDICTION USING DEEP LEARNING TECHNIQUES

ABSTRACT

Stock market prices are difficult to forecast due to their highly volatile, complex, and nonlinear nature, which accounts for numerous inconsistencies in their data. These inconsistencies are usually influenced by factors such as political and economic alterations, leadership changes, investment sentiments, financial policies, public psychology, and news articles pertinent to the situation of companies. In the past, artificial neural networks (ANNs) were popularly employed to predict and analyze stock market movements, financial incomes, and exchange rates. ANNs have the capacity to learn from historical stock data, however, one major limitation of ANNs is that they tend to suffer from overfitting issues due to the presence of large numbers of parameters to fix and little to no historical information about the importance of the inputs in the analyzed problems. On the contrary, recurrent neural networks (RNNs) are special types of ANNs that were developed to learn sequential or time varying patterns. Long Short-Term Memory (LSTM) neural network is such an example of a strong RNN architecture that can learn from historical data and predict models applied in the field of computational intelligence. LSTM specializes at remembering values for either long or short durations of time. A detailed explanation of how LSTM works, and its overall architecture has been provided in the following section below.

The primary purpose of this thesis is to analyze stock price predictions using deep learning techniques, such as Keras and TensorFlow. Keras is an open-source neural network library that is written in Python and can run on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theono, or PlaidML. Keras was developed to allow fast analyses with neural networks; it particularly focuses on its ability to be user-friendly, modular, and extensible. TensorFlow is also an open-source Python-friendly library that was designed to perform numerical computations, thereby making

machine learning faster and easier. The datasets used for this thesis consisted of Apple, Goldman Sachs and Credit Suisse stock prices extracted from Yahoo to facilitate real-time predictions. The extracted data was preprocessed, normalized to have values between 0 and 1, and then split into training and testing datasets. Using the Keras and TensorFlow libraries in Python, the respective datasets were fed into the LSTM neural networks to train the models. The hyperparameters of the LSTM neural networks were tuned while fitting the test datasets with the trained models. Finally, the root mean square error (RMSE) values of each model were evaluated and their corresponding graphical models were plotted to demonstrate the performance of the LSTM neural networks using Keras and TensorFlow.

Author: Abrar Ul Alam

Co-Advisor: Dr. Kevin Lu

Co-Advisor: Dr. Min Song

Date: May 5, 2020

Department: Electrical and Computer Engineering

Degree: Master of Science

Acknowledgement

I would like to take this opportunity to thank both Dr. Kevin Lu, Dr. Ming Song, and Dr. Serban Sabau for their immense support and encouragement in pursuing this thesis topic and proceeding further with the development. Without their guidance and valuable input, the completion of this thesis would not have been possible. Their words of wisdom and knowledge that they have shared will always resonate within me and help me to go a long way in my future endeavors.

This thesis proved to be an amazing learning experience for me as I was able to fully comprehend and simultaneously observe the practical implications of applying deep learning techniques for predicting stock market movements. Over the course of my thesis work, I was able to gain relevant research experience and explore various ways of approaching a research project, for which I will always be grateful to Dr. Kevin Lu. I intend to use everything that I have learned and apply them in real-world applications either as a Data Scientist or a Data Engineer.

Table of Contents

List of Tables

List of Figures

I. Introduction

In today's era of technological advancements and modern innovations, stock price prediction has become an area of immense importance in the financial industry because a feasibly accurate prediction acquires the capacity to generate high financial advantages and hedge against market risks. The swift expansion of the internet and complex computing technologies demand for more frequent operations on the stock market; it is believed that the said frequency has recently increased to approximately fractions of seconds. According to [1], the Brazilian Stock Exchange (BM&FBovespa) has operated in high-frequency and its number of high-frequency actions increased from 2.5% in 2009 to 36.5% in 2013. In 2016, it was estimated that high-frequency trading on average consisted of 10% to 40% of trading volume in equities and 10% to 55% of volume in foreign exchanges and commodities. Additionally, JPMorgan approximated that in 2017 just 10% of the United States' trading volume contained regular stock pricing. These numbers gravely emphasize that the high-frequency stock market has become a global trend.

A. Literature Review

Several approaches can be implemented in the prediction of stock market movements with varying sets of pros and cons. Previously, Hegazy et al. [2] suggested a particle swarm optimization (PSO) algorithm in conjunction with least square support vector machine (LS-SVM), where the PSO algorithm was applied to optimize the LS-SVM technique to forecast daily stock prices. LS-SVMs can be defined as the least square versions of classical support vector machines that primarily consist of sets of related supervised learning techniques employed for investing data and detecting patterns in classification, as well as regression analyses. The solutions in LS-SVM are generally determined by solving a set of linear equations instead of convex quadratic programming (QP) problems for standard SVM. The PSO algorithm operates by selecting a combination of the best free features for LS-SVM so that overfitting and other issues pertinent to local minima can be

avoided to improve the accuracy of predictions. The proposed algorithm was tested for several companies in the Standard and Poor's (S&P) 500 stock market index and its performance was compared with that of the artificial neural network (ANN) approach integrated with the Levenberg-Marquardt (LM) algorithm.

Another group of researchers in [3] described an approach that could enhance the precision of stock price predictions by gathering a large amount of time series data and evaluating them with respect to news articles, with the aid of deep learning techniques. This approach was suggested on the claim that there is a strong correlation between stock price movements and news article publications. Financial news focused on companies tend to have more substantial effects on their stock prices and hence, the authors suggested an automated system that can accumulate financial news related to target companies in real-time and apply machine learning models on those data, as well as on historical stock price information to predict future prices. Stock prices of S&P 500 companies over a 4-period were used for this experiment along with 265463 related news articles gathered over the same period of time. The dataset, split into 90% training, 5% validation, and 5% testing datasets, was employed to train three machine learning models, which included Auto Regressive Integrated Moving Average (ARIMA), Facebook Prophet, and recursive neural network (RNN) Long Short-Term Memory (LSTM). Besides news articles, public's sentiments towards rumors are also believed to have a significant correlation with public psychology, which has inherently made it tricky to acquire a centralized insight on the market. Therefore, another group of researchers had developed a model that would consider public preferences/sentiments through traded volume, number of transactions along with price fluctuation analyses, and later feed the obtained results into neural networks to forecast the percentage change in stock prices. In order to conduct this experiment, the authors utilized the theory of supply and demand, which formed the basis of their claim that performing manual evaluations of supply and demand can help to acquire

valuable insights on future stock prices. The parameters considered for the designing the predictive were collected in 30-minute intervals, redundant data were filtered and removed accordingly to maintain data integrity, and an optimized feed-forward (3x10) neural network model was employed by adjusting the number of neurons in the hidden layers. Considering the training time and complexity of the model in conjunction with the wrapper technique, three relevant features, i.e., the number of transactions, traded volume, and last traded price, were chosen.

In another research paper, an LSTM-based neural network model was proposed to predict short-term alterations of stock transactions. According to [4], in a conventional recurrent neural network algorithm during the gradient back-propagation phase, the gradient signal could be multiplied a huge number of times by the weight matrix related to the connections between the neurons of the hidden layer. This phenomenon implied that the magnitude of the weights in the transition matrix might potentially have had a strong impact on the learning process. If the leading eigenvalue of the weight matrix was smaller than 0.1, then vanishing gradients occurred, where the gradient signals diminished to the extent at which learning either became very slow or stopped operating entirely. However, if the leading eigenvalue of the weight matrix was larger than 0.1, then the gradient signals became large enough to cause the learning rate to diverge, an occurrence that is frequently termed as exploding gradients. The LSTM technique presented a new and enhanced structure that will be discussed in the following sections below. LSTM with real-time wavelet-denoising functions [5] was a hybrid technique that was used to forecast stock price fluctuations. In this novel algorithm, the wavelet embraced a sliding window mechanism to exclude future data while flexibly optimizing its system configuration based on a few predefined specifications. Stocks are generally noisy and denoising has been deemed as one of practical ways of improving prediction performances. Wavelet transform can be defined as a time-frequency decomposition that decays time series data in both time and frequency domains, despite the data

being nonstationary. Moving averages (MAs) and Fourier transforms can both be used to denoise data, but wavelet denoising inhibits the capacity to overcome their disadvantages; MAs usually lag behind actual data trends and Fourier Transforms cannot properly handle time information and thus, becomes unsuitable for nonstationary signals such as stock prices. A hybrid approach of Hodrick-Prescott (HP) filter and support vector regression (SVR) was also suggested to optimize stock price prediction models. In [5], the HP filter was denoted as a tool in macroeconomics that is typically used to separate the cyclical components of time series models from raw data. HP filters can be applied to achieve smooth-curved representations of time series models that are more sensitive to long-term than short-term inconsistencies. HP filters can be used because it can be applied on nonstationary time series models, which has become a topic of great concern for numerous macroeconomic and time series data.

B. Problem Formulation

The prediction of stock prices is a popularly researched topic in several fields of study including trading, finance, statistics, computer science and nowadays, in engineering as well. The motivation for forecasting the direction of stock price movements is the ability to buy and sell stocks at profitable positions [6]. Stock prices are typically considered to be very dynamic and vulnerable to rapid fluctuations due to the inherent nature of the financial domain and partly due to the amalgamation of known and unknown features such as election results, rumors, industry related news articles, financial regulatory changes, etc. Stock price prediction can aid traders in taking the right decisions by allowing them to purchase stocks before their prices increase and sell them before their values decrease. Not only that, forecasting stock prices can assist companies to use information from historical data, combine them with existing data, plan ahead and make the necessary changes to their business models in order to adapt to the altering circumstances in the future. Accurate prediction algorithms have the capacity to generate high profits for investment

firms, implying that a direct relationship exists between the accuracy of prediction algorithms and profits made from these algorithms [6]. However, it is not actually as easy as it might sound to accurately predict time series movements like stock prices.

C. Problem Solution

For more than a decade, various techniques have been used to predict stock prices, which can be segmented into two groups. The first category of algorithms attempts to augment the performance of predictions by enhancing the prediction models while the second category of algorithms focus on boosting the features based on which the predictions are made. This thesis focuses on the first category. The primary purpose of this thesis is to analyze the effects of deep learning tools, such as Keras and TensorFlow, in collaboration with the LSTM-RNN architecture. The prediction accuracy, using root mean square error (RMSE) as the performance measures, were computed for both of these methodologies and then compared to the one obtained from running the SVR algorithm as it operates as a neural network with the radial basis (kernel) function as a parameter.

D. System Setup

In order to perform the experiments for this thesis, a Google Cloud Platform (GCP) service called Google Colab was used. Since TensorFlow is a Google product, it was very easy and efficient to run the LSTM-RNN architecture on Google Colab. Each model was split into the following sections while being in accordance with the extract-transform-load (ETL) process:

- ➢ Data was extracted from Yahoo to facilitate real-time analysis of stock price movements
- ➢ Extracted data was preprocessed to find and eliminate any null values
- ➢ Preprocessed data was further transformed and split into training, testing, and validation datasets; in this case, transformation implies normalizing the preprocessed data to have

values between 0 and 1, and reshaping them accordingly to be fed into the appropriate neural network

- ➢ Transformed data was then loaded into the LSTM-RNN architecture for training the neural network models
- ➢ The trained models were used to fit the test dates to acquire the predicted stock prices and compare them to the test stock prices
- ➢ Finally, the accuracy of the models was calculated using RMSE and illustrated on multiple plots

1) Getting Familiar with the LSTM Architecture

Invented by Hochreiter and Schmidhuber in 1997 [7], LSTM is believed to be an excellent alternative of RNN that assumes the behaviour of most RNN algorithms and solves the issues of gradient disappearance caused by gradient backpropagation [8]. Figure 1 below demonstrates the schematic of a typical LSTM architecture.

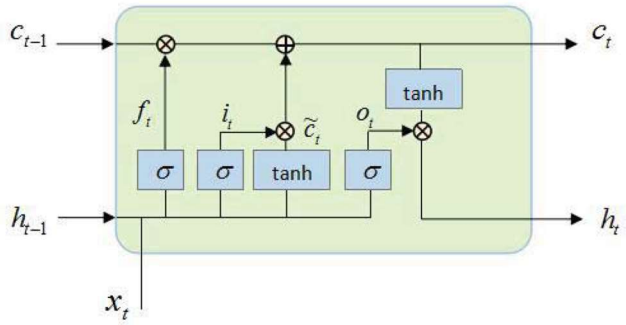

Figure 1. Schematic of a typical LSTM architecture [9].

Like human beings, RNNs provide short-term memory capabilities that enable them to produce outputs that are generally subject to past and prevent events. However, RNNs find it difficult to apprehend long-term dependencies when data is transmitted for a long time. LSTMs can overcome the limitations of RNNs and that is why they are popularly applied for modelling

neural networks. Moreover, the internal design of an LSTM contains more components than an RNN. The central component of an LSTM structure is its memory block that consists of one or more memory cells and three multiplicative gating cells apportioned between all the cells in the block. As stated in [8], the core idea of the LSTM structure is to present a cell state connection that stores the necessary information while updating its internal entry control structure and generating the control data as outputs. LSTM incorporates linear units known as Constant Error Carousels (CECs) that prevent it from experiencing gradient vanishing or explosion problems, which have hindered the performances of previous RNNs. As stated in [9], each CEC has a fixed self-connection and is circumscribed by three gating units that regulate the flow of information in and out of the CEC. LSTM implements a forward propagation process based on its gating units and can be described as [9]:

- Step 1: The input gate learns about the data that needs to be stored in the memory:

$$input_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (1)$$

  Where x_t can be denoted as the input value at time step *t,* h_{t-1} as the output value of the previous time step, and the subscript *i* as the value related to the input gate

- Step 2: The forget gate learns how long the memory needs to be stored:

$$input_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (2)$$

  Where the subscript *f* indicates that the value is pertinent to the forget gate.

- Step 3: The memory cell gets updated at current time step:

$$c_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

$$cell_t = forget_t \circ cell_{t-1} + input_t \circ c_t \quad (4)$$

- Step 4: The output gate learns when memory out needs to be read and then calculates the output value $h_t$at time *t*:

$$output_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (5)$$

$$h_t = output_t \circ tan(cell_t) \quad (6)$$

2) Defining TensorFlow and Keras:

TensorFlow can be defined as an open-source deep learning framework developed by Google [10]. Its front-end aids several development languages, including but not limited to, Python, C++, and Java. The framework's backend was designed using C++, Compute Unified Device Architecture (CUDA), and a combination of other object-oriented languages. TensorFlow facilitates the usage of algorithms that can be easily transported onto numerous heterogeneous systems as it consists of a detailed and flexible ecosystem of tools, libraries, and other resources that offer workflows with high-level application programming interfaces (APIs). It is favored by many developers [10]. The algorithms used in this framework possess the capacity to apply queue and thread operations from the bottom layer, rapidly call hardware resources, provide input data, graph node structure, and object functions, as well as allocate nodes to many devices for parallel operation. Other features of TensorFlow include:

- ➢ Easy model building that issues multiple levels of abstraction to develop and train algorithms
- ➢ Robust machine learning production using any programming language and/or platform
- ➢ Powerful experimentation for research work through flexible and controlled aspects such as the Keras Functional API and Model Sub-classing API for the formation of complex topologies

The TensorFlow framework is mainly made up of two core building blocks; a library required for describing computational graphs and a runtime for implementing such graphs on various kinds of hardware, for example, a central processing unit (CPU) and graphics processing unit (GPU). A computational graph can be expressed as an abstract way of explaining computations as a directed graph, whereas a directed graph can be denoted as a data structure made of nodes (vertices) and

edges [11]. In other words, a directed graph is a group of vertices connected pairwise by directed edges. TensorFlow operates by employing directed graphs internally to represent computations, which are referred to as data flow graphs or computational graphs. Nodes in a directed graph can represent any entity, but the nodes in a computational graph mostly demonstrate operations, variables, or placeholders. Operations are responsible for producing or manipulating data in accordance with specified rules. In TensorFlow these operations are called Ops. Diversely, variables illustrate shared, persevering states that can be influenced by running Ops on those variables. The edges constitute data or multidimensional arrays, commonly referred to as Tensors, that circulate through the different operations. Basically, edges carry information from one node to another. The output of one operation, which is one node, becomes the input to another operation and thus, the edge connecting the two nodes carry the values. A computational graph in TensorFlow is composed of numerous components:

- Variables that are just like regular variables used in programming languages. Variables in TensorFlow can be altered at any point in time, but they need to be initialized prior to running the graph in a session. These variables denote modifiable parameters within the graph. Weights or biases are excellent examples of variables in a neural network.
- Placeholders are also variables which enable users to input data into the graph from the outside, but they are not required to be initialized. Rather they simply serve as assigned methods to define the shape and the data type. Placeholders can be considered as empty nodes in the graph that are typically used for feeding in inputs and labels.
- Constants are parameters that cannot be modified
- Operations are used to express nodes in the graph that perform calculations on Tensors
- Graphs represent a central hub that attaches all the variables, placeholders and constants to operations

- ➢ Sessions create runtimes in which operations are implemented and Tensors are evaluated. Sessions are also responsible for assigning memories and storing values of intermediate results and variables

On the other hand, Keras is a modular neural network library that can be used in Python language [12]. For running Keras, either TensorFlow or Theano can be chosen as the primary backend source. Moreover, it consists of a diverse selection of modules, including activation function modules, preprocessing modules, objective function modules, layer modules, optimization technique selection modules and so on. Amidst the modules, the activation function modules and optimization method selection modules incorporate numerous latest and best optimization approaches along with activation functions that have been widely used in recent years. With the assistance of these modules, neural network algorithms can be designed easily while maintaining a certain level of flexibility to enhance the vital parameters of the said neural networks for swift prototyping and seamless running on CPU and GPU. The crucial advantages of Keras are as follows:

- ➢ User-friendly, which makes Keras simple, consistent and interface efficient for general use cases
- ➢ Modular and composable, which allows Keras models to connect adjustable building blocks together with limited regulations
- ➢ Ease to extend, which enables researchers to easily compose customizable building blocks for novel ideas and innovations
- ➢ Easy to use as Keras offers stable and simple APIs for diminishing the number of user actions necessary for general use cases

The easiest way to create a model is to employ the sequential API that enables developers to stack multiple neural network layers together. Nevertheless, the major issue with the Sequential API is that it does not allow models to have multiple inputs or outputs, which are required for solving

some deep learning problems. That is why the functional API exists so that models can be designed with more flexibility by sacrificing simplicity and readability. Functional APIs such as *Conv2D*, *MaxPool2D*, etc. allow models to integrate multiple inputs with shared layers to create multiple outputs and hence facilitates the development of complex network structures. The use of functional API requires the previous layer to be passed onto the current layer and the use of an input layer.

Although Keras runs on top of TensorFlow, they both have their differences. TensorFlow facilitates both high and low-level APIs while Keras only supports high-level APIs. Due to its flexibility, TensorFlow can support immediate iterations along with intuitive debugging, but Keras provides simple and steady high-level APIs with the ability to follow best practices that can significantly minimize the cognitive load for users. However, since Keras is built in Python, it makes it way more user-friendly than TensorFlow.

E. Contributions

The algorithms used for this thesis were chosen from existing frameworks to evaluate the prediction accuracy of the LSTM models. Besides analyzing stock price predictions using Keras and TensorFlow, another major goal was to enhance the prediction accuracy of the existing frameworks. These goals can be achieved by tuning the hyperparameters of the Keras and TensorFlow frameworks that are likely to have lasting effects on the prediction performance. This thesis discusses in details the parameters that were tuned, how they were tuned, and why they were identified as tunable parameters for the planned experiments, thereby providing an in-depth insight of each experiment's influence on the acquired RMSE and mean square error (MSE) values. Furthermore, this thesis demonstrates the importance of conducting experiments to determine the best combination of parameters that are likely to vary with different sets of stock price data. For example, within the same framework, one combination of parameters may produce the best predicted results of a certain stock dataset, but the same combination may not work the same way

for another stock dataset, possibly leading to unwanted outputs. Another contribution is to show how simple it is to use Keras while it is extremely challenging to apply TensorFlow. Additionally, this thesis provides a detailed insight of the conditions under which Keras and TensorFlow work well and the different types of stock price predictions that can be performed using these two tools.

II. Methodology

Each analyses in this thesis was performed in four main parts; extracting a large dataset from a reliable source, preprocessing and transforming the extracted dataset based on the deep learning technique applied, loading the transformed datasets into the appropriately designed neural networks for training the models, and forecasting the target stock prices followed by computing the prediction accuracy, as well as plotting the results against test datasets.

A. Data Extraction

The dataset used for this thesis consisted of Apple, Goldman Sachs, and Credit Suisse stock prices extracted from Yahoo using the *pandas_datareader* module in Python. Pandas can be defined as an open-source library that offers high-performance, easy-to-use data structures, and data analysis tools for Python. It also provides a diverse range of utilities for data collection, manipulation, and analysis, which makes it one of the most popular resources for developing trading strategies. Consequently, the *pandas_datareader* module facilitates a simple, consistent API for users to gather data from platforms such as IEX and Quandl. Hence, the module was used to extract the said stock prices from Yahoo by specifying the start and end date parameters and thereby, furnishing a more real-time analysis of the stock prices. Figures 2, 3, and 4 demonstrate the open, close, high, and low stock prices of Apple, Goldman Sachs, and Credit Suisse, respectively, plotted against time.

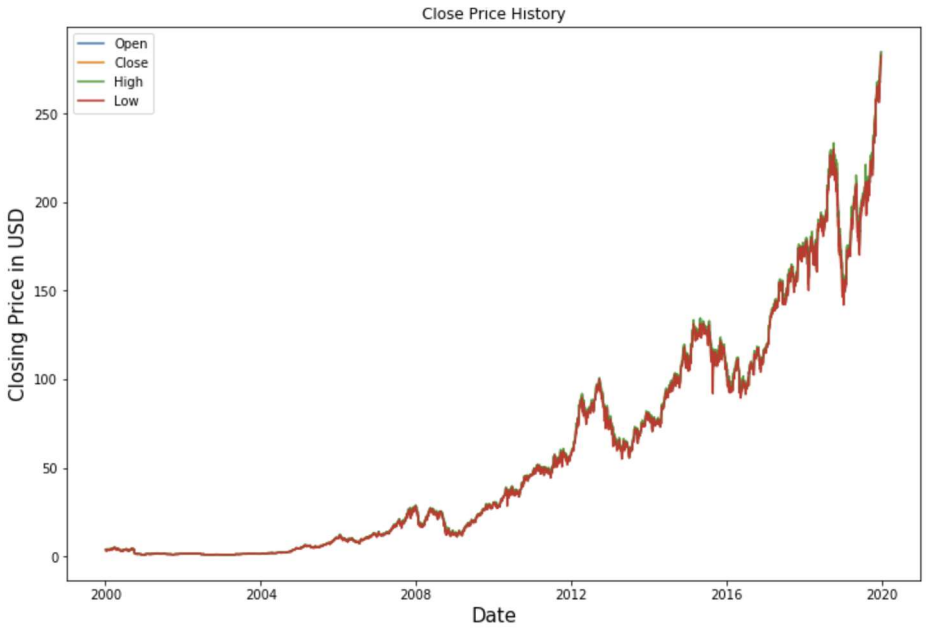

Figure 2. Open, close, high, and low stock prices of Apple over time.

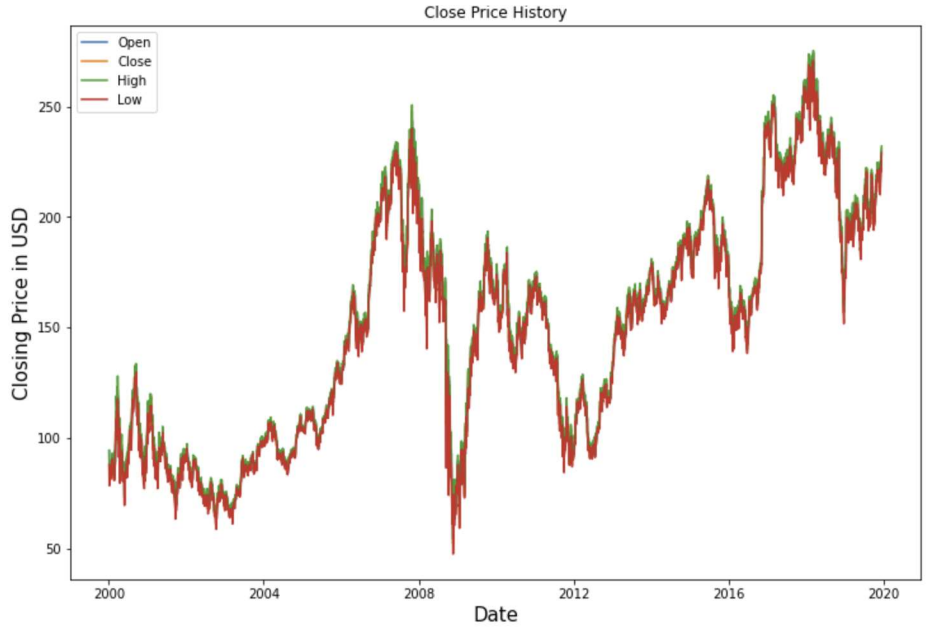

Figure 3. Open, Close, high, and low stock prices of Goldman Sachs over time.

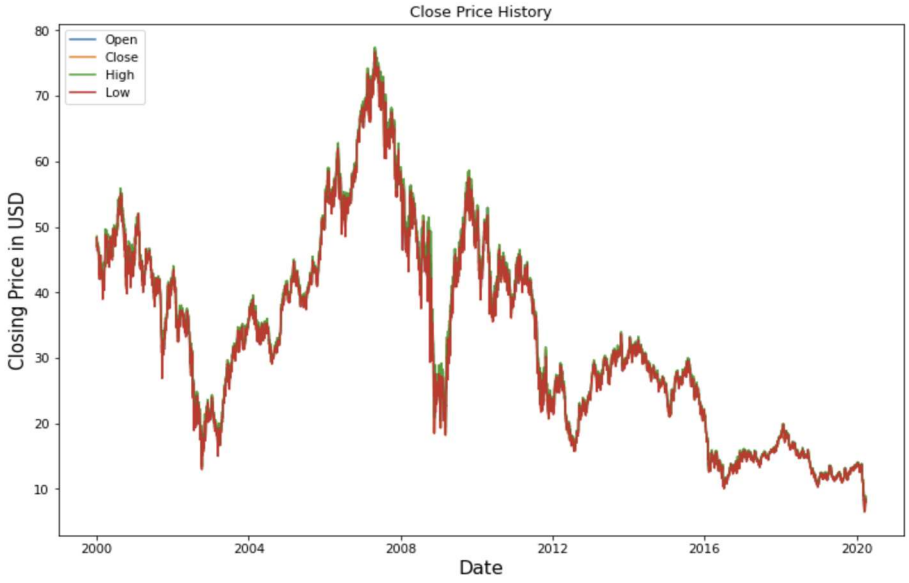

Figure 4. Open, Close, high, and low stock prices of Credit Suisse over time.

As it can be observed from the plots above, the Apple stock prices follow a more exponential trend whereas the ones for Goldman Sachs and Credit Suisse are more erratic. However, the Credit Suisse stock prices were higher between 2007 to 2010 and started getting lower after 2011. This shows the highly volatile nature of stock prices; all three firms are huge multinational conglomerates, but the stock prices from each company follows a trend of its own depending on several underlying factors. That was the reason for choosing closing stock prices from three known companies to see how the prediction accuracy changed with the Keras and TensorFlow LSTM neural networks.

B. Data Preprocessing and Transformations

For using the Keras-LSTM model, a new dataframe was produced by filtering only the closing stock prices, which was immediately converted to an array using the. values function. Then a variable was created to store the length of the dataset that consisted of 80% of the original dataset. Following the initial conversion, the data was transformed to have values between 0 and 1 by applying the *MinMaxScaler* module of Python's Scikit-learn library. Typically, it is considered good practice to scale data prior to feeding them into the target neural network. After scaling the

data, a training dataset was formed with historical 60-day closing price values as they were required to forecast the 61st closing price value. Basically, this implies that the first column in the *xtrain* dataset accommodated values from index O to index 59 (total of 60 values), the second column contained values from index 1 to index 60 (also a total of 60 values) and so forth. Accordingly, the *ytrain* dataset comprised the 61st value positioned at index 60 for its first column, the 62nd value located at index 61 for its second column and so on. The independent *xtrain* and dependent *ytrain* datasets were further transformed to numpy arrays so that they can be employed for training the LSTM structure. Prior to feeding the model with the input data, the *xtrain* dataset was reshaped in the 3-dimensional form [number of samples, number of time steps, and number of features] to meet the requirements of the Keras framework.

Similar to Keras, only the closing stock prices were filtered and converted to an array for the TensorFlow-LSTM model, but by using a combination of the *.loc* and *.value* functions. The array was then divided into training and testing datasets, where the first 80% of the array was the training set and the remaining 20% was the testing set. Both the training and testing values were scaled between 0 and 1 using the MinMaxScaler module and were later reshaped in the 2-dimensional form [data_size, num_features]. As it can be observed in Figure 2, the closing stock price values vary with time, i.e., the closing stock prices are different for different periods of time. Therefore, the training data was normalized by grouping the entire time series into windows. If windowing was not performed, then the older data would have been close to 0 and would not have contributed any value to the learning process. Thus, a window size of 1000 was chosen. It was made sure that a small window size was not chosen for windowed normalization as each window gets normalized independently and doing so would have inserted a break at the end of every window. Not every sample of the training data was normalized by the selected smoothing window size and hence, the last bit of the remaining data was normalized separately. Once this step was

complete, the training and testing datasets were reshaped back to the form of [data_size] so that they can be introduced into the TensorFlow structure. An important note is to be made here that only training data should be scaled as fitting the MinMaxScaler module to the test data would lead to inaccurate predictions.

C. Setting up the LSTM Neural Network

In order to build the LSTM neural network using Keras, a few dependencies were first imported. The dependencies are as follows:

- *from keras.models import Sequential*
- *from keras.layers import Dense, LSTM, Dropout*

The main goal of the Keras library in Python is to aid in the development of neural network models as a sequence of layers. The simplest model defined in the Sequential class is a linear stack of layers. Using the Sequential class, a model can be created in two ways; a Sequential model can be designed by defining all the layers in the constructor or the same task can be achieved by adding the necessary layers in the order of computation and conforming to the user's requirements. For simplicity and ease of understanding, the latter was performed in this thesis.

The LSTM model using Keras was initially designed with two LSTM layers, each having 50 neurons, and two dense layers, one with 25 neurons and the latter with a single neuron. The first layer in the LSTM model was specified with the shape of the input, which represented the number of input attributes and was defined by the input_dim argument. Suitably, the argument expected an integer. The dense layer in the structure suggests that the layers are fully connected by the neurons in the network layer, i.e., every neuron in the network accepts an input from all the neurons existing in the previous layers and thus, they are densely connected. Figure 5 below illustrates a typical densely connected layer of neurons.

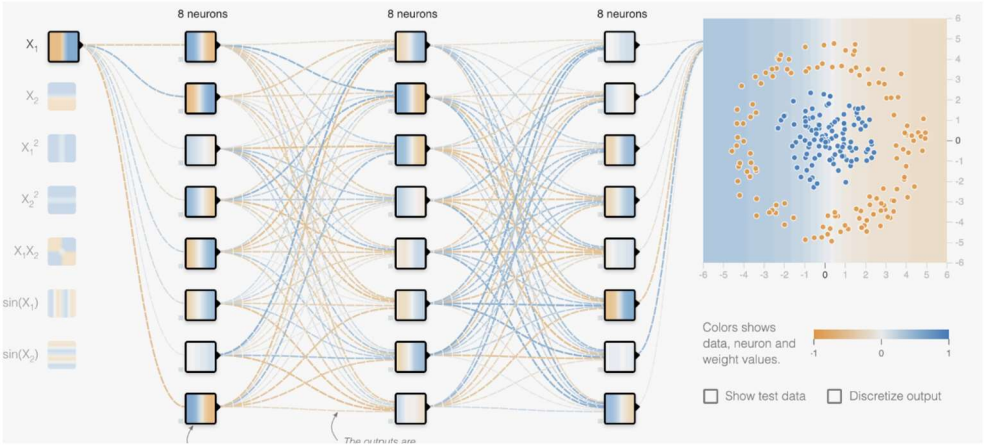

Figure 5. Illustration of a typical dense layer connected by neurons.

One of the most important reasons for why dense layers are preferred to convolutional layers is that a densely connected layer offers learning features from all the blends of the attributes of the previous layer, whereas a convolutional layer heavily depends on the constant features with a receptive field. Once defining the neural architecture was completed, the model was compiled using the MSE loss function and the Adam optimizer. Adam is an adaptive learning rate method [11] that combines the principles of RMSprop and Stochastic Gradient Descent (SGD) with momentum. Being an adaptive learning technique means that Adam calculates individual learning rates for varying parameters. Like RMSprop, it applies square gradients to scale learning rates while it benefits from momentum like SGD by using MAs of the gradient instead of the gradient itself. The *compile()* function accepts three considerable features including model optimizer, loss function, and metrics, but only the first two were sufficient for this thesis. Moreover, Adam reaps the advantages of AdaGrad, which operates exceptionally well in settings with sparse gradients, but grapples in non-convex optimization of neural networks. However, RMSprop, which aids to overcome some of the issues of Adagrad, works feasibly in online settings. Hence, the Adam optimizer was the ideal choice for the Keras-LSTM network.

Compiling the model in this way allowed for the formulation of an improved structure employed by the underlying backend (TensorFlow in this case) in order to execute the neural network during training in an efficient manner. For training, the model was fit onto both the *xtrain* and *ytrain* datasets. The batch size, which denoted the total number of training samples present in a single batch, and epochs, which signified the number of iterations when the entire training datasets were passed forward and backward through the neural network were specified in the *fit()* function. Following the training of the model, the test datasets, *xtest* and *ytest*, were created, where the independent *xtest* variable was transformed to a numpy array so that it could be implemented for testing the LSTM model. The numpy array was then reshaped to be 3-dimensional in the form [number of samples, number of time steps, and number of features] because, as it was mentioned earlier, the LSTM model was anticipating a 3-dimensional dataset. Finally, the transformed and reshaped test dataset was used to forecast the closing stock prices from the model. The same Keras-LSTM model was then implemented to predict the closing stock prices of Goldman Sachs and Credit Suisse, which have been tabulated in the Results section below.

LSTM is a very powerful time-series model with the capacity to forecast an unpredictable number of steps into the future. To reiterate the five essential components of LSTM that enable it to structure both long-term and short-term information are as follows [13]:

- Cell state (c_t,), which represents the internal memory of the cell that gathers both short-term and long-term data
- Hidden state (h_t), which contains the output state information computed with respect to the current input, previous hidden state, and current cell input, and is ultimately utilized to forecast future stock prices. Moreover, the hidden state has the ability to decide whether to retrieve only the short-term, long-term or both types of memory stored in the cell state to make the following, immediate prediction

- Input gate (c_t), which controls the flow of information from the current input to the cell state
- Forget gate (f_t), which decides the amount of information that is needed to flow from the current input and the previous cell state into the current cell state
- Output gate (o_t), which determines the amount of data that is required to flow from the current cell state into the hidden state, so that LSTM can only select the long-term or short-term and long-term memories.

In the TensorFlow-LSTM neural network, a datagenerator was first created and implemented to train the model. The data generator consisted of a method called *.unroll_batches()* that was responsible for generating a set of *num_rollings,* i.e., batches of input data procured sequentially, where a batch of data was of size [batch_size, 1]. For example, *num_unrollings = 3* and *batch_size = 4*, then the unrolled batches may appear in the format below:

- Input data: $[x_0, x_1 0, x_2 0, x_3 0], [x_1, x_1 1, x_2 1, x_3 1], [x_2, x_1 2, x_2 2, x_3 2]$
- Output data: $[x_1, x_1 1, x_2 1, x_3 1], [x_2, x_1 2, x_2 2, x_3 2], [x_3, x_1 3, x_2 3, x_3 3]$

Figure 6 below demonstrates how a batch of data is usually created.

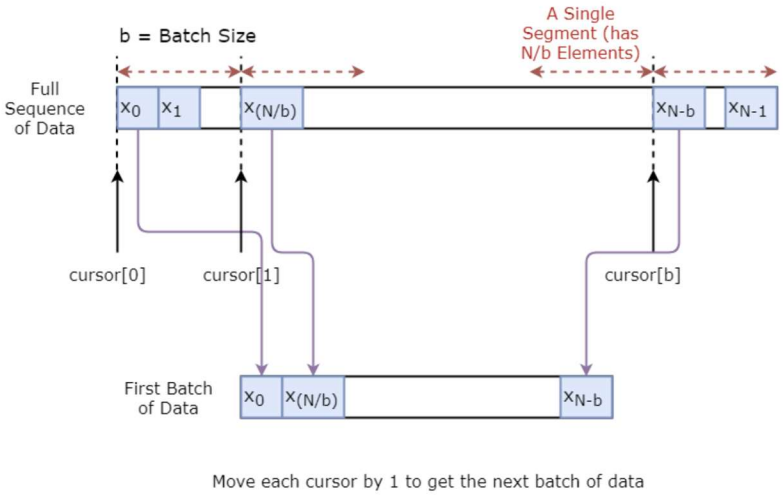

Figure 6. Visual creation of a batch of data [13].

In the next step, several hyperparameters were defined. *D* represents the dimensionality of the input data. Since the primary task was to take the previous closing stock price as input and forecast the next price as output, the value of *D* was set to 1. The *num_unrollings* defined above was also used as one of the hyperparameters as it was pertinent to the backpropagation through time (BPTT) and could optimize the LSTM model. This hyperparameter basically denoted the number of continuous time steps that was to be considered for a single optimization step. This can be thought of as, instead of optimizing the model by looking at a single time step, the entire network can be optimized by considering *num_unrollings* time steps. The larger the *num_unrollings* parameter the better the model would be optimized. Then there was the *batch_size*, which determined the number of data samples to be considered in a single time step. The final hyperparameter to be defined was *num_nodes*, which represented the number of hidden neurons in each cell. Three layers of LSTM were implemented in the TensorFlow network.

Next, the placeholders for training inputs and labels were defined. This was very straightforward as there was a list of placeholders, where each placeholder consisted of a single batch of data, and the list contained *num_unrollings* placeholders, that was used for a single optimization step. Three layers of LSTM and a linear regression layer, represented by w and b, were used. The regression layer was implemented to take the output of the LSTM memory cell and forecast the next time step. The *MultiRNNCell* module in TensorFlow can be applied to enclose the three LSTM cells that were formed. Moreover, dropout layers were added after each LSTM layer to enhance performance and diminish overfitting. Dropout is a technique that is popularly used to prevent overfitting and facilitates a way of approximately integrating exponentially several distinct neural network architectures efficiently. According to [11], the term dropout refers to dropping out units (hidden and visible) in a neural network. Dropping out a unit from a neural network implies that the unit is temporarily removed from the network along with all its incoming and outgoing

connections. The selection of which units to drop is arbitrary. During dropout, each unit is preserved with a fixed probability *p* independent of other units, where *p* can be chosen using a validation set or can simply be set at 0.5, which would appear to be close to feasible for a broad range of networks and tasks. For this thesis, the said probability was set at 0.2.

In the following section, TensorFlow variables *c* and *h* were first created to hold the cell and hidden states of the Long Short-Term Memory cell. The list of *train_inputs* were then converted to the shape of [num_unrollings, batch_size, D], which was required for computing the LSTM results with the *tf.nn.dynamic_rnn* function and divide the outputs back to a list of *num_unrolling* tensors, the loss between the forecasts and true stock prices. After this step, the loss was calculated. However, it was observed that there was a distinct aspect when the loss was calculated. The Mean Squared Error (MSE) was calculated for each batch of predictions and true outputs. Finally, the optimizer was defined that was ultimately implemented to optimize the overall performance of the neural network. In this thesis, the Adam optimizer was utilized as it is an extremely recent and well-performing optimizer. Furthermore, with the results produced from the experiments in Keras, it will be evident that Adam produces a lower RMSE value than any other optimizer. For the prediction related calculations, the prediction related TensorFlow operations were defined. Initially, a placeholder was defined for providing the (sample_inputs), denoted by *sample_in*, then like the training stage, the state variables (sample_c and sample_h), denoted by *c_smple* and h_smple, were defined to facilitate the predictions. The predictions were computed by implementing the *tf.nn.dynamic_rnn* function, which were later transmitted through the regression layers *w* and *b*. Additionally, the *reset_states_smple* operation was created to reset the cell and hidden states. It is recommended to carry out the operation at the beginning of each sequence of predictions.

In the part that runs the LSTM neural network, the model was trained, and the stock price movements were forecasted for several epochs to check whether the predictions improve or get worse over time. The detailed procedure outlined below was followed:

- A test set of starting points was defined, denoted by *test_pnts_seq*, was defined on the time series in order to analyze the model
- For every epoch:
    - A group of *num_unrollings* was unrolled
    - The neural network architecture was trained using the *num_unrollings* batches
    - The training average loss was computed
    - For every starting point in the test set:
        - The LSTM state was updated by back propagating through the previous num_unrolled sample data points discovered prior to the test point
        - Forecasts were generated for *n_pred_once* steps constantly by employing the prior prediction as the current input
        - The MSE loss between the *n_pred_once* points forecasted and the true stock prices at those time stamps were calculated

III. Results

A. Stock Price Prediction Results Using Keras

The ultimate goal here is to demonstrate the performance of the Keras-LSTM neural network model under varying conditions. To reiterate, the closing stock prices of three major companies were employed to achieve the final results. The stock prices of these companies were chosen due to their highly volatile behavior, where Apple stock prices follow an exponentially increasing trend, Goldman Sachs stock prices follow one of the most erratic trends and Credit Suisse stock prices follow a trend in which they reach peak value approximately between 2007 and 2008, drops below $20 around 2009 and 2010, consecutively, rises above $50 prior to 2012 and then finally keep decreasing until they fall below $10 close to 2020. As it can be observed from Tables 1, 2, and 3, the series of experiments were conducted by altering crucial parameters in the Keras model such as optimizer, batch size, epoch, LSTM layers, LSTM layer units, dropout layers, and training sample size.

1) Analysis of Predicted Apple Stock Prices

Table 1. Experimental RMSE values of Apple stock prices.

Item	Test Cases	RMSE
1	Original parameters; optimizer = 'adam', 'sgd', 'rmsprop', 'adadelta', 'nadam'	5.8500, 12.1144, 4.6872, 128.6728, 8.5247
2	Original parameters with Adam optimizer; batch_size = 1; epochs = 1, 10, 20, 30	5.8500, 7.4353, 9.5243, 8.0036
3	Original parameters with Adam optimizer; batch_size = 16; epochs = 1, 10, 20, 30	7.8762, 6.3305, 4.5493, 5.6321
4	Original parameters with Adam optimizer; batch_size = 32; epochs = 1, 10, 20, 30	13.2695, 6.5739, 5.4941, 4.6498
5	Original parameters with Adam optimizer; batch_size = 64; epochs = 1, 10, 20, 30	9.9078, 8.0643, 6.9585, 4.8536
6	Adding a dropout layer (0.2) after each LSTM layer; batch_size = 1; epochs = 1, 10, 20, 30	6.6956, 4.7437, 3.9904, 4.0017
7	Adding a 3rd LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	9.6158, 4.4359, 10.8195, 18.7184
8	Adding a 4th LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	11.4377,10.0069, 17.8406, 27.0081

Item	Test Cases	RMSE
9	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 1, 16, 32, 64; epochs = 1	14.1290, 12.0564, 12.7879, 18.5972
10	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 60, 100, 250, 500	9.2927, 8.4733, 8.7675, 11.7346
11	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 50, 30, 20, 100.	15.7208, 5.4972, 7.2283, 7.3388
12	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 25, 20, 15, 10.	8.3170, 8.1463, 8.3622, 8.9993
13	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 1, 45, 55, 60.	16.3818, 6.4220, 8.9432, 9.2874
14	1st LSTM layer (units = 256); 2nd LSTM layer (units = 256); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 256); 1st Dense layer (units = 1); batch_size = 32; epochs = 30, 45, 55, 60.	6.3412, 5.2388, 5.8134, 5.7812
15	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 1); batch_size = 64; epochs = 30, 50, 100, 150.	6.7294, 5.5440, 4.1147, 12.9159
16	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 25); batch_size = 64; epochs = 50, 55, 60, 65, 100.	6.4775, 4.9053, 4.5826, 5.7833, 7.5841
17	Adding more data, 3090 samples; batch_size = 128; epochs = 100, 150	6.9900, 6.5353

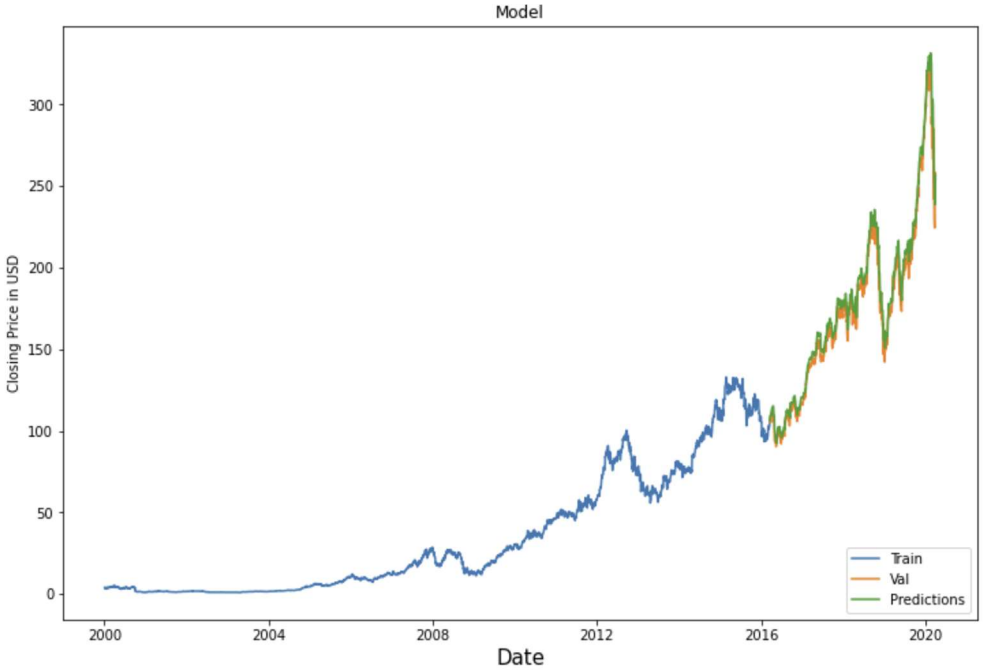

Figure 7. Trained, validated, and predicted values of Apple stock prices over time.

2) Analysis of Predicted Goldman Sachs Stock Prices

Table 2. Experimental RMSE values of Goldman Sachs stock prices.

Item	Test Cases	RMSE
1	Original parameters; optimizer = 'adam', 'sgd', 'rmsprop', 'adadelta', 'nadam'	5.8089, 9.1472, 14.0039, 103.4144, 5.5782
2	Original parameters with Adam optimizer; batch_size = 1; epochs = 1, 10, 20, 30	5.8089, 3.5732, 3.5457, 3.3806
3	Original parameters with Adam optimizer; batch_size = 16; epochs = 1, 10, 20, 30	8.2720, 6.4140, 4.3572, 4.1412
4	Original parameters with Adam optimizer; batch_size = 32; epochs = 1, 10, 20, 30	9.6328, 6.8875, 6.3929, 4.7078
5	Original parameters with Adam optimizer; batch_size = 64; epochs = 1, 10, 20, 30	12.9096, 7.2441, 6.5448, 6.2632
6	Adding a dropout layer (0.2) after each LSTM layer; batch_size = 1; epochs = 1, 10, 20, 30	6.4062, 4.1351, 3.5559, 5.4839
7	Adding a 3rd LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	7.4688, 4.2605, 4.2814, 4.6265
8	Adding a 4th LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	8.9789, 14.0404, 5.8253, 3.4023

Item	Test Cases	RMSE
9	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 1, 16, 32, 64; epochs = 1	9.7180, 10.3743, 10.5675, 13.1048
10	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 60, 100, 250, 500	4.8512, 3.7102, 5.3475, 3.8299
11	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 50, 30, 20, 100.	4.0847, 5.4576, 6.2475, 3.5155
12	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 25, 20, 15, 10.	6.6073, 7.0614, 8.3633, 8.8069
13	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 1, 45, 55, 60.	10.3710, 4.9079, 4.0774, 3.7505
14	1st LSTM layer (units = 256); 2nd LSTM layer (units = 256); 3rd LSTM Layer (units = 256); 4th LSTM layer (units = 256); 1st Dense layer (units = 1); batch_size = 32; epochs = 30, 45, 55, 60.	6.4176, 5.3616, 5.0465, 5.2465
15	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 1); batch_size = 64; epochs = 30, 50, 100, 150.	6.4583, 5.6676, 3.7499, 3.4742
16	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 25); batch_size = 64; epochs = 50, 55, 60, 65, 100.	6.1191, 5.2765, 6.1513, 5.1523, 4.2865
17	Adding more data, 3090 samples; batch_size = 128; epochs = 100, 150	5.4087, 3.2947

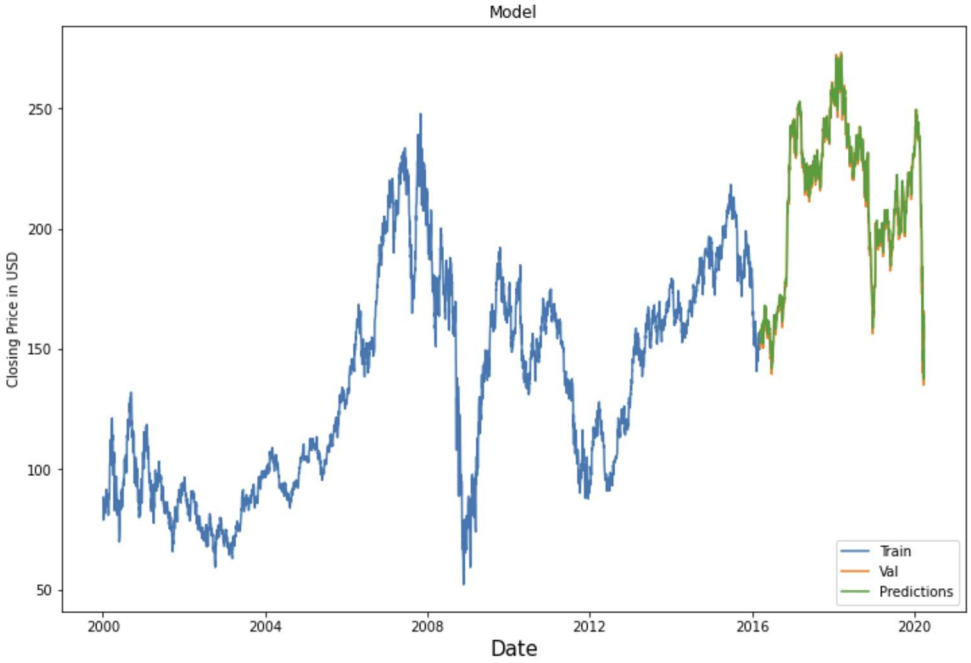

Figure 8. Trained, validated, and predicted values of Goldman Sachs stock prices over time.

3) Analysis of Predicted Credit Suisse Stock Prices

Table 3. Experimental RMSE values of Credit Suisse stock prices.

Item	Test Cases	RMSE
1	Original parameters; optimizer = 'adam', 'sgd', 'rmsprop', 'adadelta', 'nadam'	0.3266, 0.9237, 0.4290, 1.7914, 0.6169
2	Original parameters with Adam optimizer; batch_size = 1; epochs = 1, 10, 20, 30	0.3266, 0.4558, 0.1982, 0.2252
3	Original parameters with Adam optimizer; batch_size = 16; epochs = 1, 10, 20, 30	0.6169, 0.3276, 0.2672, 0.2729
4	Original parameters with Adam optimizer; batch_size = 32; epochs = 1, 10, 20, 30	0.6790, 0.4649, 0.6839, 0.4346
5	Original parameters with Adam optimizer; batch_size = 64; epochs = 1, 10, 20, 30	0.9640, 0.6006, 0.4010, 0.3649
6	Adding a dropout layer (0.2) after each LSTM layer; batch_size = 1; epochs = 1, 10, 20, 30	0.5641, 0.2702, 0.1971, 0.3326
7	Adding a 3rd LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	0.5519, 0.4235, 0.4865, 0.2909
8	Adding a 4th LSTM layer (unit = 50); dropout layer (0.2); batch_size = 1; epochs = 1, 10, 20, 30	1.6986, 0.2652, 0.9404, 0.8893

Item	Test Cases	RMSE
9	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 1, 16, 32, 64; epochs = 1	1.4155, 1.0554, 0.7880, 1.2424
10	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 60, 100, 250, 500	0.2497, 0.4048, 0.8442, 0.4113
11	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 50, 30, 20, 100. **Performed on Google Colab**	1.0051, 0.8698, 0.5337, 0.2196
12	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 32; epochs = 25, 20, 15, 10. **Performed on Google Colab**	0.4432, 0.5623, 0.4226, 0.8142
13	1st LSTM layer (units = 100); 2nd LSTM layer (units = 100); 3rd LSTM Layer (units = 100); 4th LSTM layer (units = 100); 1st Dense layer (units = 25); batch_size = 16; epochs = 1, 45, 55, 60. **Performed on Google Colab**	0.6807, 0.7692, 0.4215, 0.2479
14	1st LSTM layer (units = 256); 2nd LSTM layer (units = 256); 3rd LSTM Layer (units = 256); 4th LSTM layer (units = 256); 1st Dense layer (units + 1); batch_size = 32; epochs = 30, 45, 55, 60. **Performed on Google Colab**	0.5375, 0.3189, 0.3207, 0.4790
15	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 1); batch_size = 64; epochs = 30, 50, 100, 150. **Performed on Google Colab**	0.4277, 0.5670, 0.8983, 0.6399
16	1st LSTM layer (units = 512); 2nd LSTM layer (units = 512); 3rd LSTM Layer (units = 512); 4th LSTM layer (units = 512); 1st Dense layer (units = 25); batch_size = 64; epochs = 50, 55, 60, 65, 100. **Performed on Google Colab**	0.6665, 0.4082, 0.6483, 0.2792, 0.3181
17	Adding more data, 3090 samples; batch_size = 128; epochs = 100, 150 with GPU setting on **Google Colab**	1.4516, 0.3862

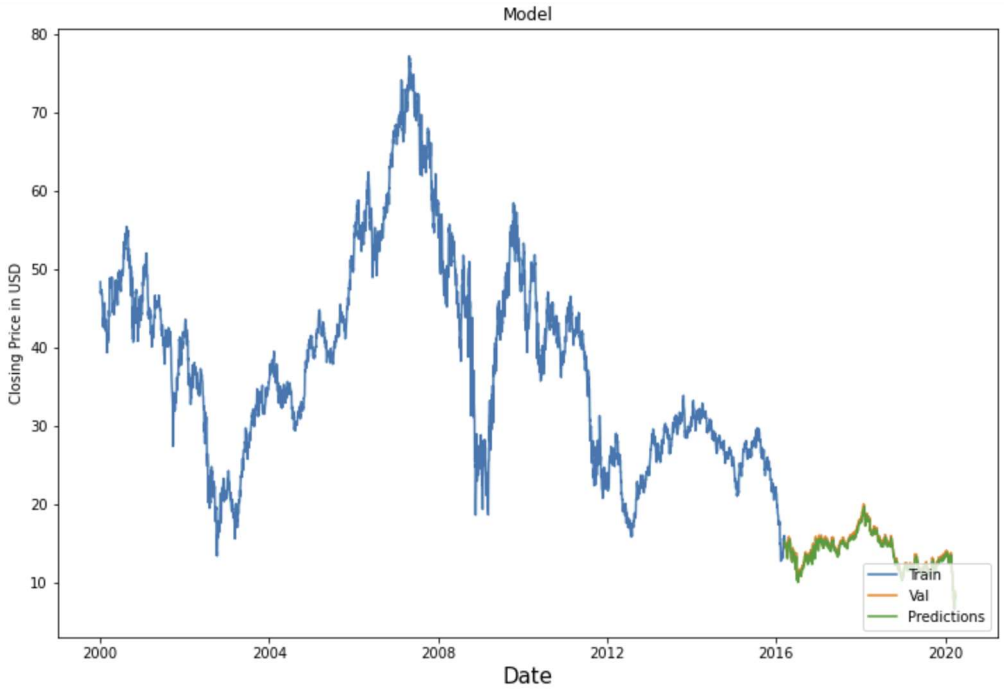

Figure 9. Trained, validated, and predicted values of Credit Suisse stock prices over time.

4) Discussion

In the tables above, the highest and lowest achieved RMSE values are highlighted in red and green, respectively. The reference RMSE values obtained from using all original parameters were 5.8500 for Apple, 5.8089 for Goldman Sachs, and 0.3266 for Credit Suisse, which were eventually considered as target thresholds for determining the lowest achieved RMSE values with the best possible combination of parameters. The first item involved using different optimizers. In all three cases, the *adadelta* optimizer produced the RMSE values, whereas *rmsprop* produced the lowest RMSE value for Apple, *nadam* for Goldman Sachs, and *adam* for Credit Suisse. Adadelta is a more robust extension of the Adagrad optimizer that is best suited for learning rates based on a moving window of gradient updates, rather than an amalgamated version of all historical gradients. This aspect of Adadelta might have accounted for the generation of high RMSE values. On the contrary, Nadam is basically RMSprop with Nesterov momentum, like Adam is crucially RMSprop is regular momentum. Hence, the name Nadam that stands for Nesterov Adam optimizer.

Items 2 through 5 consisted of changing the batch size while training the Keras model through 1, 10, 20, and 30 epochs. Batch size represents the number of sample training data points to be considered in each epoch. For example, a batch of size 16 applied to a training dataset of size 2003 implies that the algorithm takes 16 data samples (1st to 16th) from the training dataset and trains the neural network. Next, it takes the following 16 data points (17th to 32nd) from the training set and trains the network again, and so on until the algorithm has propagated through all the specified number of epochs. Epoch represents the number of iterations that a neural network algorithm propagates through each training procedure. Looking at Tables 1 and 3, altering the batch size has an erratic effect on the predicted stock prices, which is evident from the obtained RMSE values, however, using a batch size of 64 sequentially reduces the RMSE values as the algorithm propagates through each epoch for 10, 20, and 30 epochs respectively. On the other hand, items 2 through 5 in Table 2 illustrates that as the batch size increases, the performance of the algorithm improves as the RMSE values gradually decrease.

In items 6 through 8, three more LSTM layers were added along with respective dropout layers. In each case, the batch size was kept as 1, but the neural network was trained for epochs 10, 20, and 30. According to Table 1 item 6, the RMSE value decreases and then fluctuates in the mid-4 range while reaching the lowest value of 3.9904. Items 7 and 8 in Table 1, initially reduce in RMSE value, but later increase exponentially generating high values of 18.7184 and 27.0081. Unlike Table 1, items 6, 7, and 8 in Table 2 exhibit a rather unpredictable pattern as the values fluctuate between 3.4023 and 14.0404. Moreover, item 8 produces only one high value while the remainder of the values are below 9. Items 6 through 8 in Table 3 exhibit similar behavioral characteristics, but the values, including the highest and lowest values are much better than the ones in Tables 1 and 3, which implies that so far the algorithm works best with Credit Suisse closing stock prices. For the remainder of the experiments, the parameters were changed by adding more

neurons to all the four LSTM layers while keeping the number of neurons in the Dense layers unchanged. The batch sizes were constrained within five values, 1, 16, 32, 64, and 128, but they seemed to have minimal effect on the RMSE values and hence, the epoch sizes were selected from a broader range to achieve the best possible prediction accuracy. In the last steps of each experiment with Keras, the sample size was increased by 3090 data points. However, in all three cases, simply adding more sample data points proved to be ineffective in achieving the best possible performance accuracy. While all the experiments with Keras produced results with unpredictable patterns, there was one common pattern that was very apparent among the analyses. A closer look at the first three tables, would reveal that the Keras-LSTM neural network model performed better with Credit Suisse stock prices than with both Apple and Goldman Sachs stock prices while the model with Goldman Sachs stock prices performed better than the one with Apple stock prices and worse than the one with Credit Suisse stock prices. Moreover, both Apple and Credit Suisse stock prices achieved target results at the same neural network configuration. Figures 7, 8, and 9 further illustrate the performance of the Keras framework as the predicted stock prices almost perfectly overlap the sample validation points, which represent the test dataset. Table 4 below summarizes the results extracted from Tables 1, 2, and 3.

Table 4. Summary of the Keras experimental Analyses.

	Best Combination of Parameters	**Highest RMSE Value at Best Calibration**	**Lowest RMSE Value at Best Calibration**
Apple	Adding a dropout layer (0.2) after each LSTM layer; batch_size = 1; epochs = 1, 10, 20, 30	6.6956	3.9904
Goldman Sachs	Original parameters with Adam optimizer; batch_size = 1; epochs = 1, 10, 20, 30	5.8089	3.3806
Credit Suisse	Adding a dropout layer (0.2) after each LSTM layer; batch_size = 1; epochs = 1, 10, 20, 30	0.5641	0.1971

B. Stock Price Prediction Results Using TensorFlow

TensorFlow 1.15.2 was used for this thesis. TensorFlow 2.0 and above are the newer versions, which use the Keras module for sequential modeling. Keras is not available in the older versions of TensorFlow and hence, prior to conducting the experiments the TensorFlow version on Google Colab was downgraded to the older version to support *contrib* and *placeholder* operations. The dataset for the next series of experiments ranged from January 2000 to December 2019 because any dataset below this size would cause *MinMaxScaler()* to throw out an error for the dataset being "too small." The only parameters that were altered for this series of experiments were *num_unrollings*, *batch_size*, *num_nodes*, GPU/TPU (tensor processing unit) setting on Google Colab and the size of the datasets. Tables 5, 6, and 7 list the results for Apple, Goldman Sachs, and Credit Suisse stock prices.

1) Analysis of Predicted Apple Stock Prices

Table 5. Experimental MSE and RMSE values of Apple stock prices with TensorFlow 1.0.

Item No.	Test Case	Best MSE	Best RMSE
1	Num_unrollings = 50; batch_size = 500	6.0651	1.9875
2	Num_unrollings = 100; batch_size = 500	5.8815	2.9085
3	Num_unrollings = 100; batch_size = 1000	6.7245	2.1126
4	Num_unrollings = 150; batch_size = 1000	6.6485	2.8495
5	Num_unrollings = 150; batch_size = 1500	6.4438	2.4093
6	Num_unrollings = 200;; batch_size = 1500	6.3992	2.6199
7	Num_unrollings = 200; batch_size = 2000; Hardware Accelerator = TPU	6.5986	2.6208
8	Num_unrollings = 500; batch_size = 2000; Hardware Accelerator = TPU	System crashed at this setting when using only closing stock prices	
9	Num_unrollings = 1000; batch_size = 2000; Hardware Accelerator = TPU	System crashes at this setting	
10	Num_unrollings = 600; batch_size = 2000; Hardware Accelerator = TPU	System crashed at this setting when using only closing stock prices	
11	Num_unrollings = 400; batch_size = 500; Hardware Accelerator = GPU	6.9385	2.4045
12	Num_unrollings = 250; batch_size = 500; Hardware Accelerator = GPU	6.1513	2.7941
13	Num_unrollings = 50; batch_size = 500; num_nodes = [250, 250, 200]; Hardware Accelerator = GPU	5.2750	1.5530
14	Num_unrollings = 200; batch_size = 1000; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU	5.5439	1.8547

Item No.	Test Case	Best MSE	Best RMSE
15	Num_unrollings = 400; batch_size = 1000; num_nodes = [300, 300, 250]; Hardware Accelerator = TPU; Runtime shape = High-RAM	5.3648	2.1598
16	Num_unrollings = 300; batch_size = 800; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU; Runtime shape = High-RAM, Increase dataset size by 3033 sample points	1572.0055	4.6779
17	Num_unrollings = 400; batch_size = 1000; num_nodes = [300, 300, 250]; epochs = 40; Hardware Accelerator = TPU; Runtime shape = High-RAM; Dataset size = +3033	1613.2532	3.3110

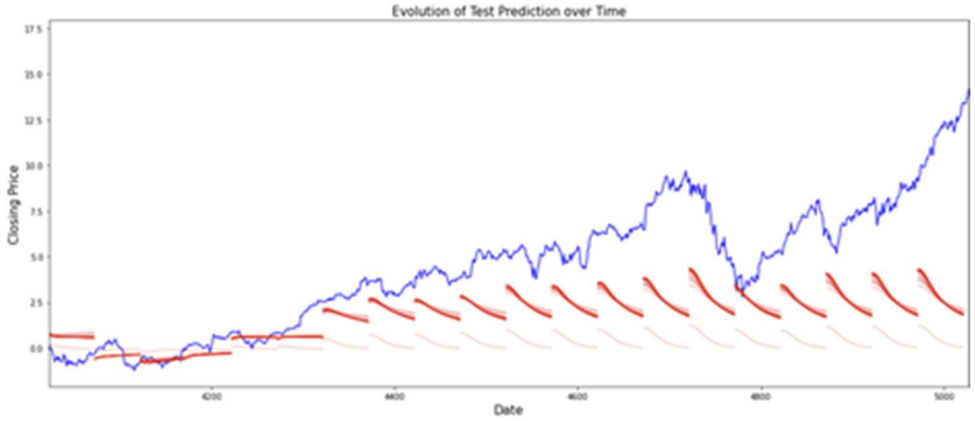

Figure 10a: Evolution of Test Predictions over Time (Apple)

Figure 10b: Best Test Predictions over Time (Apple)

2) Analysis of Predicted Goldman Sachs Stock Prices

Table 6. Experimental MSE and RMSE values of Goldman Sachs stock prices with TensorFlow 1.0.

Item No.	Test Case	Best MSE	Best RMSE
1	Num_unrollings = 50; batch_size = 500	0.5788	3.1710
2	Num_unrollings = 100; batch_size = 500	0.6550	2.5342
3	Num_unrollings = 100; batch_size = 1000	0.8921	1.6570
4	Num_unrollings = 150; batch_size = 1000	0.9722	1.4999
5	Num_unrollings = 150; batch_size = 1500	0.7850	2.7502
6	Num_unrollings = 200;; batch_size = 1500	0.7182	2.6387
7	Num_unrollings = 200; batch_size = 2000; Hardware Accelerator = GPU	0.8831	2.2070
8	Num_unrollings = 500; batch_size = 2000; Hardware Accelerator = GPU	System crashes at this setting. Can no longer use TPU	
9	Num_unrollings = 1000; batch_size = 2000; Hardware Accelerator = TPU	System crashes at this setting. Can no longer use TPU	
10	Num_unrollings = 600; batch_size = 2000; Hardware Accelerator = GPU	System crashes at this setting. Can no longer use TPU	
11	Num_unrollings = 400; batch_size = 500; Hardware Accelerator = GPU	System crashes at this setting. Can no longer use TPU	
12	Num_unrollings = 250; batch_size = 500; Hardware Accelerator = GPU	0.7045	2.2491
13	Num_unrollings = 50; batch_size = 500; num_nodes = [250, 250, 200]; Hardware Accelerator = GPU	0.8403	3.1435
14	Num_unrollings = 200; batch_size = 1000; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU	0.6709	1.5788
15	Num_unrollings = 300; batch_size = 800; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU; Runtime shape = High-RAM, Increase dataset size by 169 sample points	1.1272	1.5705

Item No.	Test Case	Best MSE	Best RMSE
16	Num_unrollings = 300; batch_size = 700; num_nodes = [300, 300, 250]; epochs = 50; Hardware Accelerator = TPU; Runtime shape = High-RAM, Increase dataset size by 169 sample points	0.9888	1.3422
17	Num_unrollings = 350; batch_size = 600; num_nodes = [350, 350, 300]; epochs = 60; Hardware Accelerator = TPU; Runtime shape = High-RAM	0.6953	1.1207

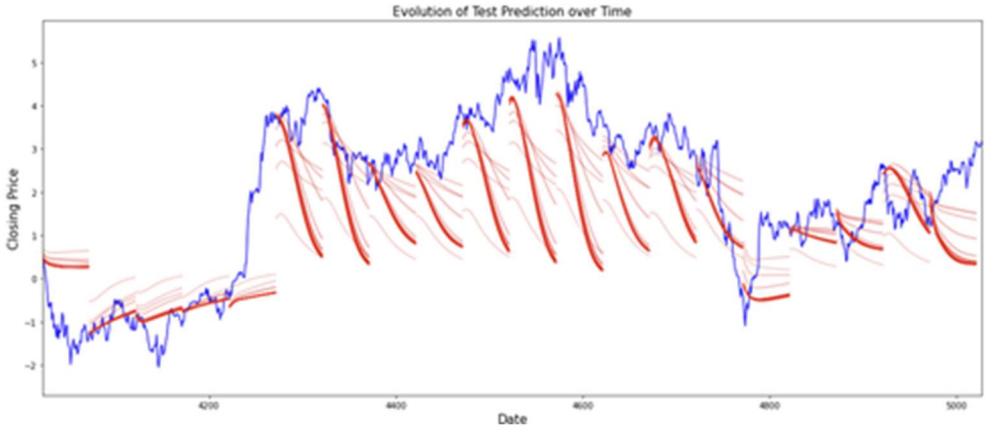

Figure 11a: Evolution of Test Prediction over Time (Goldman Sachs)

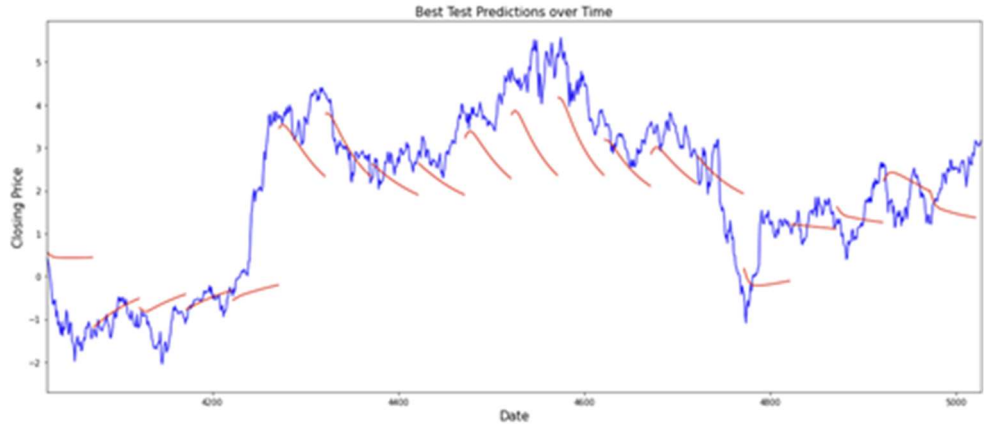

Figure 11b: Best Test Predictions over Time (Goldman Sachs)

3) Analysis of Predicted Credit Suisse Stock Prices

Table 7. Experimental MSE and RMSE values of Credit Suisse stock prices with TensorFlow 1.0.

Item No.	Test Case	Best MSE	Best RMSE
1	Num_unrollings = 50; batch_size = 500	2.04372	2.1744
2	Num_unrollings = 100; batch_size = 500	2.0503	1.8102
3	Num_unrollings = 100; batch_size = 1000	3.2210	0.8986
4	Num_unrollings = 150; batch_size = 1000	3.2936	0.9636
5	Num_unrollings = 150; batch_size = 1500	2.4319	0.5646
6	Num_unrollings = 200;; batch_size = 1500	2.1885	0.8292
7	Num_unrollings = 200; batch_size = 2000; Hardware Accelerator = GPU	2.5912	0.6725
8	Num_unrollings = 500; batch_size = 2000; Hardware Accelerator = GPU	System crashed at this setting	
9	Num_unrollings = 1000; batch_size = 2000; Hardware Accelerator = TPU	System crashed at this setting	
10	Num_unrollings = 600; batch_size = 2000; Hardware Accelerator = GPU	System crashed at this setting	
11	Num_unrollings = 400; batch_size = 500; Hardware Accelerator = GPU	1.0946	1.6757
12	Num_unrollings = 250; batch_size = 500; Hardware Accelerator = GPU	1.5971	1.9597
13	Num_unrollings = 50; batch_size = 500; num_nodes = [250, 250, 200]; Hardware Accelerator = GPU	1.2071	1.7699
14	Num_unrollings = 200; batch_size = 1000; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU	1.5732	1.9461

Item No.	Test Case	Best MSE	Best RMSE
15	Num_unrollings = 300; batch_size = 800; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU; Runtime shape = High-RAM, Increase dataset size by 1170 sample points	6.9071	1.1436
16	Num_unrollings = 300; batch_size = 700; num_nodes = [300, 300, 250]; epochs = 50; Hardware Accelerator = TPU; Runtime shape = High-RAM, Increase dataset size by 1170 sample points	6.2816	0.8478
17	Num_unrollings = 350; batch_size = 600; num_nodes = [350, 350, 300]; epochs = 60; Hardware Accelerator = GPU; Runtime shape = High-RAM	0.4666	0.8988

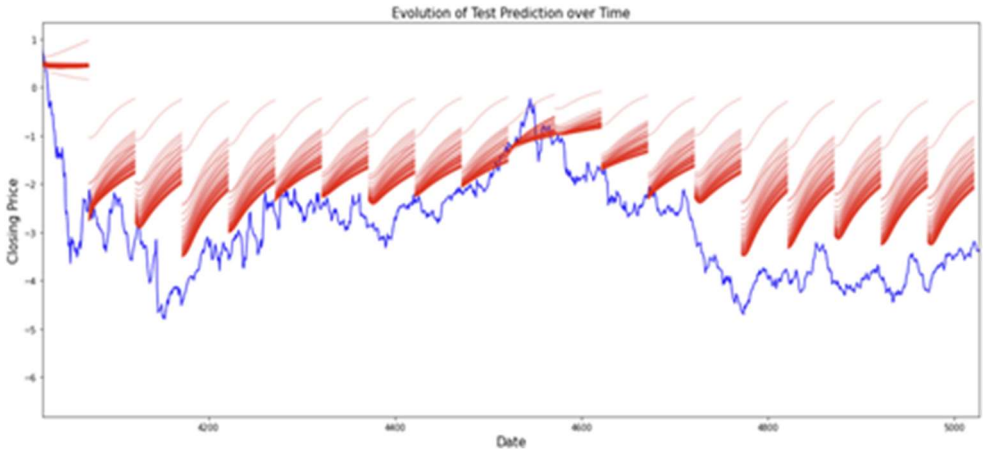

Figure 12a: Evolution of Test Prediction over Time (Credit Suisse)

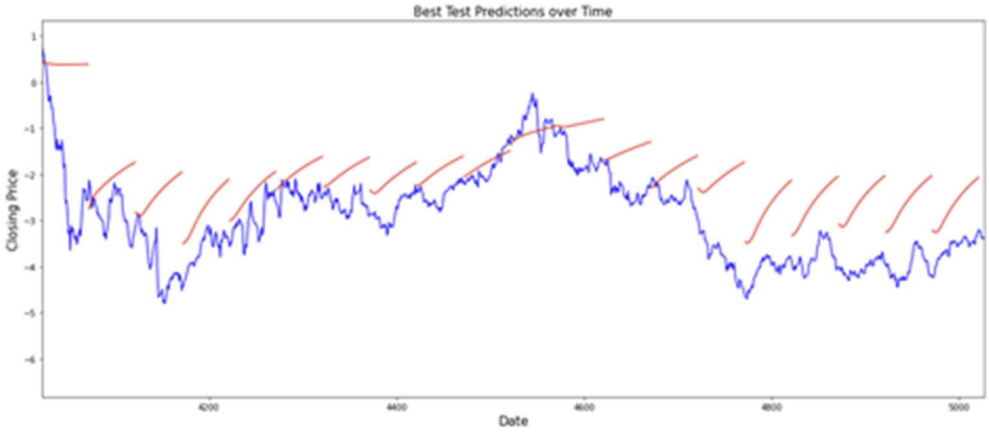

Figure 12b: Best Test Predictions over Time (Credit Suisse)

4) Discussion

The GPU hardware accelerator in Google Colab facilitates accelerated cloud computing services, which makes it best suited for computers with NVIDIA GPUs while TPU offers powerful performance and four times larger in-memory capacity than NVIDIA's best GPU V100. However, one major difference between the two hardware accelerators is that TPU's training time is longer with smaller batch sizes than GPU's training time, whereas both hardware accelerators perform almost the same with larger batch sizes. For the stock prices from all three companies, the TensorFlow-LSTM neural network produced fluctuating accuracy. Changing the hardware accelerator from GPU-TPU-GPU had insignificant impact on the overall performance of the model. However, the TPU accelerator seemed to crash at certain configurations, especially with larger *num_unrollings* and batch sizes. Adding more sample data points to train the neural network caused the forecasted Apple stock price movements to reach extremely high MSE values, whereas the forecasted Goldman Sachs and Credit Suisse stock price movements generated acceptably low MSE and RMSE values. Nonetheless, from Tables 5, 6, and 7, it is very evident that the model worked best with Goldman Sachs stock prices as the best MSE over time was constrained within

2.000. On the other hand, the model worked better with Credit Suisse stock prices than with Apple stock prices, but worse than with Goldman Sachs stock prices. Figures 10a, 10b, 11a, 11b, 12a, and 12b illustrate the actual stock price movements, evolution of test predictions, and best test predictions over time. The results are summarized in Table 8 below.

Table 8. Summary of the TensorFlow experimental analyses.

	Best Combination of Parameters	**Highest MSE**	**Highest RMSE**	**Lowest MSE**	**Lowest RMSE**
Apple	Num_unrollings = 400; batch_size = 1000; num_nodes = [300, 300, 250]; epochs = 40; Hardware Accelerator = TPU; Runtime shape = High-RAM, Increase dataset size by 3033 sample points	1613.2532	3.3110	5.2750	1.5530
Goldman Sachs	Num_unrollings = 300; batch_size = 800; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU; Runtime shape = High-RAM, Increase dataset size by 169 sample points	1.1272	1.5705	0.6550	2.5342
Credit Suisse	Num_unrollings = 300; batch_size = 800; num_nodes = [300, 300, 250]; Hardware Accelerator = GPU; Runtime shape = High-RAM, Increase dataset size by 1170 sample points	6.9071	1.1436	0.4666	0.8988

With the TensorFlow, the predictions were based on whether the past recent values were moving up or down in value (not the actual value). For example, the predictions imply that the next day prices are likely to be lower, if the prices have been dropping in the past few days and higher if the prices have been increasing. The blue fluctuating curves in Figures 10a, 10b, 11a, 11b, 12a, and 12b signify by how much stock prices fall and rise, whereas the broken red curves represent

the predicted movement of stock prices over time. The graphs representing Apple stock prices agree with the results in Table 5. After the first few years, the predicted price movements fail to accurately represent the actual price movements, especially at times when the prices rise steeply and drop drastically. However, for Goldman Sachs stock prices, the predicted movements intercept the actual movement at every epoch and thus, reveal that as the actual price falls the predicted price is likely to fall and vice versa. An approximate behavior is illustrated by the Credit Suisse stock price movements in Figures 12a and 12b, but with a lower prediction accuracy than for Goldman Sachs.

IV. Conclusion

The following section presents some future suggestions on enhancing the performance of the neural network models.

A. Future Recommendations

There are many ways to improve the performance of the neural network models. As it has been mentioned previously in the Literature Review section, stock market prices are not only influenced by the performance of their respective companies, but they are also impacted by several other factors such as political alterations, financial changes, budgeting issues, related news articles, public rumors, etc. Due to their reactions to the said changes, stock prices are typically considered as highly volatile entities that are massively difficult to forecast with the desired prediction accuracy. Thus, predicting future stock prices on just the open, low, high, or closing stock prices may not always be accurate and consequently, leading the predictions to highly fluctuate for similar configurations. A hybrid approach employing various types of filters can be used to filter out the fluctuations and train the neural networks based on the actual trends of stock prices. This methodology primarily involves considering stock prices as digital signals consisting of white Gaussian noise, i.e., unknown fluctuations, which needs to be removed in order to create smoothness in the filtered signal. Moreover, textual data containing relevant news articles and public rumors should also be accounted for by combining the filters and LSTM neural networks with natural language processing (NLP) techniques. Other than combining textual data and filtering methods, financial ratios can be employed as additional features to facilitate the training process of the neural networks and as a result enhance the prediction accuracy. Eventually, further research work should be conducted to find the best possible combination of features, parameters, and neural network layers to build a universal algorithm that can not only forecast future stock prices with any number of input features, but can also forecast future global recessions and allow enough time to

organizations to devise appropriate contingency plans to tackle the recession. This is a tool that the companies or stock market indexes do not have in their possession and thus, fail to accurately predict future recessions.

B. Final Remarks

To conclude, this thesis analyzes predictive models for stock price prediction by incorporating Keras and TensorFlow with the LSTM neural network architecture. Stock prices of three multinational conglomerates were extracted from Yahoo and a series of experiments were conducted using a variety of different parameters that could directly have an impact on the prediction accuracy of the Keras and TensorFlow powered models. Keras was simple to use and allowed the LSTM architecture to be designed within just a few lines of code with the help of the Sequential API, whereas the TensorFlow framework was overly complicated and hence, difficult to comprehend. As a result, Keras consumed lower runtime than TensorFlow. Due to the simplicity of the Keras framework, it is relatively easy to identify the parameters that can be tuned to alter the performance of the model, but TensorFlow's complexity allowed only a few parameters to be changed, mostly the batch size, *num_nodes,* and *num_unrollings parameters*. This thesis shows that for certain operations such as stock price prediction, TensorFlow does not always offer flexibility. Although TensorFlow and Keras were used to perform two types of predictions, it was clearly evident that Keras was able to generate stable results while TensorFlow's results were unstable, especially at certain configurations where the system crashed and failed to produce any result at all. Based on these observations, a recommendation can be made to use Keras as the desired Deep Learning tool for predicting future stock prices. TensorFlow seems to be better suited for predicting stock price movements, which will eventually aid to determine whether prices are likely to fall or rise in the future, but the generated results will more likely to be unstable and may cause some predicted movements to disagree with actual price movements.

V. References

[1] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao, "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets," *Computational Intelligence in Data-Driven Modelling and Its Engineering Applications*, Vol. 2018, April 2018.

[2] O. Hegazy, O. Soliman, and M. Salam, "A Machine Learning Model for Stock Market Prediction," *International Journal of Computer Science and Telecommunications*, Vol. 4, Issue 12, pp. 17-23, December 2013.

[3] S. Mohan, S. Mullapudi, S. Sammeta, P. Vijayvergia, and D. Anatasiu, "Stock Price Prediction Using News Sentiment Analysis," 2019 IEEE 5th International Conference on Big Data Computing Service and Applications (BigDataService), pp. 205-208, 2019.

[4] S. Liu, G. Liao, and Y. Ding, "Stock Transaction Prediction Modeling and Analysis Based on LSTM," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 2787-2790, 2018.

[5] Z. Li and V. Tam, "Combining the Real-Time Wavelet Denoising and Long-Short-Term-Memory Neural Network for Predicting Stock Indexes," Department of Electrical and Electronic Engineering - The University of Hong Kong, 2017.

[6] L. Sayavong, Z. Wu, and S. Chalita, "Research on Stock Price Prediction Method Based on Convolutional Neural Network," 2019 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), 2019.

[7] C. Lai, R. Chen, and R. Caraka, "Prediction Stock Price Based on Different Index Factors Using LSTM," 2019 International Conference on Machine Learning and Cybernetics (ICMLC), 2019.

[8] J. Du, Q. Li, K. Chen, and J. Wang, "Forecasting stock prices in two ways based on LSTM neural network," IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC 2019), 2019.

[9] S. Yao, L. Luo, and H. Peng, "High-Frequency Stock Trend Forecasting Using LSTM Model," 2018 13th International Conference on Computer Sciences & Education (ICCSE), 2018.

[10] Z. Zeng, Q. Gong, and J. Zhang, "CNN Model Design of Gesture Recognition Based on TensorFlow Framework," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC 2019), 2019.

[11] E. Dominic, “Understand TensorFlow by mimicking its API from scratch,” Medium: Machine Learning, Jan. 10, 2019. https://medium.com/@d3lm/understand-tensorflow-by-mimicking-its-api-from-scratch-faa55787170d

[12] Z. Jiang and G. Shen, “Prediction of House Price Based on The Back Propagation Neural Network in the Keras Deep Learning Framework,” 2019 6th International Conference on Systems and Informatics (ICSAI), 2019.

[13] DataCamp, “Stock Market Predictions with LSTM in Python.” DataCamp, Jan.1 2020, https://www.datacamp.com/community/tutorials/lstm-python-stock-market

[14] A. Shayka, A. Pokhrel, A. Bhattarai, P. Sitikhu, and S. Shakya, “Real-Time Stock Prediction using Neural Network,” 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 71-74, 2018. (Important)

[15] M. Quahilal, M. Mohajir, M. Chahhou, and B. Mohajir, “Optimizing Stock Market Price Prediction using a Hybrid Approach Based on HP Filter and Support Vector Regression,” 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt), pp. 290-294, 2016. (Important)

[16] T. Gao, Y, Chai, and Y. Liu, “Applying Long Short Term Memory Neural Networks for Predicting Stock Closing Price,” 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017. (Important)

[17] Medium, “Stock Price Prediction Using Python & Machine Learning,” Medium, Dec. 23, 2019,https://medium.com/@randerson112358/stock-price-prediction-using-python-machine-learning-e82a039ac2bb