UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

Financial time series forecasting applying deep learning algorithms

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:
Solís Garcés Erik David

Tutor:
Ph.D. Cuenca Erick

Urcuquí - Agosto 23, 2021

Urcuquí, 4 de agosto de 2021

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2021-00028-AD

A los 4 días del mes de agosto de 2021, a las 14:30 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. LEIVA , HUGO , Ph.D.
Miembro No Tutor	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.

El(la) señor(ita) estudiante **SOLIS GARCES, ERIK DAVID**, con cédula de identidad No. **1004037444**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **FINANCIAL TIME SERIES FORECASTING APPLYING DEEP LEARNING ALGORITHMS**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Presidente Tribunal De Defensa	Dr. LEIVA , HUGO , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.	9,0
Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.	10,0

Lo que da un promedio de: **9.7 (Nueve punto Siete)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

Certifico que *en cumplimiento del Decreto Ejecutivo 1017 de 16 de marzo de 2020, la defensa de trabajo de titulación (o examen de grado modalidad teórico práctica) se realizó vía virtual, por lo que las firmas de los miembros del Tribunal de Defensa de Grado, constan en forma digital.*

SOLIS GARCES, ERIK DAVID
Estudiante

Firmado electrónicamente por:
ERIK DAVID SOLIS

HUGO LEIVA
Firmado digitalmente por HUGO LEIVA
Fecha: 2021.08.06 10:47:21 -05'00'

Dr. LEIVA , HUGO , Ph.D.
Presidente Tribunal de Defensa

Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Tutor

ERICK EDUARDO CUENCA PAUTA
Digitally signed by ERICK EDUARDO CUENCA PAUTA
Date: 2021.08.06 09:39:33 -05'00'

Dr. ARMAS ARCINIEGA, JULIO JOAQUIN , Ph.D.
Miembro No Tutor

Firmado electrónicamente por:
JULIO JOAQUIN ARMAS ARCINIEGA

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

TATIANA BEATRIZ TORRES MONTALVAN
Firmado digitalmente por TATIANA BEATRIZ TORRES MONTALVAN
Fecha: 2021.08.06 09:28:37 -05'00'

Autoría

Yo, **Erik Solís**, con cédula de identidad **1004037444**, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Agosto del 2021.

Firmado electrónicamente por:
ERIK DAVID SOLIS

Erik Solís
CI: 1004037444

Autorización de publicación

Yo, **Erik Solís**, con cédula de identidad **1004037444**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Agosto del 2021.

Firmado electrónicamente por:
ERIK DAVID SOLIS

Erik Solís
CI: 1004037444

Dedication

"This thesis is dedicated to my mother, brother, and father who has always been a constant source of support and inspiration during the whole university phase. This work is also dedicated to professors who shared their knowledge and encourage students to introduce to the research field. Also to friends and classmates with whom it was possible to study, develop projects and share moments that marked the university phase."

Acknowledgments

I would like to express special thanks to my thesis advisor Erick Cuenca for his guidance and support in completing this project. I would also like to extend my gratitude to my previous thesis advisor Lorena Guachi for supporting the proposed project and encouraging the development of research skills.

Abstract

This project presents a deep learning algorithm for intraday stock prices forecasting of Amazon, Inc. Deep learning methods can identify and analyze complex patterns and interactions within the data allowing to optimize the trading process. This study focuses on deep architectures such as convolutional neural networks (CNN), long short-term memory (LSTM), and densely-connected neural networks (NN). Results have shown that the combination of these architectures performs accurately when forecasting non-stationary time series. The evaluation of the proposed method has resulted in a mean absolute error (MAE) of 6.7 for one-step-ahead forecasting and 9.94 for four-step ahead forecasting.

Keywords: Deep learning (DL); Convolutional neural networks (CNN); Long shot-term memory (LSTM); Densely-connected neural networks (NN); Mean absolute error (MAE).

Resumen

Este proyecto presenta un algoritmo de aprendizaje profundo para el pronóstico de los precios de acciones de Amazon, Inc. Los métodos de aprendizaje profundo son capaces de identificar y analizar patrones complejos e interacciones presentes en el conjunto de datos utilizado, esto permite la optimización de los procesos de inversión. Este estudio se enfoca en el análisis de arquitecturas profundas como redes neuronales convolucionales, redes neuronales recurrentes y redes neuronales profundas. Los resultados del método propuesto han demostrado que la combinación de estas arquitecturas proveen un buen desempeño al pronosticar series de tiempo no estacionarias. La evaluación del método propuesto ha arrojado un error absoluto medio de 6.7 para predicciones de un paso por delante y un error absoluto medio de 9.94 para predicciones de cuatro pasos por delante.

Palabras Clave: Aprendizaje profundo; Redes neuronales convolucionales; Redes neuronales recurrentes; Redes neuronales profundas; Error absoluto medio.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A time series is an ordered sequence of values that are usually equally spaced over time [2]. Time series are encounter in stock prices, weather forecasts, or historical trends. Moore's law is an example of historical trends, it is empirical forecasting about the development of microchips [3]. This law describes the regularity in which the number of transistors on integrated circuits doubles approximately every two years. For the case of Moore's law, there is a single value describing each time step, so these types of series are called univariate. There are also multivariate time series, where the sequence is composed of multiple values at each time step. An example of multivariate time series is the register of births versus deaths in a period of time. Multivariate time series are useful for understanding the correlation between variables allowing to analyze the impact of the related data. For example, if the number of deaths passes to the number of births, it leads to a population decline [4].

Time series contains patterns describing different behaviors. One of these patterns is the trend, where time series have a specific direction that they are moving in. Another is seasonality, which occurs when patterns repeat at predictable intervals. An example of this is the sales of shopping sites that peak on weekends and decrease the following days [5]. Also, there are time series with a completely random behavior producing what is typically called white noise. Another type is auto-correlated time series, where the value at each time step is dependent on previous ones. Commonly, time series such as weather forecast, stock prices, or population statistics are described as a combination of trend, seasonality, auto-correlation, and noise [6].

Algorithms focused on forecasting time series are known as sequential models [7]. These models are designed to spot patterns within the data. Once the model spot these patterns, it is possible to make predictions. Traditional sequential models are based on the assumption that patterns that existed in the past will continue in the future. But this assumption can not be translated to stock price prediction since the stock market behavior is influenced by different external factors. Events such as financial crises, political events, or changes in company policies can impact drastically the behavior of financial markets. Due to the different factors affecting financial markets over time, sequences composed of stock prices are considered non-stationary time series [8].

Analyzing stock market movements has become an extremely challenging task for both investors and researchers. The complexity related to this task is based on the behavior of the stock market characterized by being non-stationary. Stock markets are affected

by interrelated factors such as economic, industry, company, psychological, and political variables. These variables interact in a very complex manner leading to the assumption that stock markets can not be predicted. In this sense, the efficient market hypothesis states that those asset prices reflect all available information at the moment. This hypothesis implies that it is impossible to predict consistently the market behavior since market prices should only react to new information [9].

However, some researchers state that markets are inefficient, in part due to the psychological variables of market participants and the inability of the markets to immediately respond to newly released information [10]. Based on this hypothesis, financial variables such as stock prices are thought to be predictable. Therefore, due to potential market inefficiencies, market participants have focused on the development of accurate forecasting strategies of financial variables.

In order to analyze the stock markets, statistical and machine learning methods have been explored. Statistical approaches often employ autoregressive moving average (ARMA) [11], autoregressive integrated moving average (ARIMA) [12] or linear discriminant analysis (LDA) [13]. On the contrary, one of the most common machine learning techniques used to forecast financial variables has been artificial neural networks (ANNs). Conventional ANNs were mostly used in stock market prediction in the latter part of the last century [14]. The following trend of machine learning application on financial markets focused on applying Multilayer perceptron (MLP) [15]. For instance, a comprehensive review of these studies, both statistical and machine learning, can be found in Atsalakis Valavanis (2009) [16]. Some of the studies put in evidence the drawbacks of ANN when dealing with sequential data. The incapability of ANNs for dealing with sequential information is based on the assumption that all units of the input vectors are independent of each other preventing to capture the relationship between earlier and later values of a sequence. However, deep architectures can overcome these problems. In this sense, sequential models such as recurrent neural networks (RNNs) have transformed speech recognition, natural language processing, and other areas focused on the analysis of time series [17].

Regarding the application of deep architectures on financial time series, deep neural network (DNN) approaches have improved the limitations of ANNs yielding good results for developing High-Frequency trading strategies [18]. Deep learning frameworks based on deep architectures such as autoencoders and long-short term memory (LSTM) have been combined for stock price forecasting [19]. Other approaches focused on combining deep architectures such as RNNs for dealing with sequential data and convolutional neural networks (CNNs) for identifying features within the data such as interdependencies among the companies to understand the market dynamics [20]. Since financial markets are constantly affected by events such as political decisions or changes in company policies, deep architectures for stock market prediction from financial news articles based on RNNs and CNNs have been developed [21]. In order to face the characteristic of non-stationary time series, decomposition methods such as empirical mode decomposition (EMD) and complete ensemble empirical mode decomposition with adaptive noise (CEEMDAN) have been combined with LSTMs increasing the accuracy of time series forecasting [22].

1.1 Problem statement

In this study, the capability of deep architectures for forecasting non-stationary time series composed of stock prices is analyzed. A deep learning model composed of deep architecture such as CNNs, LSTMs, and densely-connected neural networks is evaluated to forecast intraday stock prices of Amazon Inc (AMZN). This model uses as input a batch dataset composed of sequences with a period of time long enough to capture a high diversity in price movements, the sequence construction is based on a sliding window approach. The evaluation of this model consists of one-step and multi-step ahead forecasting.

1.2 Objectives

This research builds on the growing demand for more sophisticated algorithms for analyzing stock markets, and the development of deep learning algorithms for forecasting non-stationary time series.

1.2.1 General Objective

To develop a model using deep learning techniques for time series forecasting on stock market data.

1.2.2 Specific Objectives

- To consolidate a time-series stock market dataset for training, testing, and validating a deep learning model.
- To compare performance metrics such as accuracy, mean absolute error, loss, sensitivity, and specificity with different DL architectures in order to improve the model performance.
- To test different parameters involved in the training of DL models such as optimizers, learning rate, loss, and activation functions to achieve an accurate algorithm.

1.3 Contributions

The contributions of this project consist of an overall review of State-of-the-Art deep learning techniques inspired by the research of experts leading the development of artificial intelligence. This project also contributes with an approach for building datasets composed of non-stationary financial time series that can be used for training and testing sequential models. The main contribution of this project consists of a deep learning model able to perform one-step and multi-step ahead forecasting of financial time series. The results provided by the model suggest that deep learning techniques can optimize the development of trading strategies allowing to speed up the trading process.

1.4 Document Organization

In this project, deep architectures have been introduced for forecasting non-stationary time series consisting of stock prices. Chapter 2 provides an overall review of state-of-the-art deep learning techniques used on tasks such as image classification, natural language processing, or speech recognition.

Chapter 3 describes the methodology used for forecasting non-stationary time series. This section provides a detailed explanation of how the dataset is built. This section also describes the proposed deep architecture and depicts the working of the model through a block diagram. Also, the methodology used to define an optimum set of parameters for training and testing is described.

Chapter 4 describes the materials used for the development of the project, the model implementation, and the experimental setup. This section provides an analysis of the experiments performed to define an optimum window size and the number of training examples that improve the accuracy of the model. This section also compares the forecasted values with the actual values of the test dataset.

Chapter 5 is a result and discussion section which shows that deep architectures provide accurate results when forecasting non-stationary time series. Finally, chapter 6 carries the conclusion and future studies.

Chapter 2

Theoretical Framework

2.1 Artificial Intelligence and Machine Learning

In the early days of computers, people wonder if computers might become able to make inferences intelligently and not only perform automated routine labors. Nowadays, artificial intelligence (AI) is a wide field with many practical applications such as analyze and understand speech, text, or images, make diagnoses in medicine, product recommendations and support scientific research in other areas [23]. Problems that are difficult for humans beings but can be described by a list of formal mathematical rules are relatively straightforward to solve for computers. Tasks such as recognizing words or images can be performed easily for humans beings but can become difficult for computers since these tasks are difficult to describe formally. The main objective of artificial intelligence is to try to solve these intuitive problems allowing computers the ability to learn from experiences and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts [23]. Based on this approach, computers are able to build their own representation of the world with minimal need for hand-engineered operators that formally specify the knowledge acquisition method.

Since tasks like object recognition or speech require a big amount of knowledge about the world by a person, this knowledge can be acquired and interpreted in different ways and therefore difficult to express in a formal way. Different artificial intelligence projects have tried to articulate knowledge about the world in formal languages in order to allow computers to infer from these languages using logical inference rules. This approach is known as knowledge base, one of the most popular projects based on this methodology is the Cyc Project[1] [24]. This project was an inference engine and a database of statements managed by human supervisors. The main complication of the Cyc project was providing enough formal statements that describe the world. Cyc struggle on understanding a story about a person named Fred shaving in the morning. Cyc infers that people do not have electrical parts, but as Fred was using an electric razor, the inference engine concludes that the entity "FredWhileShaving" contained electrical parts. Then, Cyc asked if Fred is still a person while he was shaving [23].

Based on the difficulties of using knowledge-based approaches, artificial intelligence

[1] `https://www.cyc.com` [Last access: March 4, 2021]

methods focused on the ability of own knowledge acquisition by extracting features of raw data. This pattern recognition approach is known as machine learning (ML). Machine learning techniques allowed computers the ability to face problems involving knowledge of the real world and make inferences on tasks that appear to be subjective. The main idea in machine learning is that computers can achieve learning by inferring feasible models that explain the observed data. Then, these models can be used for making predictions on unobserved data and make rational decisions based on the predictions. Observed data can be consistent with many models, and therefore an uncertainty factor exists on which model is more suitable for a given data. Machine learning techniques such as logistic regression can determine if a cesarean is needed or by using machine learning algorithms such as Naive Bayes it is possible to separate legitimate mails from spam mails [25].

2.1.1 Feature Representation

An important objective in machine learning is to learn deep hierarchies of features for other tasks. Machine learning algorithms such as logistic regression or Naive Bayes are models built from experience, which constitutes data acquired from actual cases [26]. Therefore, simple algorithms like logistic regression and Naive Bayes are highly dependent on the representation of the data they are given to achieve good performance. For instance, in the case of logistic regression for recommending cesarean, the inputs for the machine learning algorithm consist of relevant information such as the presence or absence of a uterine scar. Therefore, each piece of information that represents a patient is known as a feature. Then, logistic regression learns to correlates the different features to classify and return a prediction.

In order to achieve good performance from different artificial intelligence algorithms, the choice of features or data representation on which they are applied is important. Therefore, much of the effort when developing a machine learning algorithm is focused on designing processes of data transformation that result in a different representation of the data that can be supported by a machine learning algorithm [26].

However, this process of hand-engineered data representations is labor-intensive and implies a disadvantage for learning algorithms. One solution for this problem is representation learning. This approach allows artificial intelligence algorithms the possibility to not only discover the representation from inputs to outputs but also to extract and organize the discriminative information from the data to build its own representations. Learned representations can achieve much better results than hand-engineered representations. By reducing the dependence on feature engineering, artificial intelligence applications can be developed faster and also adapt to new tasks, and therefore representation learning algorithms are able to discover the right set of features for a novel application in minutes or days for a complex application.

The main objective when developing algorithms for learning features is to separate the factor of variation that explains the observed data. These factors refer to separate sources of influence present in the data, and they may exist as either observed objects or unobserved forces in the physical world that affect observable quantities. They are called factors of variation because they represent different aspects of the data that can vary separately and often independently. For example, when analyzing an image there are multiple factors of variation, such as position, orientation, or lighting of an object.

Most of the data present on real-world applications of artificial intelligence are influenced by factors of variation, like the color of an object in an image influenced by the daylight, or the accent of a speaker when analyzing speech recordings. All of these factors make it difficult to extract relevant features from raw data [26]. Therefore, in order to solve this problem, deep architectures introduce an approach for building data representations in terms of simpler representations. As a result, an artificial intelligence algorithm can combine simpler concepts such as the edges, blurring, or sharpening of an image in order to represent complex concepts such as the silhouette of a person [27].

2.1.2 Deep Architectures for Artificial Intelligence

In order to learn complicated functions that can represent high-level abstractions such as vision or language, deep architectures are needed. An example of a deep architecture is the multilayer perceptron (MLP). A multilayer perceptron is a mathematical function that maps a set of input values to output values, this function is built by composing multiple layers of simpler functions where each function provides a new representation of the input. The main idea of deep architectures is that depth enables AI algorithms to learn a sequence of instructions allowing that later instructions can refer back to the results of earlier instructions [23].

Fig 2.1 shows the process of an artificial intelligence algorithm interpreting an image. This process is composed of multiple modules in charge of processing the information, starting with the raw pixels and ending in a linear classifier. The intermediate modules of the algorithm consist on extracting the low-level features that are invariant to small geometric variations such as crops or rotations, transforming the features until gradually become invariant to contrast changes and contrast inversions, and then detect the most frequent patterns. A common approach for extracting useful features from natural images is transforming the raw pixel representations into more abstract representations, starting from the presence of edges, detection of more complex but local shapes, until the identification of abstract categories representing objects in the image, and then integrating all these representations to capture enough understanding of the scene to answer questions about it [27].

According to this approach of deep architecture, not all the information present in different level representations necessarily encodes factors of variation that explain the input. If an artificial intelligence algorithm can capture the factors that explain the statistical variation of the data and the interaction between factors of variation that generates the observed data, then it is possible to state that the AI algorithm understands those aspects of the world represented by the factors of variation. But most factors of variation present in data such as natural images are difficult to understand analytically due to the lack of formalized prior knowledge about the world that explains the possible image variations, even for a simple abstraction as a dog standing from Fig 2.1. This high-level abstraction of a standing dog can be seen as a feature or even a category, and therefore all of the factors included in the representation of a feature can be used to design a category detector. Therefore, the objective of deep architecture learning is to automatically discover such abstractions starting with low-level features such as pixels to the highest level concepts such as objects in an image.

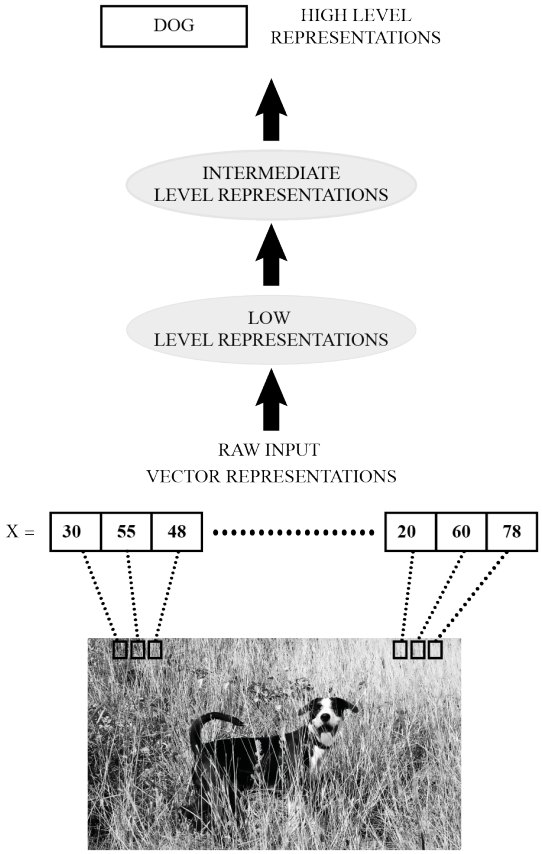


Figure 2.1: Hierarchy feature extraction process of a Deep Neural Network.

2.1.3 Deep Architecture Representation

Deep architectures achieve learning by building hierarchies of features, higher feature hierarchies are formed by the composition of lower-level features. This architecture of multiple levels of abstract representations allows the learning of complex functions that maps the inputs to outputs from the raw data and avoid the complete need for hand-engineered features. The ability of deep architectures to extract features by themselves will improve as the amount of data to be analyzed increases [17].

Depth of architectures refers to the number of levels of compositions of non-linear operations in the learned function [27]. There are two main approaches for measuring the depth of an architecture. The first approach is a computational graph based on the number of sequential instructions that must be executed to evaluate the architecture, it can be seen as the longest path of the graph from its inputs to outputs. The second approach is a probabilistic modeling graph that measures the depth based on how concepts are related to each other. This is because lower levels of hierarchies can be refined based on information about higher-level hierarchies representing more complex features. For example, an AI algorithm analyzing the image of a face with one eye on the shadow may only see one at the beginning, but when detecting the presence of a face, the algorithm can infer that a second eye is probably present [23]. Based on this approach, the graph can be built based on two concepts, one layer for eyes and another layer for faces Figure 2.2.

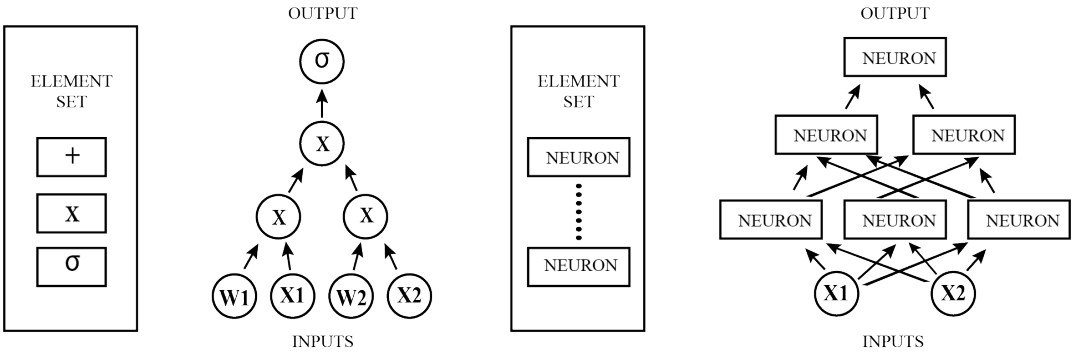


Figure 2.2: Illustration of two different approaches for measuring the depth of an architecture. A deep architecture can be represented through a computational graph mapping an input to an output where each node performs an operation. The figure to the left represents an approach where each node performs an operation such as addition, multiplication, or sigmoid function. To the left, an approach where each node corresponds to a neuron. Both approaches consisting of different element sets will result in the measure of different depths.

Figure 2.2 shows a diagram of a deep architecture representing two approaches based on graphs. Left, the first approach defines depth as the length of the longest path from inputs to outputs and consider which can be defined as a computational step. In this case, the computation considered is logistic regression, the operations used are addition, multiplication, and logistic sigmoids. Based on these three operations as the elements describing the computer language as $x * \sigma(a * x + b)$, the computed depth is equal to 3. Right, if the elements considered to measure the depth of the model are artificial neurons describing $f(x) = \sigma(b + wx)$. Then, each set of neurons has a different set of parameters (w, b), and therefore the result is a multi-layer neural network where the depth corresponds to the number of layers in the neural network, in this case, depth is equal to 2 [27].

Inspired by the depth architecture of the human brain, many attempts at training deep neural networks with the objective of achieving learning have yielded poor results. One of the first successful results takes place in 2006 with the introduction of Deep Belief Networks (DBN) [28]. Deep Belief Networks is a learning algorithm that greedily trains one layer at a time, exploiting an unsupervised learning algorithm for each layer. After that, the results of learning algorithms improved with the introduction of autoencoders. This approach exploits the principle of guiding the training of intermediate levels of representations using unsupervised learning, which can be performed locally at each level.

Different techniques for training deep architectures have been applied successfully. These improvements have allowed the development of AI algorithms for dimensionality reduction, object segmentation, or natural language processing. An advantage of autoencoders is the ability to training models with unlabeled data. However, autoencoders have been used to initialize supervised neural networks achieving successful results in tasks such as features extraction or dimensionality reductions [29] [30].

The success of deep architectures such as restricted Boltzmann machines (RBMs) and autoencoders have been due to their ability for learning data representations in an unsupervised way. The ability of these algorithms for transforming one representation into another allowed to improve the understanding of factors of variation in the data, and therefore improve the identification of representations at different level abstractions that can be used to initialize and train deep neural network based on supervised gradient descent optimization. Similarly to the brain, where different levels of abstraction consisting of neural excitation of a reduced amount of a large number of features that are not mutually exclusive [27]. These features form a distributed representation where the information is not localized in a particular neuron but distributed across multiple neurons. [31]. Based on the characteristic of features being distributed, it appears that the brain uses a sparse representation. It means that only 1 to 4 percent of the number of neurons are activated for a given activity [32].

Beyond the biological perspective inspiring initial AI approaches. The modern deep learning perspective goes beyond a neuroscientific perspective and focuses on a more general principle of learning multiple levels of composition that can be applied to not necessarily neural-inspired models. The main reason for leaving behind the notion of deep architectures based on neuroscience is that currently there is not enough information about the brain to be used as a guide. However, the idea of allowing the interaction of multiple computation units to become intelligent was inspired by the brain. Modern deep learning approaches are inspired in different areas such as math, linear algebra, statistics, or numerical optimization.

2.1.4 Multi-Layer Neural Networks

A multi-layer neural network can be represented as multiple series of layers, where each layer combines an affine operation and a non-linearity. There are three types of layers that can be differentiated in a multi-layer neural network architecture, the input layer x, the hidden layers h^k, and the output layer h^l [23]. Figure 2.3 shows the notation used for representing multi-layer neural networks. Each layer in the network represents deterministic transformations computed in a feedforward way starting from the input layer, through the hidden layers, until the output layer in charge of comparing the resulting value with a given label y to measure the error $L(h^l, y)$ to be minimized.

Multi-layer neural network architectures are described by equation (2.1). Where h^k is the output vector computed from the previous layers h^{k-1} with parameters b^k representing a vector of offsets and W^k as a matrix of weights. Then, an activation function such as $tanh$, $Sigmoid$ or $ReLU$ is applied element-wise in order to achieve non-linearity. The output layer h^l is used for making predictions and is combined with a supervised target y into a loss function $L(h^l, y)$.

In order to return probabilistic distributions used on classification tasks, the output layer can have a non-linearity different from the one used in previous layers, for example, the softmax function (2.2). The output of the softmax function h_i^l can be used as an estimator of $P(Y = i|x)$, where Y is the class associated with a given input data x. A common loss function used in multi-layer neural networks is the log-likelihood (2.3) whose expected value over (x,y) pairs is to be minimized [27].

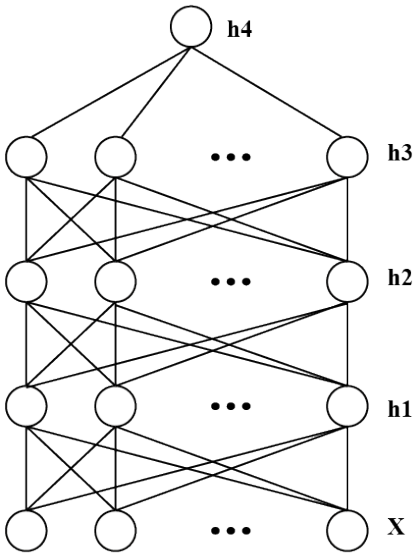


Figure 2.3: Notation used for representing multi-layer neural networks.

$$h^k = tanh(b^k + W^k h^{k-1}) \tag{2.1}$$

$$h_i^l = \frac{e^{b_i^l + W_i^l h^{l-1}}}{\sum_j e^{b_j^l + W_j^l h^{l-1}}} \tag{2.2}$$

$$L(h^l, y) = -logP(Y = y|x) = -logh_y^l \tag{2.3}$$

Training Deep Neural Networks

Until 2006, many attempts for training deep neural networks (DNN) reported negative results. Some reports suggest that training of deep supervised multi-layer neural networks and random initialization of weights gets stuck in "apparent local minima or plateaus" [33] and this drawback increase as the model gets deeper avoiding achieving good generalization of factors of variation. Some observations suggest that random initialization of weights yields poor results when treating with deep neural networks [34]. Then, some reports found out that pre-training each layer with an unsupervised learning algorithm results in much better results.

These approaches applied RBMs and autoencoders for training each layer based on the idea of greedy layer-wise unsupervised learning where the first layer is trained with an unsupervised learning algorithm, then the output of the first layers is used as the input of the following layer [33] [35]. After initializing the first layer with unsupervised learning algorithms, the neural network is trained using supervised learning algorithms. Other experiments report achieving a training classification error down to zero without applying

unsupervised pre-training even with deep neural networks [34]. Therefore, the problem pointed in two different directions, a regularization problem for unsupervised pre-training or an optimization problem for non-unsupervised training.

In the experiment where the error down to zero, the hypothesis suggested that when the top hidden layer is unconstrained, the top two layers are enough to fit the training set even if lower layers provide a poor input representation. Also, if there are enough hidden units in the top hidden layer, the training error can be reduced very low even if the lower layer is not properly training, but this can yield on a not proper generalization affecting the predictions in unobserved data. These experiments show that unsupervised pre-training is most marked for the lower layers and non-unsupervised pre-training by the top layers. One way to avoid the differentiation between optimization and regularization is to consider a truly online setting in which data becomes available in sequential order and is used to update the best predictor for future data at each step. Then, gradient descent can perform a stochastic optimization of the generalization error [27].

2.1.5 Increasing Accuracy of Deep Learning Architectures

A key factor for the success of deep learning has been the development of big datasets required for its training. Big datasets have expanded remarkably over time, and the main reason is the growing digitization of society. The increasing amount of task performed on computers make possible to gathers large amounts of information. As computers work as an interconnected network, it is possible to centralize the collected information and transform it into a dataset appropriate for deep learning applications. The process of generalizing a large amount of data based on the observation of a small amount of data is known as Big Data. In this sense, the requirements for training a deep learning algorithm decrease as the amount of training data increases, allowing to provide the resources needed by a deep learning algorithm to succeed.

One of the most widely used datasets in deep learning is the MNIST dataset [2]. The MNIST dataset is a database of handwritten digits containing 60,000 training examples and 10,000 test examples. The digits have been size-normalized and centered in a fixed-size image of 28x28 pixels. Recently, significantly larger datasets containing hundreds of thousands to tens of millions of examples have enabled the development of more sophisticated deep learning algorithms. These datasets include the public Street View House Numbers dataset [3], different versions of ImageNet dataset [4] or the WMT 2014 English to French dataset [5].

Another factor involved in the success of deep learning is the constant development of hardware. The availability of having faster computers with larger memory has made it possible the analysis of large datasets. Since the introduction of hidden layers in deep architectures, hardware advances have allowed artificial neural networks to double in size approximately every 2.4 years [23]. The availability of faster CPUs and GPUs make possible the development of faster network connectivity and better software infrastructure providing

[2] `http://yann.lecun.com/exdb/mnist/` [Last access: March 8, 2021]

[3] `http://ufldl.stanford.edu/housenumbers/`

[4] `http://www.image-net.org`

[5] `http://www.statmt.org/wmt14/`

allows to deep learning face more complex task such as medical image analysis, visual art processing or financial fraud detection [36] [37] [38].

Since the earliest deep models used to recognize individual objects in extremely small images [31], deep models have consistently improved their accuracy in tasks such as recognition and predictions. The ability of deep models for processing more complex data has scaled to object recognition networks able to process high-resolution images without the need of affecting the image resolution. An example of this is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where convolutional network approaches (CNN) have been able to beat state-of-the-art algorithms based on feature engineering with an error rate of 26.2 percent to an error rate of 15.3 percent provided by convolution neural networks. This deep architecture consists of 60 million parameters and 650.000 neurons, five convolutional layers, and three fully-connected layers with a final 1000-way softmax [39]. Since then, CNNs has consistently have won the challenge, the last winner of the competition resulted in an error rate of 2.251 percent. This architecture called SENet is constructed with a "Squeeze-and-Excitation" (SE) block that adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels [40] Figure 2.4.

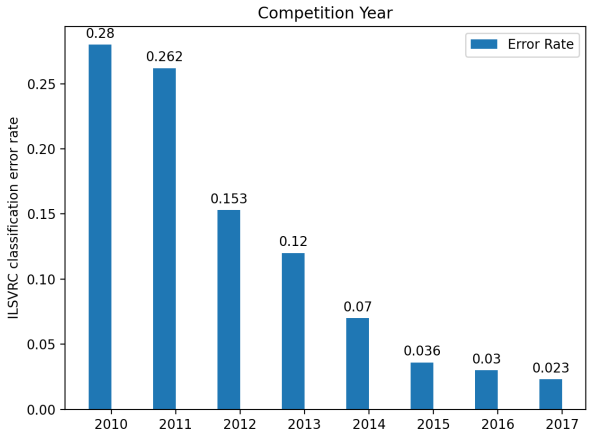


Figure 2.4: Error rate winners of ILSVRC.

The introduction of deep learning on speech recognition task yields a reduction of the error rate, in some cases, the error was reduced in a half [41]. Deep architectures also have succeeded in image segmentation resulting in a spectacular performance in traffic sign classification [42]. At the same time that the accuracy of deep learning algorithms increased, the ability of these models for solving more complex tasks also extended. Researches showed that neural networks could learn to output an entire sequence of characters translated from an image instead of just identifying a single object [43]. In order to achieve this type of learning, researchers thought that each individual element of the sequence must be labeled [44]. Then, sequence-to-sequence learning algorithms such as long short-term memory (LSTM) sequence models are used to analyze the relation between different sequences rather than just fixed inputs. This type of recurrent neural networks (RNN) has yielded on the introduction of deep learning on tasks such as machine translation [45].

The success of deep learning has also depended on the development of software infrastructures supporting the implementation of deep algorithms. Frameworks and libraries

such as TensorFlow [6], PyTorch [7], Caffe [8] or Theano [9] have supported the development of many research project or commercial products.

Deep learning has consolidated as one of the most promising branches of machine learning, based on areas such as statistics, applied math, and inspired on the human brain. Constant advancements of computer hardware and software, larger datasets, and techniques to train deeper networks have allowed deep learning techniques to expand over different research areas resulting in good results due to the accuracy of classification and prediction tasks.

2.2 Deep Learning

Deep learning technology is used in a wide variety of aspects of modern society. Object detection on images [46], speech recognition [47] or natural language processing [48](change cite NLP) are different applications of deep learning. Deep learning is a representation-learning method with multiple levels of representations obtained by composing non-linear modules that each transform the representation starting with the raw data into a representation at a higher level. By achieving enough such transformations, very complex functions can be learned [17]. In the case of classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations. For example, when classifying images, the firsts layers are in charge of discovering the presence or absence of edges, blurring, or sharpening. Subsequent layers will be responsible for assembling variations on edge positions to detect characteristics of objects in the image. Then, the last layer will classify the image based on previously analyzed characteristics and the labels assigned to each image. The idea of deep learning is to provide supervised data to the model in order to learn characteristics avoiding external human intervention.

Different types of data can be analyzed through deep learning techniques by extracting relevant features from the data. Considering different characteristics of the input data, several types of deep learning architectures. For example, recurrent neural networks (RNNs) for processing sequential data or convolutional neural networks (CNNs) for analyzing images. When analyzing sequential data, such as speech or language, RNN architectures can maintain in their hidden units information about the history of all the past elements of the sequence. Therefore, recurrent neural networks can predict the following word in a text or forecasting the next value of a stock price. More complex tasks can be approached by using RNNs such as an encoder-decoder of a sentence from one language to another [49].

2.2.1 Supervised Learning

One of the most common approaches for machine learning is supervised learning. This method is based on a set of inputs and outputs, inputs often called the predictors or

[6] `https://www.tensorflow.org`
[7] `https://pytorch.org`
[8] `https://caffe.berkeleyvision.org`
[9] `https://github.com/Theano/Theano`

independent variables, and outputs called responses or dependent variables. Outputs can be qualitative or quantitative, for example, categories such as iris discrimination [50] and glucose level [11]. In the case of Iris discrimination the output is qualitative D={Virginica, Setosa, and Versicolor} and in the case of glucose level example is quantitative D={Glucose Level, Temp Basal, Basis Heart Rate}. There is no explicit ordering in the output classes, and commonly descriptive labels are denoted as numbers to represent a category. Inputs also can be distinguished as qualitative and quantitative. And different methods can be used to make predictions using qualitative or quantitative inputs. In the case of qualitative variables, two classes such as "sarcasm" or "non-sarcasm" can be represented in a binary digit 0 or 1 [51], this numeric representation also known as targets.

A machine learning method can process inputs and produce an output in the form of a vector of scores representing a category. For example, given atmospheric data about previous days, it is possible to forecast the ozone level of the following days [52]. Or based on the pixel values of a handwritten digit [53], it is possible to predict its numeric label. During training, the machine learning method is shown an input, and its objective is assigned to the desired category the highest score of all categories. Therefore, an objective function that measures the error between the outputs and the labels modifies the internal parameters to minimize the error. These parameters called weights and biases has a similar function to a knob that defines the outputs of the model. In order to decrease the error between the computed output and the label, the machine learning algorithm computes a gradient vector that indicates by what amount the error would increase or decrease if the weight is increased minimally. Then, the weight is modified to the opposite direction of the gradient vector.

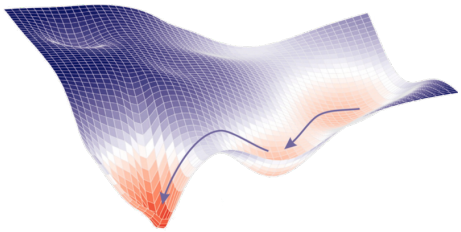


Figure 2.5: Representation graphic of gradient descent optimization, image is taken from Primo.ai [1]. Gradient descent is an optimization algorithm used to find the minimum value of a function by repeated steps moving in the direction of steepest descent as defined by the negative of the gradient.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the most commonly used algorithm to find the values that minimize the error of a given function. SGD methodology is based on taking a small set of examples and compute the output error. Then, the average gradient for those examples is computed and the corresponding weight is adjusted. This process is repeated for all the small sets until the mean of the objective function stops decreasing [17]. The main advantage of SGD is its quick ability to find values for the weights that minimize the error. Figure 2.5 shows a representation of how gradient descent is computed.

Cost Function

The cost function is a widely used method in supervised learning to measure the error between predicted outputs and the ground truth outcomes. Supervised machine learning aims to minimize the overall cost, thus optimizing the cost function through SGD is the way that machine learning algorithms are able to learn.

2.2.2 Deep Neural Networks

A deep neural network (DNN) architecture is a multilayer set of neuron-like unit sets that computes non-linear outputs. Each unit in a layer makes use of an activation function such as Sigmoid or rectifier linear unit (ReLU) to increase the selectivity and the invariance of the representation. Allowing an architecture to have multiple non-linear layers, a model can implement extremely intricate functions of its inputs that are simultaneously sensitive to minute details and insensitive to large irrelevant variations such as the background, pose, lighting, and surrounding objects. DNNs have many differences from traditional approaches for classification. Its deep architecture provides more expressivity and robustness for learning data representations without the need for hand-engineered features. An empirical demonstration of this is the ImageNet classification task across thousands of classes [39]. A standard DNN is composed of multiple layers, where the first layer often called as input layer is the one in charge of feeding the model with external data, the last layer often called as output layer returns the predicted values from the model, and the intermediate layer often called hidden layers are in charge of transforming the inputs to a non-linear representation so that the dataset categories become linearly separable by the output layer.

Backpropagation

The backpropagation procedure is based on repeated value adjustment of the weights in a network to minimize a measure of the difference between the forecast output and the desired output. As a result of the weight adjustment, deep learning models can represent important features of a dataset. The principal advantage of backpropagation is its capability to create useful new features.

The backpropagation methodology for computing the gradient of an objective function with respect to the weights of a multilayer neural network is the application of the chain rule for derivatives. The main idea is that the derivative of a function with respect to the input of a layer can be computed backward from the gradient with respect to the output of that layer. This process is repeated in order to propagate the gradients through all the modules starting from the output layer to the input layer where the model is fed. After computing these gradients, the gradients with respect to the weights are straightforward to compute [17].

Convolutional Neural Networks

Convolutional neural networks (CNNs) are designed to process data represented on multiple arrays such as images, audio recordings, or videos. In the traditional model of pattern recognition, a hand-designed feature extractor gathers relevant information from the input

and then the gathered features are classified into desired classes. Feature extractors often called filters or kernels can extract different characteristics of an array such as borders and edges in the case of images. These feature extractors commonly are represented as matrices. Therefore, it is possible to eliminate the hand-designed feature extractor and allow the neural network to build an accurate feature extractor based on the raw data.

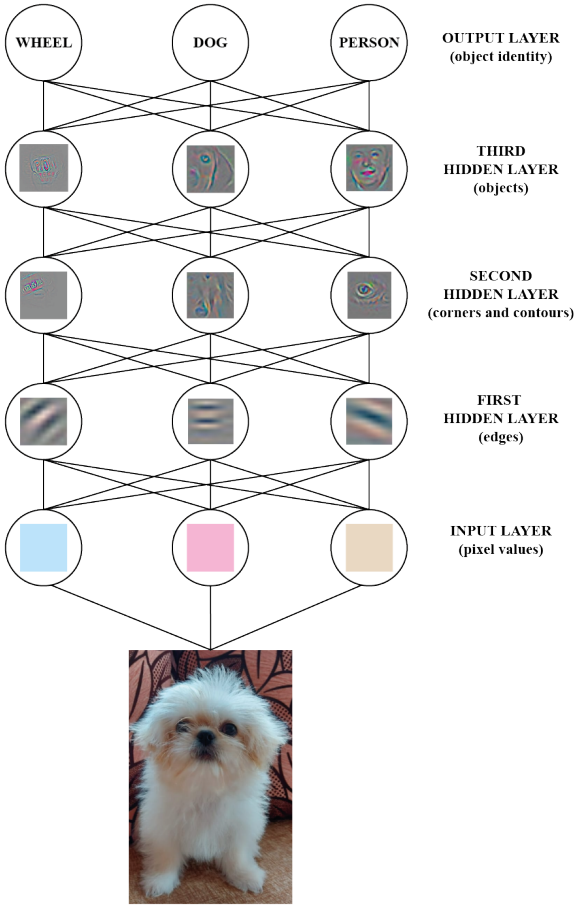


Figure 2.6: Internals of a convolutional network. This figure shows the outputs of each layer (horizontally) of a typical convolutional network architecture applied to the image of a dog. Each rectangular image is a feature map, representing different features extracted through convolutions such as edges or corners.

The main characteristic of CNNs is local connections, shared weights, pooling layers and the use of many hidden layers. Commonly CNN architecture is composed of different stages, starting with convolution and pooling layers. Units in a convolutional layer are organized in feature maps, where each unit is connected to local patches in the feature maps of the previous layer through a set of weights called filter bank [17]. Then, an activation function is applied to the resulting values. All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks. This architecture is based on two facts. First, multiple arrays such as images have local groups

of values often highly correlated, this involves local patterns that can be easily detected. Second, different local patterns can appear independently from the location in the image. Therefore, by having units at different locations which share the same weights are able to detect different pattern on different array location.

Then, a pooling layer reduces the dimension of the extracted features creating an invariance to shifts and distortions on the image. This process can be repeated by stacking convolutions layers, non-linearity, and pooling layers followed by fully connected layers to create feature extraction filters that represent low-level characteristics like an edge that can be assembled into parts that compose an object. This object detection process also can be applied to speech and text from phonemes or letters that can be assembled into syllables, words, and sentences. Figure 2.6 shows a representation of how deep architecture performs feature extraction.

2.2.3 Recurrent Neural Networks and Long Short-Term Memory

The recurrent neural network (RNN) is a type of deep architecture that has a deep structure in the temporal dimension. RNNs are widely used in time series modeling. In this sense, the traditional architecture of ANNs where all units of the input vectors are independent of each other avoids the possibility of handling sequential data. On the other hand, RNNs architectures add a hidden state that is generated by the sequential information of a time series. This characteristic of RNNs allows that the resulting output has a dependence on the hidden state, and therefore on the time series as shown in Fig 2.7.

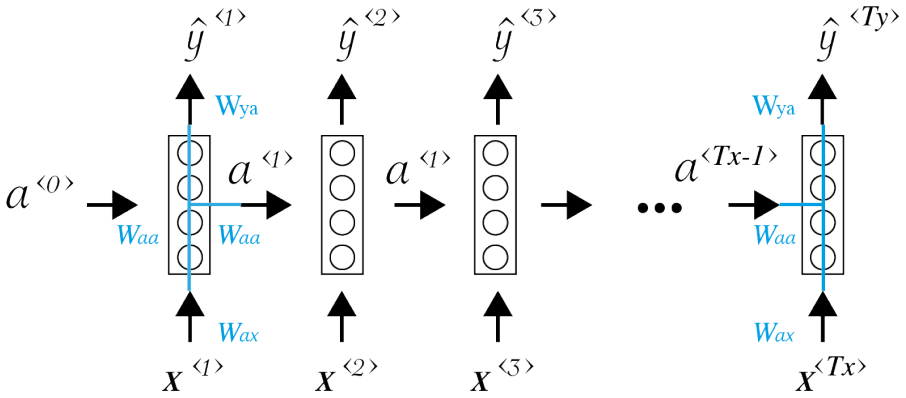


Figure 2.7: Unfolded RNN diagram.

With the introduction of backpropagation, one of the most interesting applications was focused on RNNs. RNNs process an input sequence one element at a time, allowing them to maintain in their hidden units a state vector containing implicit information about the history of the previous elements of the sequence. Therefore, it is possible to apply backpropagation by considering the outputs of the hidden units at different discrete time steps in the same way as the outputs of different neurons in a multi-layer neural network [17]. A drawback of training RNNs is that when backpropagation is applied the gradients tend to grow or shrink at each time step, and repeating this process many times result in gradient exploding or vanishing [54]. Then, the introduction of a novel, efficient, gradient-based method called long short-term memory (LSTM) allowed the improvement of the

architecture and training of RNNs. By truncating the gradient where this does not harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete-time steps by enforcing constant error flow through constant error carousels within hidden units [55].

Due to the advancements in the architecture of RNNs, they have performed successfully of task making use of sequential data such as predicting the next character in the text [56], the next word in a sequence [57] or learning phrase representations using RNNs as an encoder-decoder [49]. In this sense of using RNNs as an encoder-decoder has allowed the development of complex task of machine translation, the use of RNNs consists of reading an English sentence one word at a time, after that, an English encoder is trained to allow that the final state vector of the hidden unit is able to transform the input sentence into a good representation of the meaning expressed on the sentence. Then, this state vector can be used as input of the hidden state of a trained French decoder which output can be a probability distribution of a French word. The word chosen from the probability distribution can be used as input for the decoder which returns another probability distribution containing the translation of the second word, this process is repeated until translate the whole phrase.

Instead of using machine translation to generates sequences of French words based on a probability distribution that depends on the English sentence. RNNs also are able to translate the meaning of an image into an English sentence. In this sense, CNNs can be used as an encoder transforming the image pixels into an activity vector in its hidden units. Then, RNNs can be used as a decoder resulting in the interpretation of the interaction of the object present in the image [58].

Although RNNs main purpose is learning long-term dependencies. However, researchers have reported difficulties at training RNNs to perform tasks in which the temporal contingencies present in the input/output sequences span long intervals [54]. In order to solve this drawback, by augmenting the network an explicit memory allows to the network remember inputs for a long time [55]. A special unit called the memory cell acts like an accumulator that helps to decide when to retain or forget the values of the memory cell. This memory cell contains a connection to itself at the next time step that has a weight of one, then it copies its own state and accumulates the external signal, but this self-connection is multiplicatively gated by another unit called forget gate that learns to decide when to clear the content of the memory. LSTMs subsequently proved to perform more effective results than conventional RNNs. LSTM is an effective solution for combating vanishing gradients by using memory cells, and its approach results in the development of complex speech recognition systems or encoders and decoders for machine translation.

2.2.4 Future of Deep Learning

Unsupervised learning algorithms such as RBMs, DBNs, or autoencoder had a huge impact on reviving the interest of deep learning, but since the success of purely supervised learning due to the constant development of hardware, software and the availability of large datasets has allowed that supervised learning algorithm domain most of the current application of machine learning. Although, it is expected that unsupervised learning algorithms become more important in the coming years since human and animal learning is primarily unsupervised [17].

Much of the progress in deep learning is expected to happen from the combination

of different deep architectures such as convolutional neural networks and recurrent neural networks. The combination of different deep learning algorithms has yielded on the outperforming of state-of-the-art approaches [41]. One of the areas where deep learning is expected to have a huge impact is natural processing. It is expected that algorithms making use of RNNs will be able to understand whole documents by developing strategies for selectively attending to one part at a time [45] [58].

The major advancements in deep learning are expected to happen from systems able to combine representation learning with complex reasoning. Since deep learning is already able to solve tasks such as speech or visual recognition, new paradigms are needed to replace rule-based manipulation of symbolic expressions by operations on large vectors [17] [59].

2.3 Discussion

Stock market prediction is usually considered as one of the most challenging issues among time series predictions due to the stock market essentially being a dynamic, nonlinear, non-stationary, non-parametric, noisy, and chaotic system [60] [61]. How to accurately predict stock price movement is still an open question due to a wide set of factors that interacts in a very complex manner affecting stock markets. These factors are economic variables, such as interest rates, exchange rates, or monetary growth rates; company-specific variables, such as changes in company policies, income statements, and dividend yields; psychological variables of investors, such as the expectation of investors and institutional inversion choices; and political variables, such as the occurrence and the release of important political events. Furthermore, the Efficient Market Hypothesis states that stock prices reflect all current information, and new information leads to unpredictable stock prices. The random walk theory introduces a hypothesis that stock prices are defined randomly concluding that stock prices cannot be accurately predicted using historical values [62].

On the other hand, some researchers state that the market behavior is inefficient due to the inability of the markets to immediately respond to newly released information alongside psychological factors of market participants [10]. Based on the premise that markets are inefficient, financial variables such as stock prices, market index values, and the prices of financial derivatives can be predicted. Therefore, by using information released to the general public it is possible to make investment decisions based on stock market forecasting technologies.

Most of the common methods for price prediction are based on fundamental and technical analysis. Fundamental analysis is the traditional approach using company parameters. Technical analysis is based on Dow Theory and uses price history for prediction [63]. While opinions differ on the efficiency of markets, many empirical studies have shown that financial markets are to some extent predictable [64] [65] [66]. In this sense, different methods for stock prediction have been compared such as statistical or artificial neural network approaches.

Based on the type of financial data to be analyzed, financial time series forecasting can be classified into two categories, univariate or multivariate analysis. In the case of univariate analysis, only the financial time series itself is used as input. In the case of multivariate analysis, the different market indicators can be used as input variables such

as the stock price, volume, volatility, or even financial news.

In the case of statistical approaches, techniques such as autoregressive integrated moving average (ARIMA) [67], the generalized autoregressive conditional heteroskedastic volatility (GARCH) [68] or the smooth transition autoregressive model (STAR) [69] have been applied, mostly for univariate time series analysis. Statistical approaches for multiple input variable often used techniques such as linear regression (LR) [70], support vector machines (SVMs) [71] or quadratic discriminant analysis (QDA) [72]. A drawback of the above-mentioned methods is that the data must meet requirements such as linearity, stationarity, and normality, or in other cases the input data must be treated before been used. On the contrary, approaches such as artificial neural networks are able to handle multivariate data with no required assumptions.

Different methods including statistical approaches, data mining classification algorithms such as nearest neighbor classification [73] and composite classifier [74] have been compared to artificial neural. Based on comparative analysis of stock market forecasting methods, the literature suggests that ANN algorithms yield better performance [16]. However, different researches suggest that ANNs are not the most accurate method for dealing with time series containing noise and complex dimensionality that extends for long periods of time [33]. In this sense, the introduction of deep architectures can overcome this problem and they have already yielded good results in tasks using sequential data such as speech recognition or machine translation. The success of deep architectures on the above-mentioned tasks leads to the idea that sequential data such as financial time series can also be predicted by using deep architectures.

Deep learning algorithms are capable of identifying hidden patterns and underlying dynamics in the data through a self-learning process based on supervised learning. In the case of financial time series, the data generated is enormous and is highly non-linear. Different from other approaches, deep learning architectures can effectively model these types of data and can return a good prediction by analyzing the interactions and hidden patterns within the data. One of the most used deep architectures for forecasting sequential data is RNNs. In this sense, researchers have applied RNNs with different approaches. Certain works consider using data from a single time series as input [75] and others focused on multivariate financial time series with the inclusion of heterogeneous market information and macroeconomic variables [76]. With the introduction of long short-term memory cell into recurrent neural networks architecture, the analysis of time-dependent data become more efficient. The capability of LSTMs for holding information in extended periods of time improved the performance of stock price predictions [75] [77].

In this study deep architectures have been introduced to build a prediction algorithm of univariate financial time series. The proposed deep architecture consists of convolutions neural networks to extract features within the data, recurrent neural networks, and long short-term memory cells to handle the long-term dependencies of time series and deep neural networks for increasing the inferring capabilities of the model. Chapter 3 describes the methodology suggested for stock predictions and the datasets used for training the model. Chapter 4 implements the methodology and describes the experimental parameters setting, and chapter 5 is a result and analysis section that shows the results of the proposed architecture over different stock market prices.

Chapter 3

Methodology

3.1 Phases of the Methodology

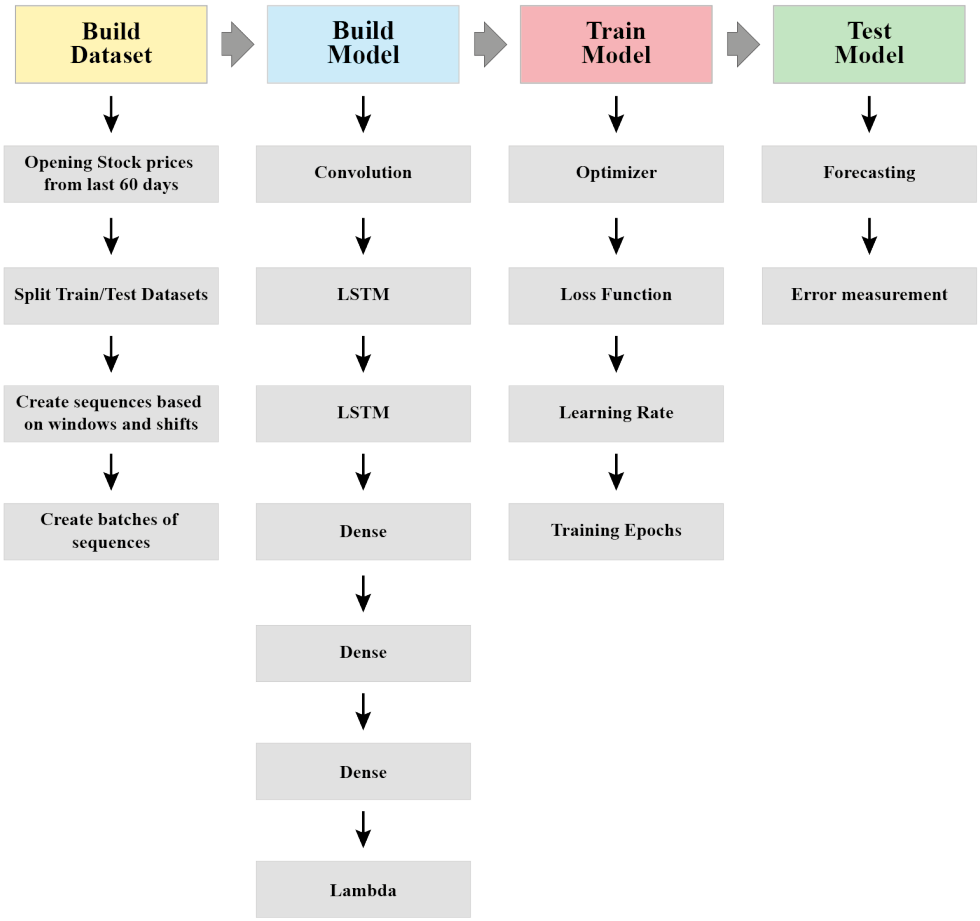


Figure 3.1: Flowchart of the proposed methodology. The following subsections explain in detail how each phase developed.

3.1.1 Build Dataset

This section covers the methodology for building a dataset composed of stock market data. This methodology focuses on overcome the drawbacks of treating with non-stationary time series.

Data description

The data used in this work involves the intraday stock prices of Amazon (ticker: AMZN). The stock prices are collected with intervals of two and five minutes during the trading hours from the last sixty days. It includes information about the volume, dividends, open, high, low, and close price. This data was obtained through yfinance [1], a library that collects historical market data from Yahoo! finance.

Univariate time series

The purpose of this study is to develop a deep learning model for forecasting univariate time series. Therefore, the series for building the dataset consists of the opening prices for two and five minutes intervals. This results in time series containing more than six thousand observations for two-minute intervals and more than three thousand observations for five-minute intervals. Figure 3.2 shows price behavior in the selected periods.

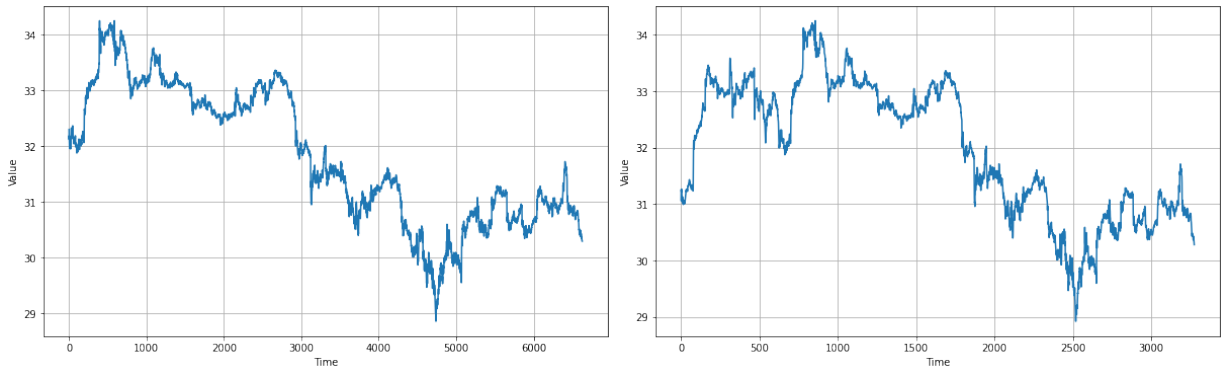


Figure 3.2: Amazon stock prices with 2 minute intervals to the left and 5 minute intervals to the right.

Split data for training and test

To train the model the data is split into two-time series, one for training and another for testing. For the data with two and five-minute intervals, 3500 and 2000 observations are selected respectively for training, and the leftover observations are used for the testing. Figure 3.3 shows the resulting time series for training and testing.

[1] https://github.com/ranaroussi/yfinance

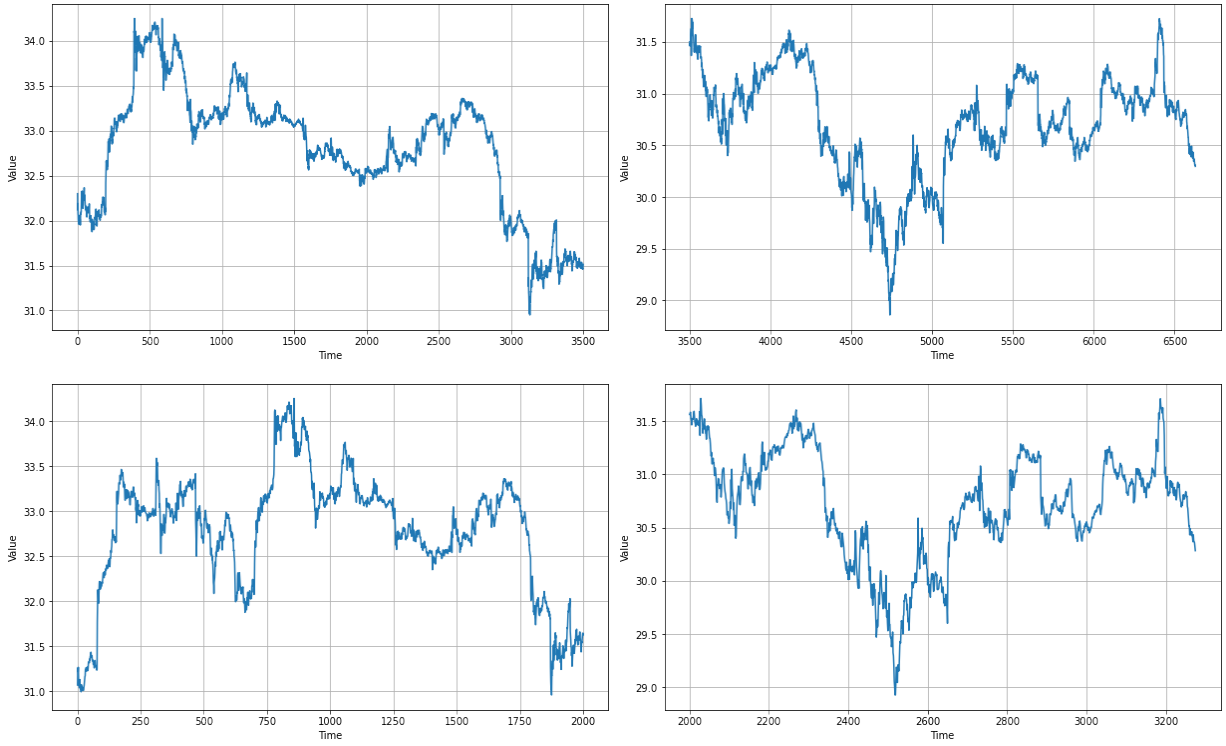


Figure 3.3: The first row of figures corresponds to the time series for training a testing with 2 minute intervals. The second row corresponds to the time series for 5 minute intervals.

Define inputs and labels for the training dataset

Our goal is to consider a period of time long enough to capture a high diversity in price movements. In the case of two-minute intervals data, a window of 60 data points corresponding to 2 hours of stock prices is selected. For five-minute intervals data, the resulting sequences consist of windows containing 108 data points corresponding to 9 hours of stock prices. This process of selecting an optimal window size is focusing on overcoming the characteristic of non-stationary time series. By selecting a limited period of time for building each sequence, the accuracy of the model increase with respect to a model processing a complete non-stationary time series.

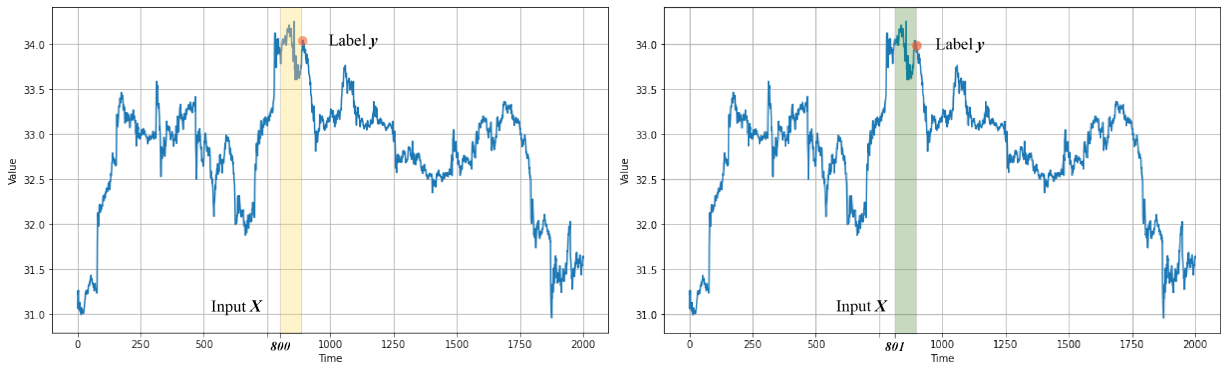


Figure 3.4: Inputs and labels selection based on sliding window approach and shifting

In order to select the inputs and labels for building the training dataset, a sliding window of size 60 or 108 is shifted every 1 value through the training time series described in Figure 3.3. The process for selecting the inputs and labels is described in Figure 3.4. This approach consists of a sliding window that is shifted every one value. In this sense, the input sequence corresponds to the values within the window, and the label corresponding to the next value of the window. The composition of an input sequence and the label corresponds to a training example.

The objective of this stage is to expand the training time series based on a time period long enough to represent the diversity in price movements. This allows the forecasting of stock prices by using a period with a similar length.

Batching the dataset

Once we have created multiple sequences based on a sliding window approach and defined the input and label for each sequence. Then, the resulting sequences are shuffled. By shuffling the data, it ensures that each sequence creates an independent change on the model meaning that each sequence is representative of the overall distribution of the data. The purpose of shuffling the data is to reduce the variance and the possibility of overfitting by allowing that the model remains general for each training example.

The shuffled dataset containing random permutations of the training sequences is then batch. Batching the data consists on merge randomly selected sequences into a batch for training. Batching the dataset allows to process of the dataset in smaller sets instead of loading all the data to the model at once. By doing this, the computer memory can be used more efficiently and allows to speed up the training time. Figure 3.5 shows a representation of batches creation. For the development of this study, the batch size was set to one hundred sequences for both datasets with intervals of two and five minutes.

3.1.2 Model Architecture

The model architecture proposed in this project consists of three types of deep learning architectures, CNNs, Dense layers, and LSTMs. Convolutional neural networks are specialized kinds of neural networks for processing data with a grid-like topology. This includes time series, which are represented with one-dimensional arrays. This type of network uses a mathematical operation called convolution, this is a specialized kind of linear operator that can be used for feature extraction.

Dense layers are regular deeply connected neural networks. This type of layer is one of the most common architectures used for developing deep learning models. The output shape of the Dense layer will be affected by the number of neurons composing the network and the defined activation functions.

RNNs are a type of deep neural network architecture that has a deep structure in the temporal dimension. In contrast, to feedforward networks, RNNs introduce a hidden state that is generated by the sequential information of a time series. RNNs consist of a hidden state and modifiable weights, RNNs focused on applying the same set of weights recursively over a graph-like structure. A recurrent neural network is defined by the following equations

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{3.1}$$

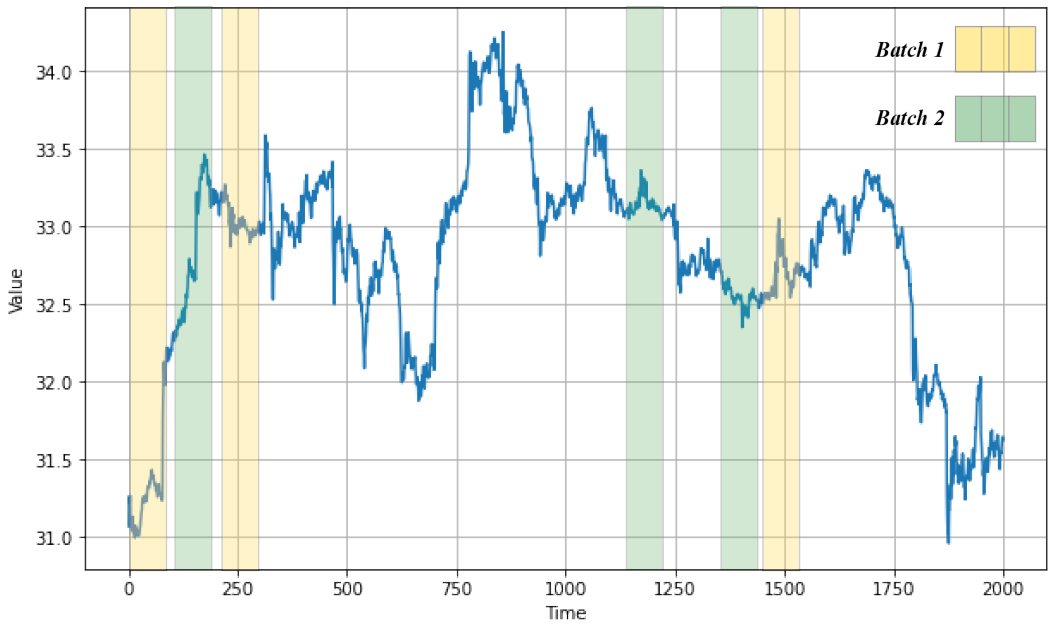


Figure 3.5: A representation of the process for shuffling and batching the dataset. This approach consists of random permutations and random selection of sequences for batch creation. In this example, the batch size is set to 3. Then, each batch is composed of 3 randomly selected sequences.

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y) \tag{3.2}$$

where the network has an input $x^{<t>}$, and an output $\hat{y}^{<t>}$. The activation value computed at a time step $a^{<t-1>}$ is passed to the next time step to compute the next activation value $a^{<t>}$. The recurrent neural network scans the data from left to right. The parameters it uses for each time step are shared, so the parameters W_{ax}, W_{ya} and W_{aa} governs the connection from the input $x^{<1>}$ to the hidden layers and from the hidden layers to the outputs $\hat{y}^{<t>}$. Then, b_a and b_y corresponds to the biases and g to the activation functions.

One weakness of RNNs is that it only uses the information that is earlier in the sequence to make a prediction. For example, when predicting $\hat{y}^{<3>}$ in Figure 2.7, the RNN does not use information about the inputs from $x^{<4>}$ to $x^{<Tx>}$. So in the sense of forecasting time series, it is important to extract information about earlier in the sequence as well as later values in the sequence. This problem could be addressed by using Bidirectional Recurrent Neural Networks (BRNNs) [78]. In the same way as RNNs, BRNNs struggle at learning long-term dependencies because of the vanishing gradient problem. Therefore, LSTMs are an effective solution for addressing vanishing gradients.

LSTM is a special kind of RNN, in this approach, the hidden layers are replaced with LSTM cells. The cells are composed of different gates that can control the input flow through the hidden layers. LSTMs consist of a forget gate, update gate, and output gate. It also consists of activation functions such as $Sigmoid$, $ReLU$, $tanh$ and element-wise multiplications Figure 3.6. LSTMs are represented with the following equations

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{3.3}$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{3.4}$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{3.5}$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{3.6}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \tag{3.7}$$

$$a^{<t>} = \Gamma_o * tanh(c^{<t>}) \tag{3.8}$$

where $c^{<t>}$ corresponds to a memory cell which provides of a memory to remember earlier input values. At every time step it is cosider update the memory cell $c^{<t>}$ with a candidate $\tilde{c}^{<t>}$ that is computed using an activation function $tanh$ of W_c plus a parameters matrix containing the previous values of the memory cell, the activation value $a^{<t-1>}$ as well as the current input value $x^{<t>}$ plus the bias b_c.

The key idea of LSTMs is the introduction of three control gates called update gate Γ_u, forget gate Γ_f and output gate Γ_o. These control gates are computed with a sigmoid function, so their resulting value will be close to zero or one. Since, the RNN is considering to update the value of $c^{<t>}$ with $\tilde{c}^{<t>}$. Depending on the values of the control gates, it is decided if the value of $c^{<t>}$ is updated. Therefore, computing the memory cell $c^{<t>}$ depends on the resulting values of Γ_u, Γ_f and Γ_o which vary between zero and one. The introduction of control gates gives the memory cell the option of keeping the old value $c^{<t-1>}$ and then adding to the memory cell the value of $\tilde{c}^{<t>}$ (3.7).

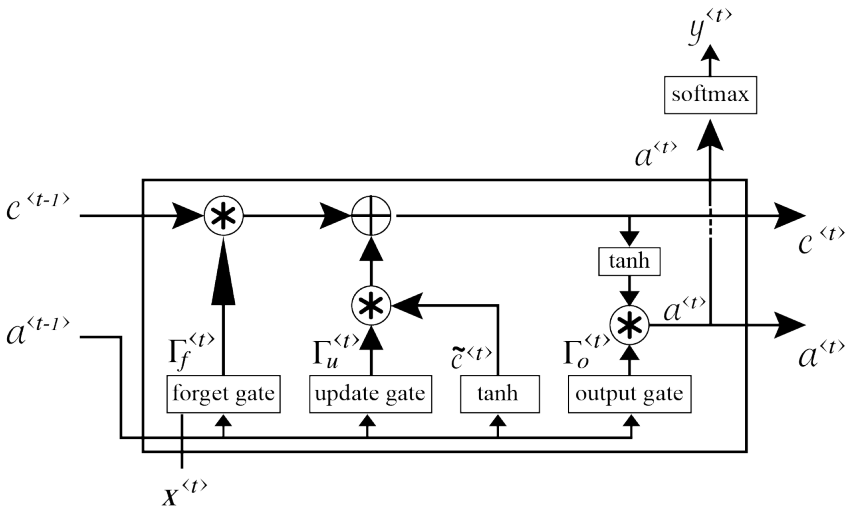


Figure 3.6: Diagram representing of a Long Short-Term Memory cell.

Model architecture

In order to develop a deep architecture for one-step and four-step stock price forecasting, this work presents a deep learning model for financial time series that integrates different deep learning approaches. Figure 3.7 shows the architecture of a 7 layer model. A batch

dataset composed of the sequences generated by a sliding window approach is presented as the input. Each batch is 1D-convolved with 60 filters, each of size 5, using a stride of 1 in the x-axis, causal padding, and ReLU activation function. The resulting feature maps are then passed through a LSTM layer where each cell is composed of 60 units and returns its output sequence. The resulting sequences serve as inputs for another LSTM layer with the same characteristics which also outputs the sequence generated by each cell.

The following layers are three regular densely-connected neural networks. The first dense layer composed of 30 units processes the outputs from the second LSTM layer. Similar operations are performed by the second layer composed of 10 units. The final dense layer is composed of 1 unit which output dimension is batch size by sequence length by dense layer units. To conclude a Lambda layer is applied to scale up the output values of the last dense layer in order to help the learning.

3.1.3 Training

The process of training the model consists of fine-tuning parameters such as the learning rate, optimizer, loss function, and the number of epochs for training. Also, this approach focused on the development of consistent data in order to decrease the training time and prevent overfitting.

Loss Function

The purpose of loss functions is to compute the quantity that a model should seek to minimize during training. The loss function used in this project is the Huber function, this function tends to be less sensitive to outliers. Therefore, the Huber loss function is able to perform well for intraday stocks characterized by having a high variance and noisy behavior.

Optimizer

Optimizers are algorithms used to modify the parameters of a neural network such as the weights and learning rate to optimize the loss function. In this model, the optimizer used is Stochastic Gradient Descent (SGD). Stochastic gradient descent is an iterative method for optimizing an objective function with suitable smoothness properties. It can be regarded as a stochastic approximation of gradient descent optimization since it replaces the actual gradient by an estimation of itself.

Defining Learning Rate

The learning rate (LR) is a tuning parameter in an optimization algorithm that determines the step size at each training iteration while moving toward a minimum of a loss function. Defining a high LR will make the learning jump over minima and a low LR will either take too long to converge or get stuck in an undesirable local minimum. Therefore, defining the correct learning rate becomes crucial for the training stage. In this sense, a learning rate scheduler is used to define the optimum learning rate for stochastic gradient descent. The learning rate scheduler is used to modulate how the learning rate of SGD changes

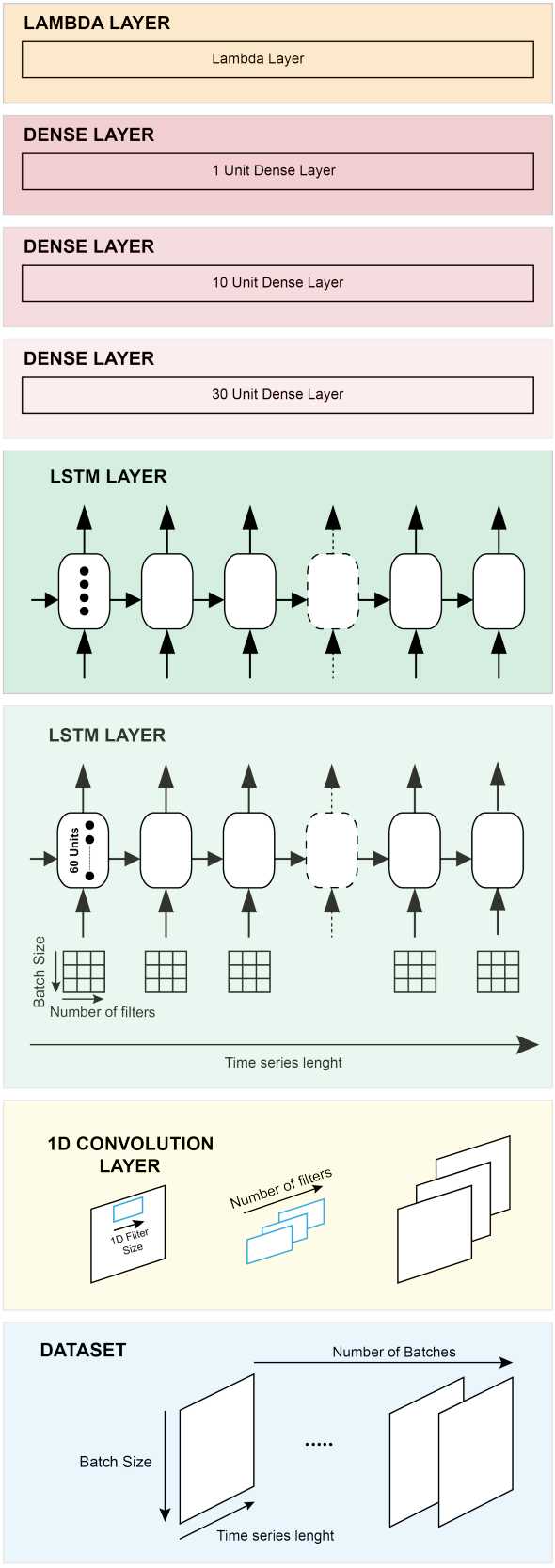


Figure 3.7: Model Architecture.

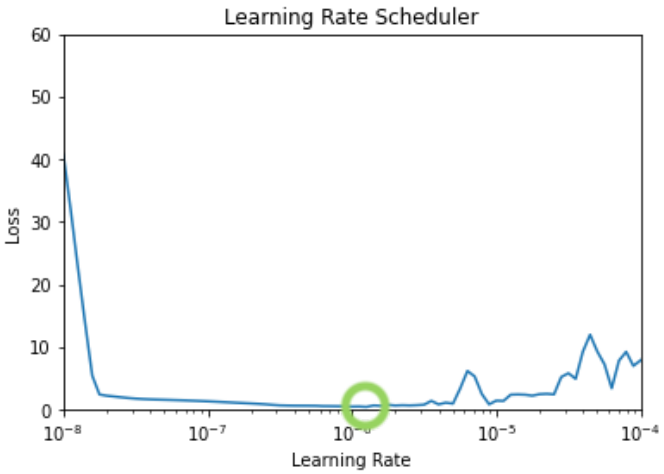


Figure 3.8: Selection of the optimum learning rate by using a learning rate scheduler.

over time, so the model is trained for a short number of epochs where the learning rate is modified to analyze the returning loss. Figure 3.8 shows the resulting plotting of the process for choosing an accurate learning rate based on using a learning rate scheduler.

Training Epochs

In order to determine the optimal number of training epochs for the model, the number of training epoch is compared with the loss values. Figure 3.9 shows the loss value corresponding to each training epoch. The figure to the left shows that the loss stays the same before 100 epochs. But the figure to the right is zoomed in to the last few epochs, where it can be observed that the trend is still downward until 400 epochs.

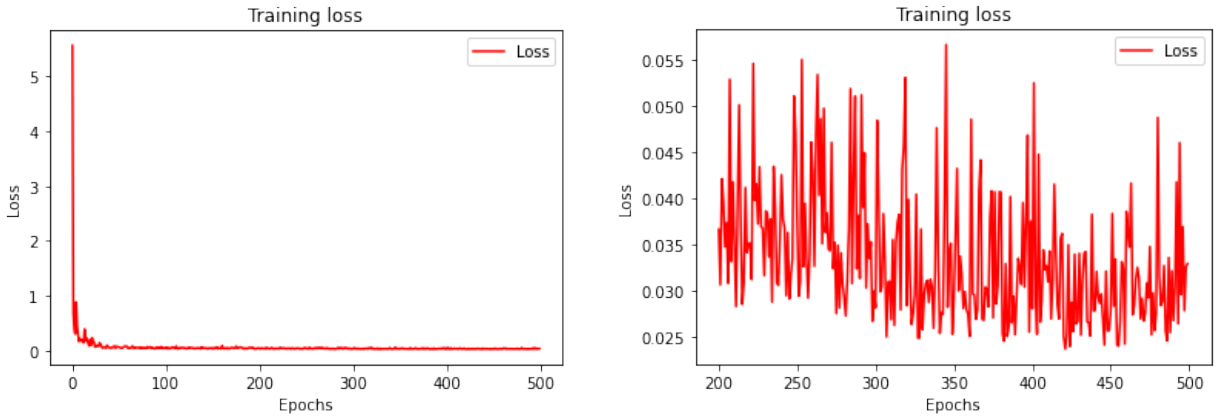


Figure 3.9: Analysis for selecting the optimum number of training epochs based on a Loss vs. Epoch plotting. The figure to the left corresponds to the loss values at each training epoch, 500 epochs. The figure to the right corresponds to the loss values of the last 300 epochs.

3.1.4 Testing

Metrics for error measure

One of the most common metrics for measuring the error between the values predicted by the model and the observed values is the mean squared error (MSE) or the root-mean-square error (RMSE). Another metric is the mean absolute error (MAE), one characteristic of this metric is not penalizing large errors as much as the MSE or RMSE does. The mean absolute error is a common measure of forecast error in time series analysis since the loss values tend to be proportional to the size of the error. The MAE is described by the following equation

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{3.9}$$

Forecast on test dataset

To evaluate the performance of the model, two types of prediction were performed over the test dataset. The first evaluation consists of one-step forecasting over the whole test data with a stride of one. And the second evaluation consists of four-step forecasting given an input sequence. Figure 3.10 shows both evaluation approaches.

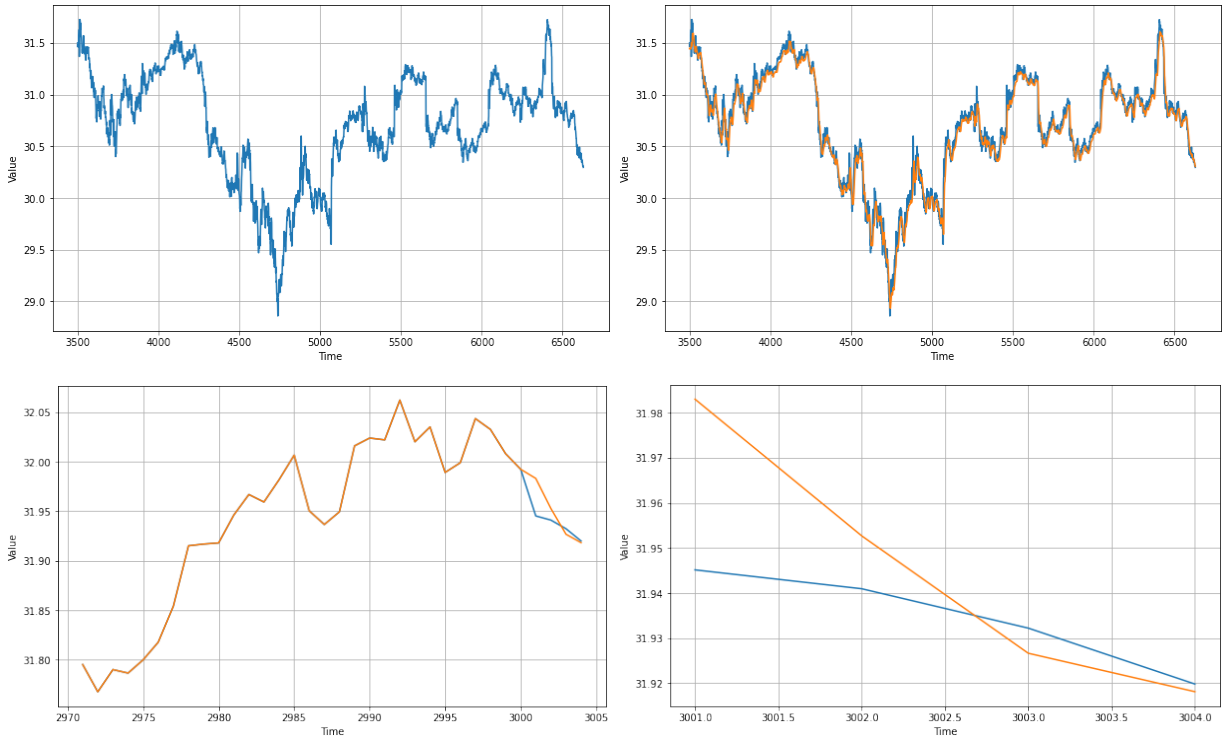


Figure 3.10: The first row of figures corresponds to one-step forecasting over the whole dataset with a stride equal to 1. And the second row corresponds to four-step forecasting given a data sequence.

Chapter 4

Experiments

The objective of the following experiments is to evaluate the performance of the model for forecasting non-stationary time series consisting of stock prices. These experiments were also used to build the proposed deep architecture and define the corresponding parameters to achieve the best performance. Figure 3.7 shows a schema of the architecture.

4.1 Materials

The implementation of this project was performed using the Keras[1], a deep learning API written in Python. This project was executed using Google Colaboratory (Colab)[2], Colab is a hosted Jupyter notebook service providing free access to computing resources including GPUs. The historical stock market prices were obtained with yfinace[3], a Python library that provides a convenient way to download historical market data from Yahoo! finance. The code developed in this project can be found in the following GitHub repository [4].

4.2 Experimental setup

This section presents the details regarding the evaluation of the model for one-step and four-step ahead stock price forecasting. The model is tested over a dataset composed of stock prices with intervals of 2 and 5 minutes. The dataset is build using a sliding window approach in order to face the drawbacks of using non-stationary time series. Different window sizes were tested for both datasets. For the 2 minute dataset, window sizes of 40, 60, and 80 were tested. And for the 5-minute dataset, window sizes of 46, 72, and 108. Testing different window sizes allow the selection of an optimum period of time capturing a high diversity in price movements. The result of this process is a set of sequences with a length equal to the window size. Then, the dataset is built by creating batches of 100 sequences. Figure 4.1 shows the experiments performed by varying the window size.

[1] `https://keras.io/about/`

[2] `https://colab.research.google.com`

[3] `https://github.com/ranaroussi/yfinance`

[4] `https://github.com/ESbros/GraduationProject.git`

Regarding the model parameters. The first layer is a CNN composed of 60 filters, a kernel size of 5, strides of 1, causal padding used for 1D convolutions, and a ReLU activation function. Then, two layers composed of LSTM cells are defined, both with the same parameters, 60 units, and enabling the possibility of returning sequences that serve as input for the next layer. The following layers consist of three regular densely-connected NN layers. The first and second dense layer implements a ReLU activation function consisting of 30 and 10 units respectively. The third dense layer composed of 1 unit corresponds to the output layer. The final layer consists of a Lambda layer which multiplies the values of the output layer by 400.

The parameters used for training the model were Huber loss function, SGD for optimizing the loss function with a momentum of 0.9, and mean absolute error for the metrics. In order to define an optimum learning rate, the model is trained for 100 epochs where at each epoch the learning rate is modified. Starting with a learning rate of $1e^{-8}$, at each epoch, the learning rate is modified as following $1e^{-8} * 10^{epoch/20}$. This process provides an approximation for selecting an optimum learning rate. Figure 4.2 shows the experiments for selecting the learning rate.

The model was trained on different datasets by varying the number of data points used for training and testing. The 'yfinance' library returns sequences containing around 6000 and 6500 data points in the case of stock prices with two-minute intervals and around 3000 and 3500 data points for five-minute intervals. The number of data points returned by the library varies depending on the day and hour that the query is performed. In this sense, the number of data points for consolidating the training set were 2500, 3500, and 4500 for two-minute intervals, and in the case of four-minute intervals, the number of data points for training was 1500, 2000, and 2500. Figure 4.3 show the model performance for different datasets varying the number of training examples.

Two forecasting approaches were performed to evaluate the model performance. These approaches consist of one-step and four-step ahead stock price forecasting. In this sense, sequences of 60 and 108 data points for two-minute and five-minute intervals are used to forecast the next data points. Additionally, a fifteen-step ahead forecasting was performed for five-minute stock price intervals. This additional experiment was only considered for two-minute interval forecasting due to the characteristics of the model developed for this interval. Figures 4.4, 4.5 and 4.6 shows the different forecasting approaches performed.

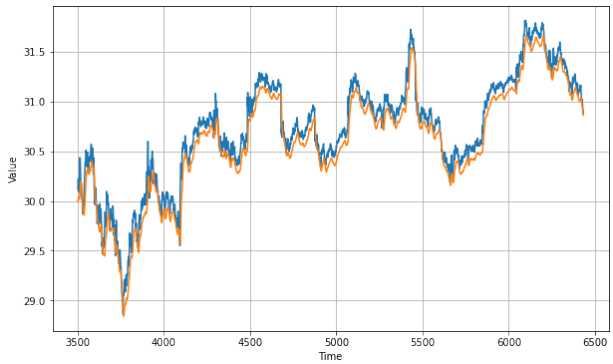
4.3 Perform Experiments

This section provides a comparative analysis of different experiments consisting of varying the window size, learning rate selection, one-set and four-step forecasting for two-minute and five-minute intervals, and a fifteen-step ahead forecasting for five-minute intervals.

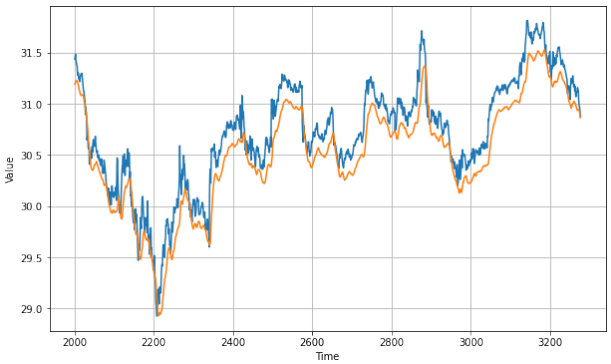
4.3.1 Varying window sizes

The purpose of this experiment is to define an optimum window size in order to face the characteristic of non-stationary time series. The experiments show that for time series with two-minute intervals, the optimum window size consists of 60 data points.

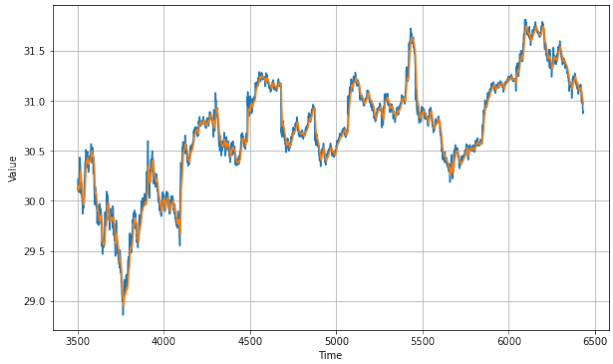
Figure 4.1a shows the one-step-ahead prediction over the test dataset with a window size of 40, this plot shows how the forecasted values tend to be below the actual values. Figure 4.1e shows the prediction of using a window size of 80, in this case, the forecasted values tend to be above the actual ones. Figure 4.1c shows how the forecasted values are quite close to the actual values. In the case of five-minute intervals, Figure 4.1f shows the one-step-ahead predictions using a window size of 108 correspondings to a period of time of 9 trading hours. This window size provides the best forecastings over the test dataset.



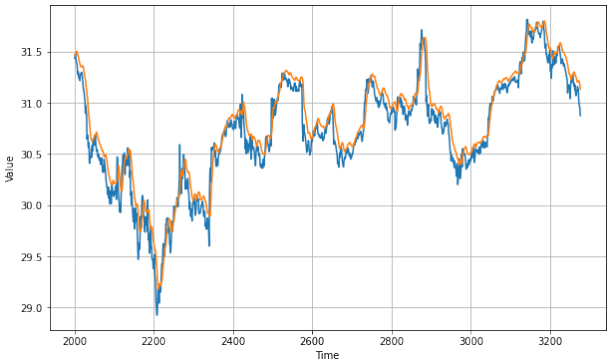
(a) Interval: 2 minutes - Window Size: 40



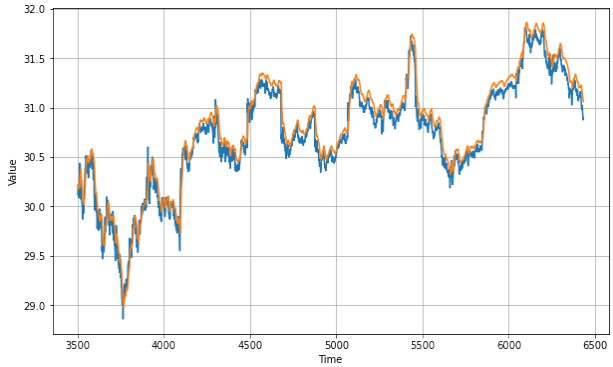
(b) Interval: 5 minutes - Window Size: 46



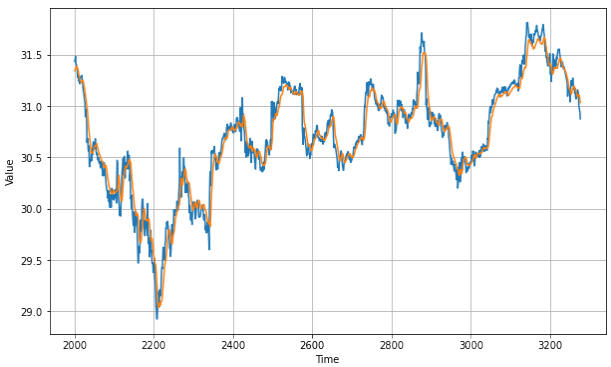
(c) Interval: 2 minutes - Window Size: 60



(d) Interval: 5 minutes - Window Size: 72



(e) Interval: 2 minutes - Window Size: 80

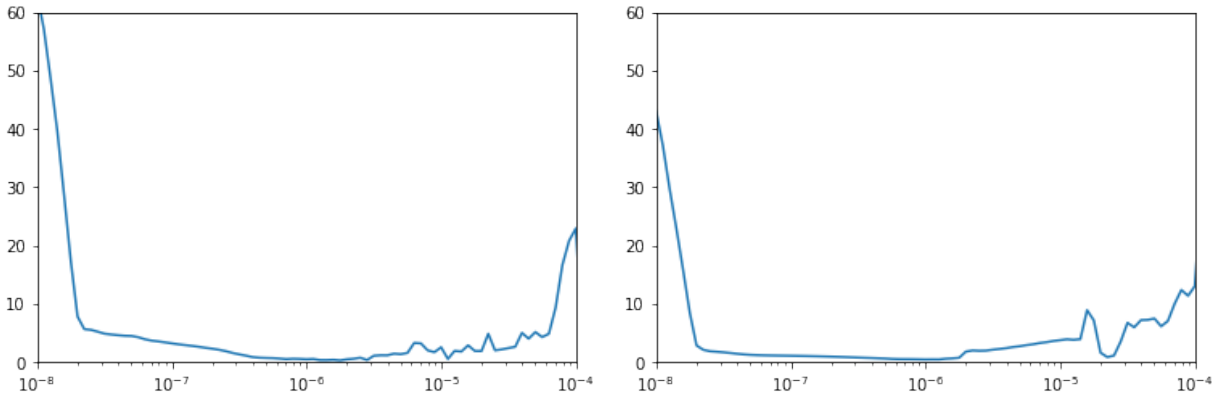


(f) Interval: 5 minutes - Window Size: 108

Figure 4.1: One-step ahead forecasting (Orange) vs. Test Dataset (Blue). A window size of 40 provides the best results in the case of two-minute intervals and a window size of 108 for five-minute intervals.

4.3.2 Optimum learning rate selection

In order to define the learning rate for training. A learning rate scheduler provides an approximation of the optimum learning rate. Figure 4.2 shows the resulting loss values for varying the learning rate for 100 training epochs. The performed experiments show that the optimum learning rate for both datasets varies between $5e^{-7}$ and $5e^{-6}$. The optimum learning rate depends on multiple factors such as the number of training epochs, model parameters, or dataset.



(a) Optimum learning rate: $2e^{-6}$.

(b) Optimum learning rate: $1e^{-6}$.

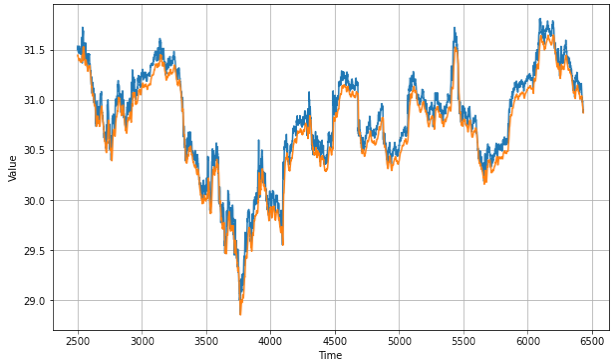
Figure 4.2: Optimum learning rate selection.

4.3.3 Training dataset

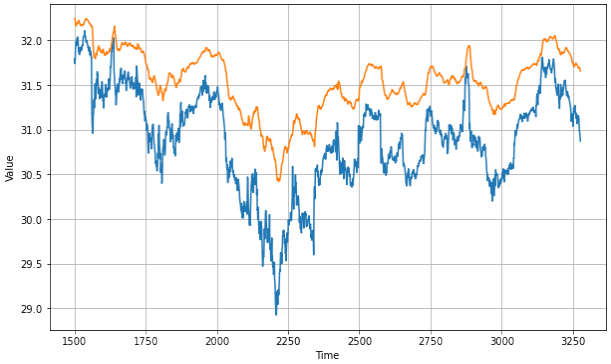
The following experiment shows how the performance of the model is affected by varying the number of training examples when creating the dataset. In the case of two-minute intervals and a training data set of 2500 data points, Figure 4.3a shows how the one-step-ahead forecasted values tend to be below the actual values. Figure 4.3c and 4.3e corresponds to the forecasted values with a dataset composed of 3500 and 4500 data points. Since the dataset is composed of non-stationary time series, it is interesting to note that the best forecasting performance corresponds to the training dataset with 3500 data points.

Figure 4.3e shows how the forecasted values tend to exceed the actual stock prices. The experiments performed for five-minute intervals show a similar behavior was 2000 data point provides the best performance. Figure 4.3b shows how the forecasted values tend to be below the actual values, Figure 4.3f shows how the forecasted values differ drastically from the actual ones when the training examples extends for a long period of time. Figure 4.3d shows how the model using 2000 data points for training provides quite similar results to the actual values.

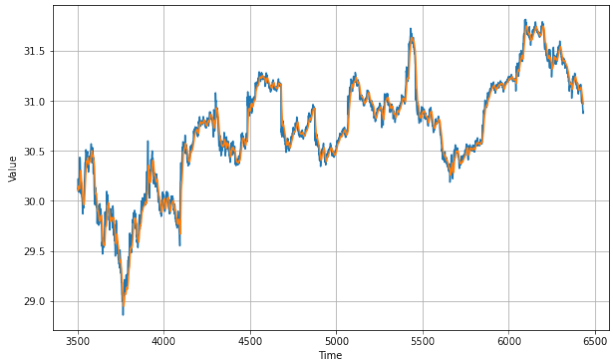
It is interesting to note that when dealing with non-stationary time series, increasing the number of training examples does not mean improving the model performance. This experiment shows how selecting an optimum number of training examples can improve the prediction performance.



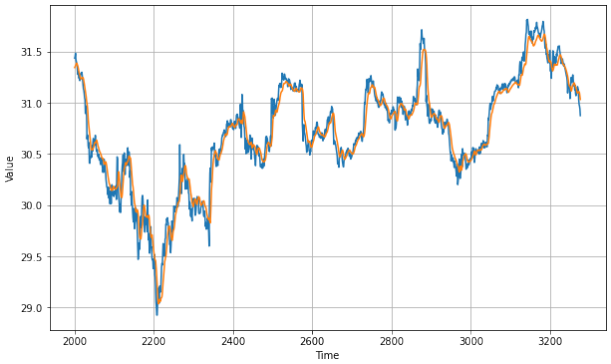
(a) Interval: 2 minutes - Training dataset: 2500



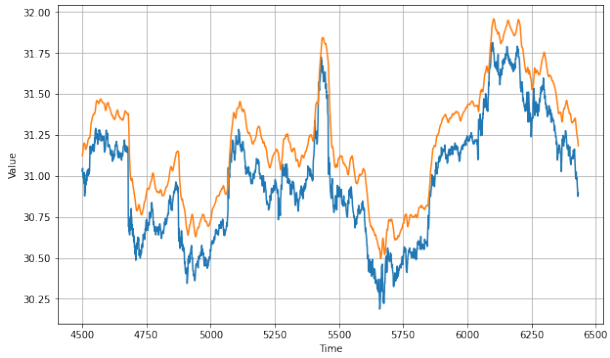
(b) Interval: 5 minutes - Training dataset: 1500



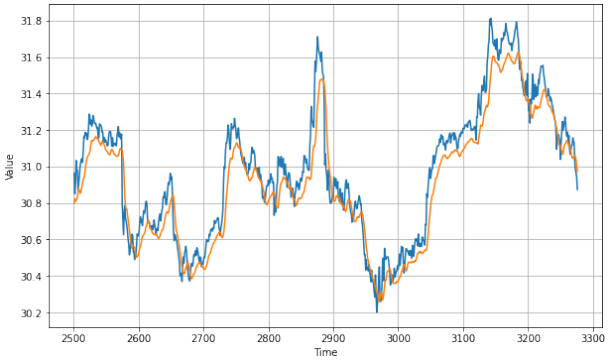
(c) Interval: 2 minutes - Training dataset: 3500



(d) Interval: 5 minutes - Training dataset: 2000



(e) Interval: 2 minutes - Training dataset: 4500



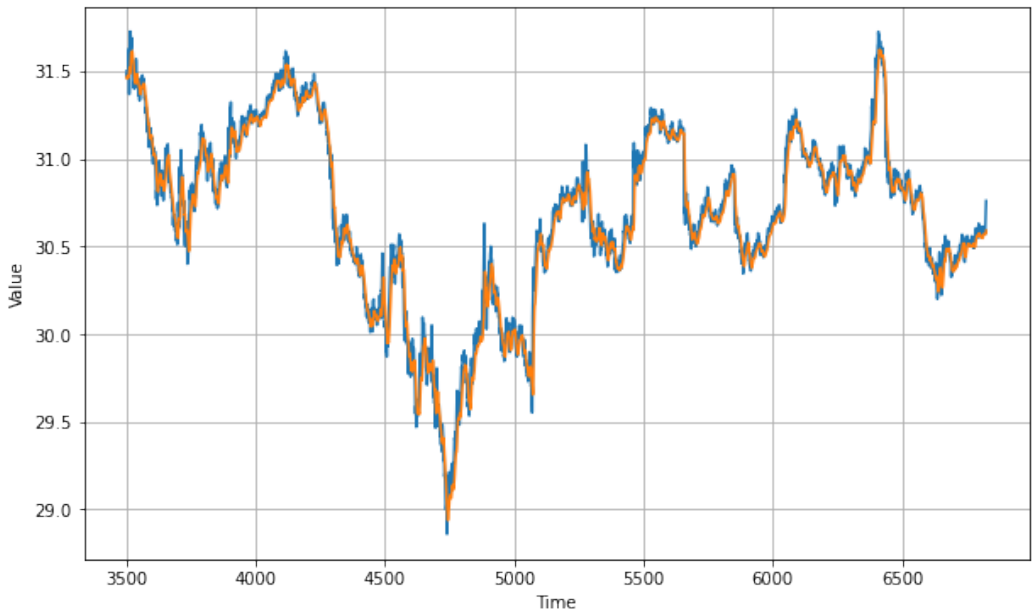
(f) Interval: 5 minutes - Training dataset: 2500

Figure 4.3: One-step ahead forecasting (Orange) vs. Test Dataset (Blue). This experiment consists of varying the number of training examples.

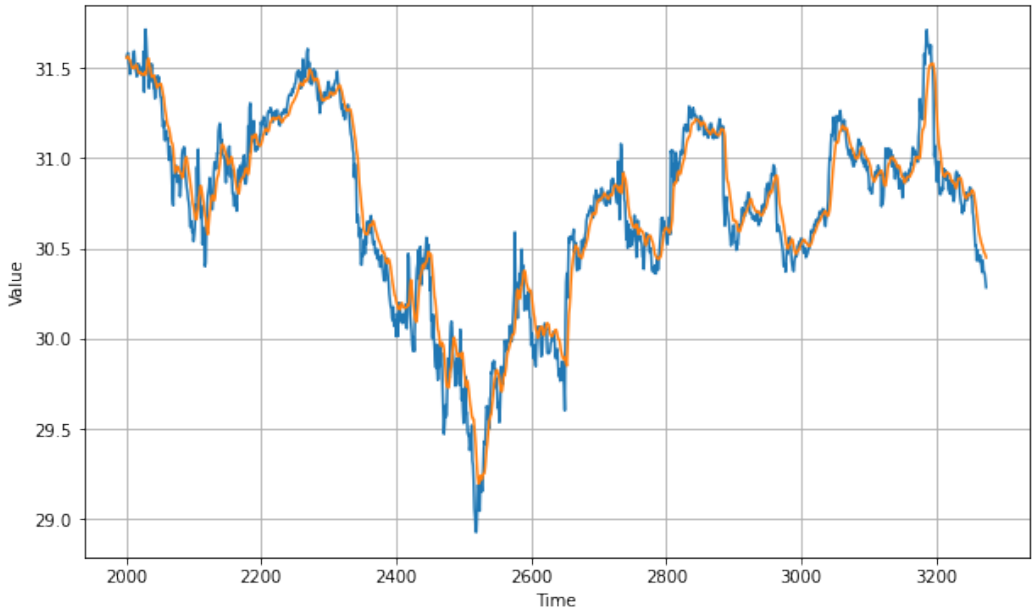
4.3.4 Forecasting

The purpose of this section is to evaluate the model for one-step and four-step ahead forecasting by using the test dataset consisting of two and five-minute intervals. The corresponding metrics for each experiment are present in section 4.4. Table 5.1 corresponds to the one-step, four-step and fifteen-step forecasting approaches. Tables 5.2 and 5.3 corresponds to the training example and window size experiments respectively.

One-step forecasting



(a) One-step ahead forecasting for 2-minute intervals dataset.



(b) One-step ahead forecasting for 5-minute intervals dataset.

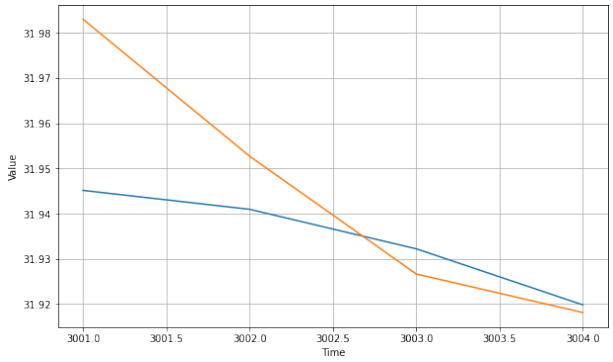
Figure 4.4: One-step ahead forecasting (Blue) vs. Test Dataset (Orange).

Figure 4.4 shows how the model performs for one-step forecasting, Figure 4.4a for 2 minute dataset and Figure 4.4b in the case of 5 minute intervals dataset. This experiment shows that the model trained for the 2-minute intervals dataset performs more accurately than the model for 5-minute intervals. However, both models provide good performance for forecasting the next data point given non-stationary time series as input. In both cases,

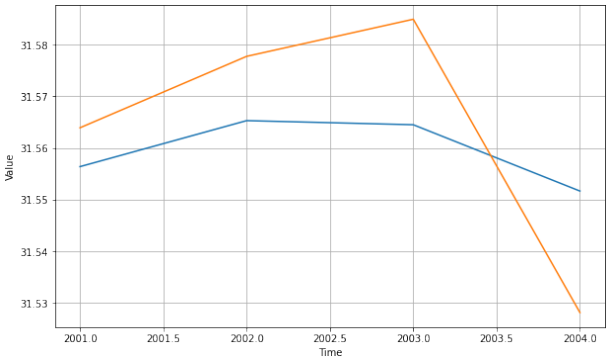
the selected number of training examples and window sizes corresponds to the values which provide the best performance on previous experiments. For the two-minute intervals dataset, window sizes of 60 and 3500 training examples. For the five-minute intervals dataset, window sizes of 108 and 2000 training examples.

Four-step forecasting

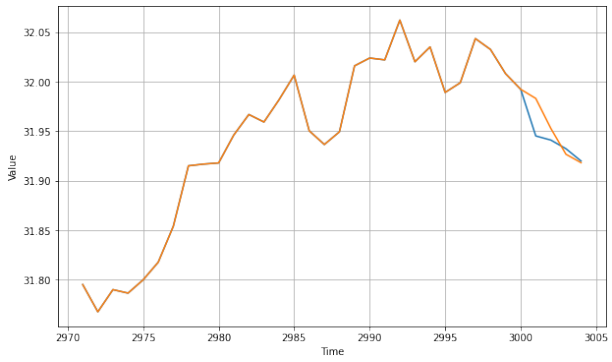
Figure 4.5 shows the model performance for four-step ahead forecasting. The first row corresponds to the forecasted values for 2-minute and 5-minute intervals datasets. The second row corresponds to the input sequence followed by the four predicted values. In this sense, Figure 4.5a and 4.5c shows how the model performs for four-step ahead forecasting for 2 minute intervals dataset and Figure 4.5b and 4.5d in the case of 5 minute intervals dataset. In this experiment, the results show that four-step ahead forecasting is more accurate in the case of 2-minute intervals. Similar experiments were performed for forecasting longer time-steps where the model for forecasting 5-minute intervals time series provides more accurate results.



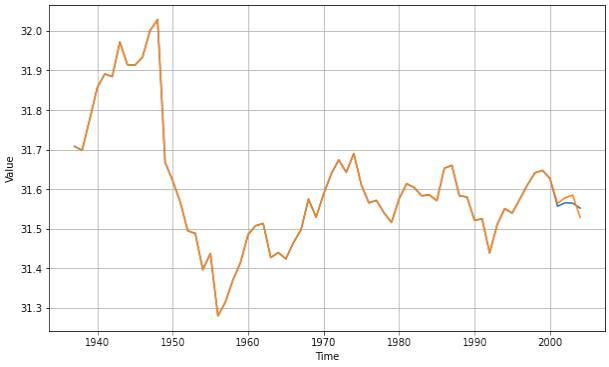
(a) Forecasted values for 2-minute intervals.



(b) Forecasted values for 5-minute intervals.



(c) Input sequence followed by forecasted values for 2-minute intervals dataset.

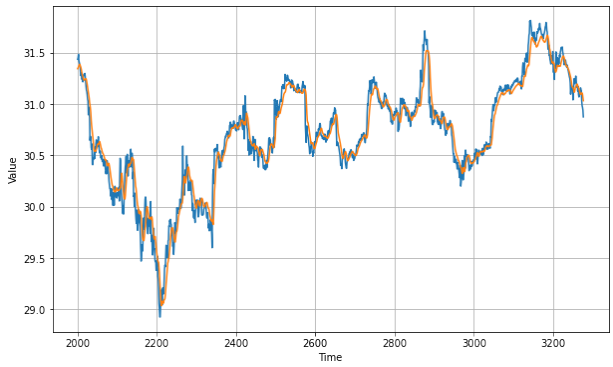


(d) Input sequence followed by forecasted values for 5-minute intervals dataset.

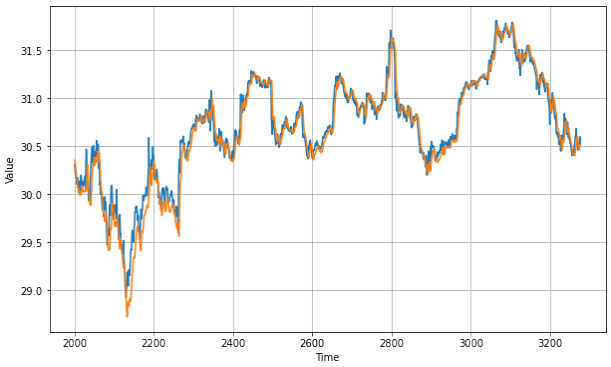
Figure 4.5: Four-step ahead forecasting (Blue) vs. Actual values (Orange).

Fifteen-step forecasting

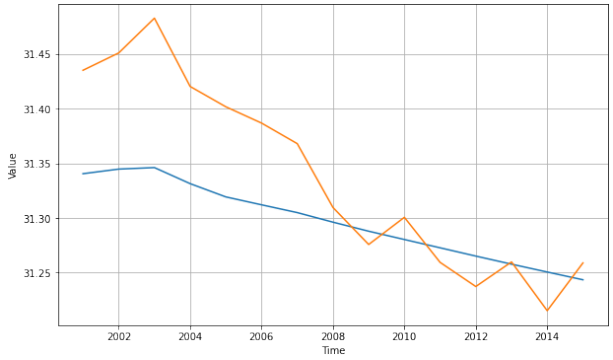
This is an additional experiment for testing the performance of the model for fifteen-step ahead forecasting. The results showed that the model trained for forecasting 5 minute intervals time series provides much more accurate results than the model for 2-minute intervals.



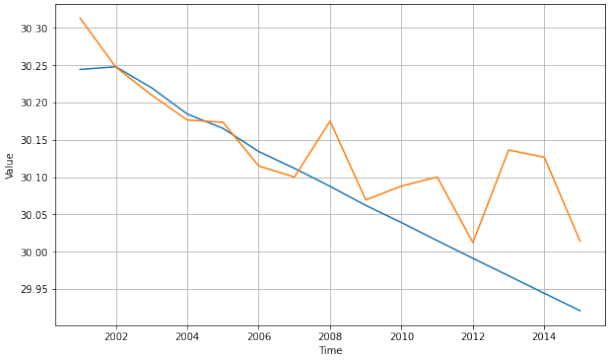
(a) Test time series corresponding to stock prices from the last sixty days ending on March 24.



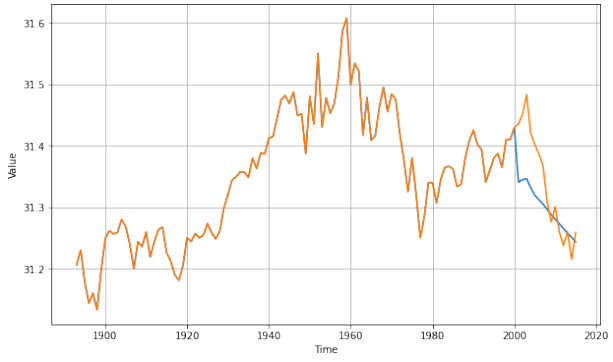
(b) Test time series corresponding to stock prices from the last sixty days ending on March 25.



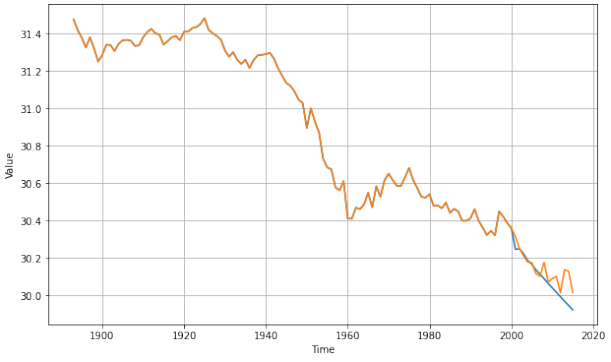
(c) Fifteen-step ahead forecasting (Blue) vs. Actual values (Orange) March 24, 2021.



(d) Fifteen-step ahead forecasting (Blue) vs. Actual values (Orange) March 25, 2021.



(e) Fifteen-step ahead forecasting (Blue) vs. Actual values and input sequence (Orange).



(f) Fifteen-step ahead forecasting (Blue) vs. Actual values and input sequence (Orange).

Figure 4.6: Fifteen-step ahead forecasting for March 24 and March 25, 2021.

Figure 4.6 only corresponds to the results provided by the model trained for forecasting 5-minute intervals. In this experiment the same forecasting approach is performed for two different time series, the first corresponds to the stock prices from the last sixty days ending at March 24 and the second corresponds to the last sixty days ending at March 25, 2021. Figure 4.6a and 4.6b shows the one-step-ahead forecasting for both time series in order to provide the context in which the prediction is performed. Figure 4.6c and 4.6d shows the model performance for fifteen-step ahead forecasting, this experiment shows that the model trained for 5-minute intervals can forecast the trend in which the non-stationary time series behaves for at least 15 data points ahead.

4.4 Evaluation Metrics

To measure the error rate of the developed models. The metric used in this project is mean absolute error (MAE). This metric does not penalize large errors as much as metrics such as MSE or RMSE. The mean absolute error is a common approach for measuring the error of in time series analysis. In this sense, to measure the error rate, the experiments were divided into three groups as shown in Table 4.1. The first group corresponds to one-step and multi-step forecasting, for each experiment the model was trained using five different time series. This approach results in five different MAE values returned by each model that later was averaged. The second and third groups correspond to the MAE values from the experiments consisting of varying the number of training examples and window sizes. For these two groups, the model was trained for a single time series, and therefore the MAE values corresponding to each experiment consist of a single evaluation.

Table 4.1: Metric definition used for different forecasting approaches.

Group	Approach	Metric
1	One-step and multi-step ahead forecasting	MAE
2	One-step ahead forecasting varying the number of training examples	MAE
3	One-step ahead forecasting varying the window sizes	MAE

Chapter 5

Results

The results of the model for one-step, four-step, and fifteen-step ahead prediction are shown in Table 5.1, this table describes the most important parameters used to achieve these results and the average error rate returned by the models that performed these experiments. The results shown in Table 5.2 corresponds to the experiments for varying the number of training examples in order to define the optimum number of training examples that provide the best performance. Table 5.3 shows the experimental results for varying the window size in order to face the drawbacks of treating with non-stationary time series and define an optimum period of time long enough to capture a high diversity in price movements.

Table 5.1: Performance results in terms of mean absolute error (MAE) for one-step and multi-step forecasting.

Approach	Interval	Window Size	Training Examples	MAE
One-step ahead forecasting	2 minutes	60	3500	6.7
One-step ahead forecasting	5 minutes	108	2000	9.94
Four-step ahead forecasting	2 minutes	60	3500	3.49
Four-step ahead forecasting	5 minutes	108	2000	8.07
Fifteen-step ahead forecasting	5 minutes	108	2000	9.84

5.1 Discussion results

The objective of this project is to measure the performance of deep architectures for forecasting non-stationary time series composed of stock prices. The proposed architecture is composed of CNNs, LSTMs, and densely-connected neural networks. The experimental results show that the proposed architecture provides the best performance for a short period of time, the best performance was achieved training the model with two-minute intervals time series, in this case, the error rate of the model is equal to 6.7. By using five minute

Table 5.2: Experimental results for experiments varying the number of training examples.

Approach	Interval	Training Examples	MAE
One-step ahead forecasting	2 minutes	2500	8.83
One-step ahead forecasting	2 minutes	3500	5.7
One-step ahead forecasting	2 minutes	4500	21.99
One-step ahead forecasting	5 minutes	1500	6.46
One-step ahead forecasting	5 minutes	2000	9.11
One-step ahead forecasting	5 minutes	2500	9.52

Table 5.3: Experimental results for experiments varying the window size.

Approach	Interval	Window Size	MAE
One-step ahead forecasting	2 minutes	40	11.53
One-step ahead forecasting	2 minutes	60	5.7
One-step ahead forecasting	2 minutes	80	9.49
One-step ahead forecasting	5 minutes	46	11.57
One-step ahead forecasting	5 minutes	72	10.32
One-step ahead forecasting	5 minutes	108	9.11

intervals time series for training, the model provides an error rate of 9.94, which is higher than the model trained for two-minute intervals but also provides good performance for a longer period of time. Figure 4.3c and 4.3d shows the forecasted values compared to the actual values for two and five minute intervals. In the case of four-step ahead forecasting, the model trained with two-minute intervals time series also provides a better error rate, 3.49 for two-minute intervals time series and 8.07 in the case of five-minute intervals time series.

Although the models trained for two-minute intervals time series provides better performance, the results show that models trained with five-minute intervals time series forecast more precisely the longer period of time than the two-minute intervals approach. The results show that for fifteen-step ahead forecasting, the model is able to maintain the error rate closer to 9 while the error rate for models trained with two-minute intervals time series is affected drastically. These results suggest that models trained with longer intervals time series are able to forecast the trend in which the market data behaves for at least fifteen steps ahead.

Regarding the analysis of non-stationary time series, the experimental results show that defining an optimum window size is fundamental for increasing the model accuracy. The selection of an optimum window size depends on the intervals of the time series. It was found that for two-minute intervals time series, the optimal window size is equal to 60 data points. In the case of five minute interval time series, the optimal window size is 108 data points. Another important factor to achieve good performance is the number of training

examples, the results show that for two-minute intervals time series the optimum number of training examples is 3500 and 2000 in the case of five-minute intervals time series. Contrary to the idea of deep learning models needing a large dataset, when treating with non-stationary time series, the idea of defining a period of time long enough to capture a high diversity in price movements becomes more important than collecting larger datasets.

The results provided by the model show that deep architecture performs accurately when forecasting stock market prices. In this sense, one-step and four-step ahead forecasting can be applied to high-frequency trading strategies. Since high-frequency trading strategies focused on short-term positions, the forecasted values by the model can be used as an indicator for determining a position. An advantage of using deep learning models in high-frequency trading is the speed at which the model provides accurate forecasting, this approach enables the possibility to exploit trading opportunities that may open up for milliseconds or seconds.

Since the financial market is a highly dynamic system, the patterns and dynamics existing within the model will not always correspond to the current dynamics of the financial market. Therefore, in order to maintain the performance of the model, a limitation of this approach is that the model must be trained constantly to allow the model to learn the current behavior of the financial market. In this sense, further researches could focus on extending the variables provided to the model to identify more complex features and increase the model accuracy. The approach of this model is to analyze univariate time series composed of stock opening prices. Further researches could focus on the analysis of multivariate time series applying deep learning algorithms.

Chapter 6

Conclusions

This work has presented a deep learning model that combines a CNN layer with two LSTM layers and three regular densely-connected NN layers for intraday stock price forecasting. The model uses as input a batch dataset built from non-stationary time series. Results presented in table 5.1 show that the model can perform one-step and multi-step ahead forecasting with a low error rate.

The proposed model uses only the opening stock price of the last sixty days to build the dataset based on a sliding window approach. These results reinforce the hypothesis that selecting shorter sequences of data points for training the model help to overcome the drawbacks of treating non-stationary time series. The experimental results show that choosing an optimum window size and number of training examples can drastically improve the model accuracy.

Regarding the model architecture, it can be concluded that the combination of different deep architectures improves the capability of the model for identifying interrelations within the time series to allow to forecast changes in trends of the stock market. The results provided by the model show that deep architectures can be applied successfully to trading strategies due to the speed at which the model performs accurate forecasting of stock prices.

Although the proposed model has a satisfactory forecasting performance, it still has some insufficiencies. For example, a multivariate dataset could provide to the model the possibility of identifying more complex features within the data to improve the model accuracy. In addition, since the model must be constantly trained to learn the current behavior of the stock prices, this process becomes time-consuming, therefore the introduction of high-performance computing (HPC) techniques for training the model can improve the performance of possible trading strategies. In order to evaluate the effectiveness of deep learning models in portfolio management, fundamental and technical analysis can be addressed to develop different trading strategies. All of these recommendations could be implemented in future studies.

Bibliography

[1] PrimoAI (Powered by MediaWiki), "Gradient descent optimization challenges," 2020, [Latest edited on 25 September 2020, at 08:41]. [Online]. Available: http://primo.ai/index.php?title=Gradient_Descent_Optimization_%26_Challenges#Gradient_Descent_-_Stochastic_.28SGD.29.2C_Batch_.28BGD.29_.26_Mini-Batch

[2] J. Durbin and S. J. Koopman, *Time series analysis by state space methods*. Oxford university press, 2012.

[3] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[4] R. Mizuno, "The male/female ratio of fetal deaths and births in japan," *The Lancet*, vol. 356, no. 9231, pp. 738–739, 2000.

[5] J. M. MacDonald, "Demand, information, and competition: why do food prices fall at seasonal demand peaks?" *The Journal of Industrial Economics*, vol. 48, no. 1, pp. 27–45, 2000.

[6] J. Wu and S. Wei, "Time series analysis," *Hunan Science and Technology Press, ChangSha. Available online at: http://www2. geog. ucl. ac. uk/˜ mdisney/teaching/-GEOGG121/time_series/GEOGG121_5_TimeSeries_ Wu. pdf. Retrieved: January*, vol. 20, p. 2018, 1989.

[7] Q. Zhang, R. Luo, Y. Yang, and Y. Liu, "Benchmarking deep sequential models on volatility predictions for financial time series," *arXiv preprint arXiv:1811.03711*, 2018.

[8] N. E. Huang, M.-L. Wu, W. Qu, S. R. Long, and S. S. Shen, "Applications of hilbert–huang transform to non-stationary financial time series analysis," *Applied stochastic models in business and industry*, vol. 19, no. 3, pp. 245–268, 2003.

[9] B. G. Malkiel, "The efficient market hypothesis and its critics," *Journal of economic perspectives*, vol. 17, no. 1, pp. 59–82, 2003.

[10] M. C. Jensen, "Some anomalous evidence regarding market efficiency," *Journal of financial economics*, vol. 6, no. 2/3, pp. 95–101, 1978.

[11] R. S. Tsay, *Analysis of financial time series*. John wiley & sons, 2005, vol. 543.

[12] S. Siami-Namini and A. S. Namin, "Forecasting economics and financial time series: Arima vs. lstm," *arXiv preprint arXiv:1803.06386*, 2018.

[13] E. I. Altman, G. Marco, and F. Varetto, "Corporate distress diagnosis: Comparisons using linear discriminant analysis and neural networks (the italian experience)," *Journal of banking & finance*, vol. 18, no. 3, pp. 505–529, 1994.

[14] H. White, "Economic prediction using neural networks: The case of ibm daily stock returns," in *ICNN*, vol. 2, 1988, pp. 451–458.

[15] H. Situngkir and Y. Surya, "Neural network revisited: perception on modified poincare map of financial time-series data," *Physica A: Statistical Mechanics and its Applications*, vol. 344, no. 1-2, pp. 100–103, 2004.

[16] G. S. Atsalakis and K. P. Valavanis, "Surveying stock market forecasting techniques–part ii: Soft computing methods," *Expert systems with applications*, vol. 36, no. 3, pp. 5932–5941, 2009.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[18] A. Arévalo, J. Niño, G. Hernández, and J. Sandoval, "High-frequency trading strategy based on deep neural networks," in *International conference on intelligent computing*. Springer, 2016, pp. 424–436.

[19] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS one*, vol. 12, no. 7, p. e0180944, 2017.

[20] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using lstm, rnn and cnn-sliding window model," in *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE, 2017, pp. 1643–1647.

[21] M. R. Vargas, B. S. De Lima, and A. G. Evsukoff, "Deep learning for stock market prediction from financial news articles," in *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE, 2017, pp. 60–65.

[22] J. Cao, Z. Li, and J. Li, "Financial time series forecasting model based on ceemdan and lstm," *Physica A: Statistical Mechanics and its Applications*, vol. 519, pp. 127–139, 2019.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "Cyc: Toward programs with common sense," *Commun. ACM*, vol. 33, no. 8, p. 30–49, Aug. 1990. [Online]. Available: https://doi.org/10.1145/79173.79176

[25] K.-M. Schneider, "A comparison of event models for naive bayes anti-spam e-mail filtering," in *10th Conference of the European Chapter of the Association for Computational Linguistics*, 2003.

[26] A. C. Yoshua Bengio and P. Vincent, "Representation learning:a review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1798–1828, 2013.

[27] Y. Bengio, "Learning deep architectures for ai," *Foundations*, vol. 2, pp. 1–55, 01 2009.

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006, pMID: 16764513. [Online]. Available: https://doi.org/10.1162/neco.2006.18.7.1527

[29] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Icml*, 2011.

[30] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[32] D. Attwell and S. B. Laughlin, "An energy budget for signaling in the grey matter of the brain," *Journal of Cerebral Blood Flow & Metabolism*, vol. 21, no. 10, pp. 1133–1145, 2001.

[33] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[34] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks." *Journal of machine learning research*, vol. 10, no. 1, 2009.

[35] M. Ranzato, C. Poultney, S. Chopra, Y. LeCun *et al.*, "Efficient learning of sparse representations with an energy-based model," *Advances in neural information processing systems*, vol. 19, p. 1137, 2007.

[36] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[37] G. W. Smith and F. F. Leymarie, "The machine as artist: An introduction," in *Arts*, vol. 6, no. 2. Multidisciplinary Digital Publishing Institute, 2017, p. 5.

[38] U. Fiore, A. De Santis, F. Perla, P. Zanetti, and F. Palmieri, "Using generative adversarial networks for improving classification effectiveness in credit card fraud detection," *Information Sciences*, vol. 479, pp. 448–455, 2019.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[40] E. Park, W. Liu, O. Russakovsky, J. Deng, F.-F. Li, and A. Berg, "Large scale visual recognition challenge 2017," 2017.

[41] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2011, pp. 24–29.

[42] D. CireAan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural networks*, vol. 32, pp. 333–338, 2012.

[43] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[44] Ç. Gülçehre and Y. Bengio, "Knowledge matters: Importance of prior information for optimization," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 226–257, 2016.

[45] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[46] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–21, 01 2019.

[47] H. Fayek, M. Lech, and L. Cavedon, "Evaluating deep learning architectures for speech emotion recognition," *Neural Networks*, vol. 92, 03 2017.

[48] D. Otter, J. Medina, and J. Kalita, "A survey of the usages of deep learning in natural language processing," 07 2018.

[49] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[50] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[51] A. Joshi, P. Bhattacharyya, and M. J. Carman, "Automatic sarcasm detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 5, pp. 1–22, 2017.

[52] A. Grover, A. Kapoor, and E. Horvitz, "A deep hybrid model for weather forecasting," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 379–386.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[54] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[55] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[56] I. Sutskever, *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

[57] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[58] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*. PMLR, 2015, pp. 2048–2057.

[59] L. Bottou, "From machine learning to machine reasoning," *Machine learning*, vol. 94, no. 2, pp. 133–149, 2014.

[60] B. Wang, H. Huang, and X. Wang, "A novel text mining approach to financial time series forecasting," *Neurocomputing*, vol. 83, pp. 136–145, 2012.

[61] G. J. Deboeck, *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets*. John Wiley & Sons, 1994, vol. 39.

[62] B. G MALKIEL, "A random walk down wall street the time-tested strategy for successful investing," 2021.

[63] J. J. Murphy, *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[64] T. Bollerslev, J. Marrone, L. Xu, and H. Zhou, "Stock return predictability and variance risk premia: statistical inference and international evidence," *Journal of Financial and Quantitative Analysis*, pp. 633–661, 2014.

[65] M. A. Ferreira and P. Santa-Clara, "Forecasting stock market returns: The sum of the parts is more than the whole," *Journal of Financial Economics*, vol. 100, no. 3, pp. 514–537, 2011.

[66] J. H. Kim, A. Shamsuddin, and K.-P. Lim, "Stock return predictability and the adaptive markets hypothesis: Evidence from century-long us data," *Journal of Empirical Finance*, vol. 18, no. 5, pp. 868–879, 2011.

[67] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, "Stock price prediction using the arima model," in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*. IEEE, 2014, pp. 106–112.

[68] P. H. Franses and H. Ghijsels, "Additive outliers, garch and forecasting volatility," *International Journal of forecasting*, vol. 15, no. 1, pp. 1–9, 1999.

[69] N. Sarantis, "Nonlinearities, cyclical behaviour and predictability in stock markets: international evidence," *International Journal of Forecasting*, vol. 17, no. 3, pp. 459–482, 2001.

[70] Y. E. Cakra and B. D. Trisedya, "Stock price prediction using linear regression based on sentiment analysis," in *2015 international conference on advanced computer science and information systems (ICACSIS)*. IEEE, 2015, pp. 147–154.

[71] N. I. Sapankevych and R. Sankar, "Time series prediction using support vector machines: a survey," *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, 2009.

[72] P. Ou and H. Wang, "Prediction of stock market index movement by ten data mining techniques," *Modern Applied Science*, vol. 3, no. 12, pp. 28–42, 2009.

[73] L. A. Teixeira and A. L. I. De Oliveira, "A method for automatic stock trading combining technical analysis and nearest neighbor classification," *Expert systems with applications*, vol. 37, no. 10, pp. 6885–6890, 2010.

[74] C.-J. Huang, D.-X. Yang, and Y.-T. Chuang, "Application of wrapper approach and composite classifier to the stock trend prediction," *Expert Systems with Applications*, vol. 34, no. 4, pp. 2870–2878, 2008.

[75] H. Jia, "Investigation into the effectiveness of long short term memory networks for stock price prediction," *arXiv preprint arXiv:1603.07893*, 2016.

[76] J. Roman and A. Jameel, "Backpropagation and recurrent neural networks in financial analysis of multiple stock market returns," in *Proceedings of HICSS-29: 29th Hawaii international conference on system sciences*, vol. 2. IEEE, 1996, pp. 454–460.

[77] J. Heaton, N. G. Polson, and J. H. Witte, "Deep learning in finance," *arXiv preprint arXiv:1602.06561*, 2016.

[78] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

Appendices