Alma Mater Studiorum · University of Bologna

SCHOOL OF SCIENCES

Master's Degree Course in Computer Science

Prediction of the performance of financial stocks through the use of deep learning techniques and complex networks

Supervisor:
Chiar.mo Prof.
Stefano Ferretti

Presented by:
Danilo Mastria

Third Degree Session
Academic Year: 2020-2021

To my parents, Rocco and Cinzia, who taught me the value of sacrifice and have always supported me in my moments of difficulty.

Index

Introduction

Financial time series forecasting, which aims to predict future stock price trends, has been the subject of studies from various disciplines in recent years, such as technical analysis initially and machine learning and deep learning recently, because It is one of the methods most used by investors before making any investment.

The basic idea has always been to predict the performance of a stock by analyzing and modeling its historical behavior which, however, in the field of finance, above all, is influenced by many variables, including external ones. like politics or financial news.

In the current state of the art of studying stock price prediction or stock performance without taking into account the structural information of stock time series, most research efforts have been devoted to various types of stocks. internet sources of information and dynamic indicators derived from share prices.

Currently, deep learning has become the most effective method for capturing the characteristics of complex financial time series by proposing different structures that have as a common goal the forecast of the price of a share. Some of these have also been enriched with attention mechanisms to capture the interactions between different variables. However, the long-term dependencies of financial time series, in the experiments done so far, have not yet been fully captured due to the complex temporal evolution of the interactions between all time points. Probably the value of a data point at time t depends both on the points temporally close to it and on those further away, defined as historical with respect to t. These dependencies are then referred to as short term and long term respectively.

In the financial field, it is essential to be able to use the dependencies between the values in the given series and thanks to the discovery of the LSTMs it was possible to perform several convincing tests on daily, weekly stock returns, monthly and even annually.

On the other hand, however, financial time series often present chaotic and complex behaviors of the movement of prices, so much so that they are defined as not stationary. Abrupt changes or unexpected reversals are taken as an "outlier" in modeling and therefore weaken the capacity of generalization of learning models. In particular, the chaotic property seriously questions these models by making them assign incorrect weights to points that are not useful for predicting further trends. So, the predictive capacity of current models is limited, as it is profoundly undermined by both of the problems just mentioned.

By exploiting the methods of complex networks to characterize dynamic systems derived from the time series, it might be possible to use structural information embedded in the time points of the series to provide new insights into stock prediction. Since the theory of complex networks and the analysis of nonlinear time series are generally considered domains of the science of complex systems, the adoption of methods of complex networks is became a popular way of nonlinear time series analysis, thus addressing fundamental questions regarding addictions long-term and the chaotic property in the prediction of time series.

From some experiments carried out on different fields of the time series it has been confirmed that the structural information extrapolated from the converted graphs from the historical series (specifically from the associations of the nodes and their weights) provide promising assistance in forecasting financial time series. First, thanks to the existence of explicit arcs between distant nodes in graphs, long-term dependencies in a time series can be captured directly through the associations between time points. By linking distant time points, long-range information can be delivered faster through these associations, thus avoiding the disappearance of information in recurrent neural networks. Furthermore, by identifying the content protruding from the chaotic time series, the weights of the graph nodes

they can provide additional knowledge for temporal attention in such a way as to being able to address the problem when manipulating financial time series.
This article then initially introduces a framework, created and tested by some Chinese researchers of the "Beihang" University of Beijing [1], that through the fusion of complex networks and artificial intelligence, succeeds to predict the performance of equities obtaining an accuracy greater than neural network models currently present in the state of the art.
Subsequently, however, the same framework comes first with a different dataset containing more recent data influenced by Sars-Cov-2 pandemic (which the authors said they wanted to exclude king in their tests) to observe his behavior; while to follow they come made small changes to the model, calculating different metrics on graphs, to compare if improvements in terms can be obtained percentage of accuracy.
The thesis is therefore divided into four chapters organized as follows: in the first one present an introduction to the main concepts of the state of the art that they are present in this work; in the second the framework and its functioning are presented in detail, in the third the new tests that will come are introduced performed and the implementations of some important phases of the flow of prediction, while in the fourth the experiments carried out and the results obtained are examined, trying to give them meaning. Finally they will come discuss the conclusions and any future developments.

Chapter 1

State of art

The first chapter introduces the main and necessary concepts in order to better understand what the framework object of this thesis is about and what it is composed of.

1.1 Definition of Graph

A graph is a set of elements called vertices or nodes that can be connected together by lines called arcs or links. In a more formal way, a triple G = (V, E, f) is called a graph where V indicates the set of nodes, E indicates the set of arcs and f is a function that associates to each arc and in And two vertices u, v in V. Two vertices u, v connected to an arc and are called ends of the arc.

If E is a symmetric relation then the graph is said to be indirect or undirected, otherwise it is said to be direct or directed. Finally, a graph is said to be complete if any two of its vertices are adjacent (i.e. there is an arc connecting them).

1.2 Complex network

A complex network is a graph that presents a non-trivial topological structure, i.e. its properties differ from those of a regular network and

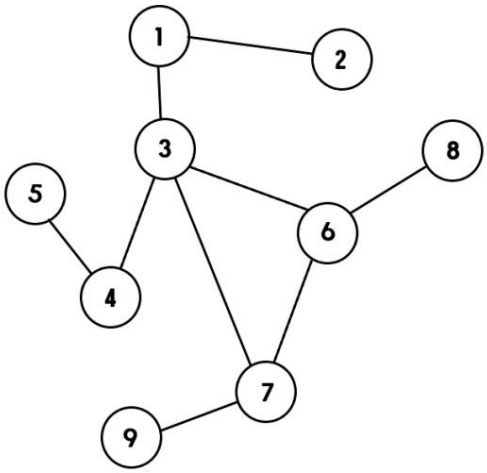

Figure 1.1: Example of a simple graph [1]

random. Natural systems often show complex properties related to the organization of connections within the network [2].
Technological development and the ability to collect more data has led to the emergence of ever larger and more complex networks showing characteristics non-intuitive that can come to be made up of millions of communicating units. This is how the network can be considered the model of reality,
where the distance between nodes measures the level of interaction between two elements in a network. The greater the distance, the less strong the interaction.
Types of complex real networks can be the social ones (social networks, telephone calls), the technological ones (internet, pc) or the biological ones (epi demiology).
One can think of a complex network by imagining the whole
of friendships that are made on facebook, where each node represents one person and the link that connects it to another node symbolizes the fact that the two they made friends.

1.3 Time series

Time-series data are sequences of values
ordered over time and marked by a historical moment. Understanding the time series and learning to model is crucial for many

Figure 1.2: Example of a complex network [2]

contexts because it allows us to predict future trends or behaviors e
act accordingly. Many businesses produce financial data
ampersand with the exact moment they are generated.

1.4 Artificial Intelligence

The term Artificial Intelligence (AI) [3] [4] identifies a branch of computer science that deals with the design and programming of hardware and software systems that, integrated with the machines, allow them to think
and act like a human being. It is intelligence understood both as the ability to computation or knowledge of abstract data and both like all those different forms of intelligence that are recognized by Gardner's theory.
The development of an intelligent system has the purpose of recreating one or more of these different forms of intelligence which, even if they are often defined simply as human minds, can actually be traced back to particular behaviors. reproducible by some machines. If you want to reproduce the operation of the human brain, AI should know:

- "Thinking humanly", ie being able to solve a problem
  with cognitive functions;

- "Act humanly", that is, indistinctly with respect to a being human;
- "Thinking rationally", that is, exploiting logic as a being does human;
- "Act rationally", that is to start a process to obtain the best expected result based on the information you have.

On the basis of this, the scientific community was able to classify intelligence artificial in two research branches:

- Weak Artificial Intelligence, which encompasses all those systems that do not have as their objective that of creating machines that have human intelligence, but systems capable of operating successfully in relation to complex human functions. They must be systems capable of simulating human behavior, without ever equaling or surpassing it.
  The basic idea is to solve a specific problem by investigating similar cases, developing a series of solutions and choosing the one that best fits the case under consideration.
- Strong Artificial Intelligence, which includes all those systems defined as "wise", that is, which may be able to develop their own intelligence, autonomously and without emulating human-like thought processes or cognitive abilities.

Two specific disciplines were born from these two categories: machine learning and deep learning, which are nothing more than one another's specialization (as can be seen from figure 1.3.

1.4.1 Machine learning

The term "Machine learning" is the general term used to indicate when computers learn from data.
Machine learning has deep links to the field of mathematical optimization, which provides methods, theories and domains of application. In fact, this discipline describes the intersection between computer science and statistics, in which

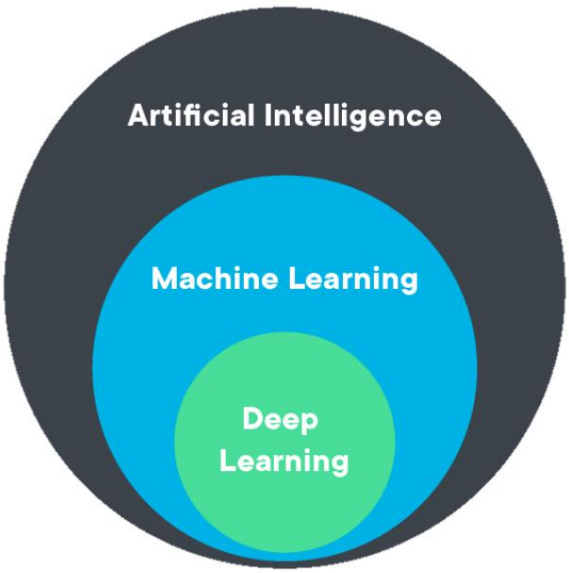

Figure 1.3: Artificial Intelligence 3

algorithms are used to perform a specific task without being explicitly programmed.
Many machine learning problems are formulated as problems of minimization of a certain loss function against a certain set of examples (training set). This function expresses the discrepancy between the values predicted by the model being trained and the expected values for each instance of the example.

The final goal is therefore to teach the model the ability to correctly predict the expected values on a set of samples not present in the training set, called test set, by minimizing the loss function in this set of instances. This leads to a greater generalization of the prediction head city.

The different machine learning tasks are typically classified into three broad categories, characterized by the type of feedback on which the learning system is based:

1. Supervised learning, in supervised learning there are

data to the system of input samples and their desired outputs, with the aim of learning a general rule (a function) capable of mapping input-output;

2. Unsupervised learning, on the other hand, in unsupervised learning, the model is provided only with input samples, without any expected output, with the aim of learning some structure in the input data. Unsupervised learning can be used, for example, for the discovery of hidden patterns in the data or to extrapolate salient features of the data, perhaps useful for the execution of another machine learning task;

3. Reinforcement learning, finally in learning with reinforcement the system interacts with a dynamic environment in which he has to achieve a certain goal (for example, driving a car or facing an opponent in a game). As the computer explores the domain of the problem, it is provided with feedback in terms of rewards or punishments, so that at each step it can be directed to the best solution.

Another yardstick according to which it is possible to distinguish different categories of tasks is the type of output expected from a certain machine learning system.
Among the main categories we find:

1. Classification, in which the inputs are divided into two or more classes and the learning system creates a model capable of assigning one or more classes among those available to an input. These types of tasks are typically addressed using supervised learning techniques.

2. Regression, regression is conceptually similar to classification with the only difference on the output domain which will be continuous and non-discrete. It too is typically addressed with learning
supervised.

3. Clustering, in which, as in classification, a set of data is divided into groups which, however, are not known a priori. The very nature of the

problems belonging to this category typically makes them unsupervised learning tasks.

1.4.2 Deep learning

Deep learning models are designed to continuously analyze data with a logical structure similar to that used by humans to draw conclusions.

To achieve this, deep learning applications use a multi-layered structure of algorithms called the Artificial Neural Network (ANN). The design of this ANN is inspired by the biological neural network of the human brain, leading to a learning process much more capable than that of standard machine learning models.

Considering the example of the ANN in figure 1.4, it can be seen that the level

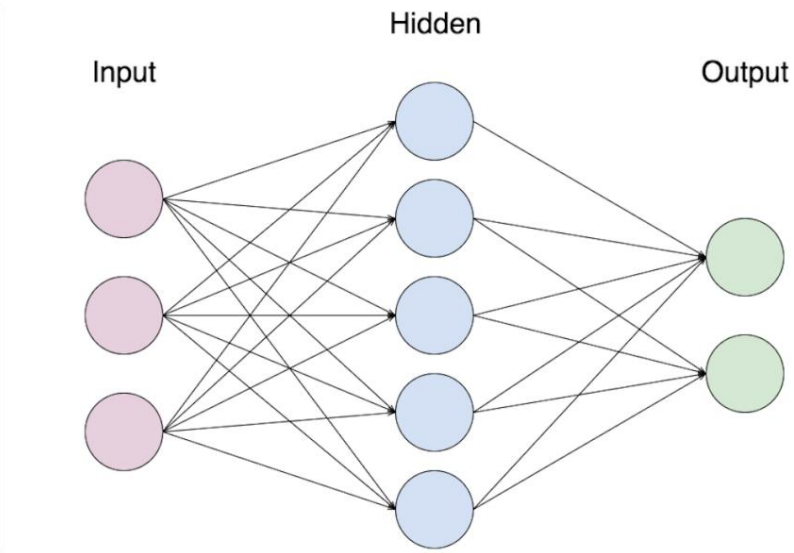

Figure 1.4: Structure of an Artificial Neural Network [4]

The leftmost is called the input level and the rightmost is the output level. The intermediate levels are called hidden levels instead, because their values are not observable in the training set. Put simply, hidden levels are calculated values used by the network to do its "magic". The depth of a network is defined by the number of hidden levels present. In general,

any ANN with two or more hidden layers is called a deep neural network.
While this shows the enormous potential of deep learning, there are two main reasons it has only recently achieved so much usability: data availability and computing power.

First, deep learning requires incredibly large amounts of data.
Second, deep learning requires a lot of computing power.
However, with the emergence of cloud computing infrastructure and high-performance GPUs (graphics processing units, used for faster computations), the time to train a deep learning network has been considerably
reduced.

1.5 Artificial Neural Networks

The graphical representation of a neural network can be seen as a flow graph where:

- the initial nodes correspond to the input nodes to which the data is passed;
- the final nodes represent the output nodes that return values according to the input data;
- intermediate nodes are the hidden nodes within the network, in which data manipulation takes place.

Each node is part of a layer, but each layer can have multiple nodes.
A neural network will therefore have three types of layers respectively based on the type of nodes it contains, namely input layer, output layer and hidden layer. Hidden layers are called in this way because they are located inside the network and determine the depth of the latter. Data processing takes place inside the hidden layers, where more specifically a function called activation function is applied to each value arriving in a node.

The nodes of each layer communicate with the nodes of the previous layer or the next layer through the arcs, which transmit the processing data from the input layer to the output layer. Each node can be connected through multiple arcs both in incoming and outgoing. Each arch is characterized by a synaptic weight which

allows you to modify the data before delivering it to the destination node.
Data propagation can take place both forward and backward, and this choice determines the type of neural network.
There are different categories of neural networks that can be chosen based on the goal you want to achieve and the type of data you want to input.

1.5.1 Feed-forward neural network

A feed-forward network is the simplest artificial neural network model to be developed since all the various connections between the units of the layers do not form cycles. In this network, information moves in a single direction with respect to previous nodes, that is, forward.
The input samples are taken from the input nodes, then they are passed between the various hidden nodes up to the output nodes. When there is at least one hidden layer of neurons between the input and the output, we speak of multi-layer perceptron, vice versa if the hidden layers are equal to zero we have a single-layer perceptron. The example of figure 1.4 represents the structure of a simple feed forward neural network.

1.5.2 Recurrent Neural Network (RNN)

A recurrent neural network is a class of artificial neural network in which i neurons can be connected in a loop. Typically, the output values of a layer at a higher level are used as input values in a layer at a lower level. This interconnection between layers allows the use of one of these as state memory, and allows, by supplying a temporal sequence of values as input, to model a dynamic temporal behavior dependent on the information received at the previous instants of time.

In this case, unlike a multi-layer perceptron which can only map from input vectors to outputs, an RNN can map the entire history of previous inputs to each output. A new dimension is therefore introduced which corresponds to time, but we will not define n layers for each time instant, but we will use a concept of parameter sharing between the

moments in time.
This type of network gives the possibility to avoid a repetition by using
a single layer that takes as input in addition to the value of the previous layer, too
the output of the same neural network at the previous instant.

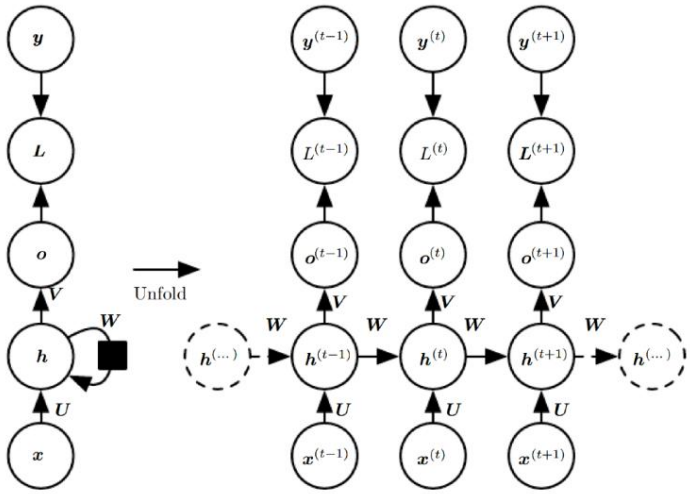

Figure 1.5: Example of an RNN [5]

1.5.3 Long Short-Term Memory (LSTM)

An LSTM is a type of recurrent neural network that was created for re
solve two problems that can occur when using an RNN
they want to capture long-term dependencies: exploding gradient and vani shing
gradient. In practice, the network comes to a point during training
in which he is no longer able to learn why the weights become respectively
either very large or very small. Using LSTM units these problems occur
they might check much less frequently.
A common LSTM unit is composed of a cell and three gates (input gate,
output gate and forget gate). The cell remembers the values over arbitrary time
intervals while the three gates regulate the flow of information in and out
from the cell.
LSTM networks are particularly suitable for classifying, processing and making

predictions based on time series data, as there can be delays of unknown duration between important events in a time series.

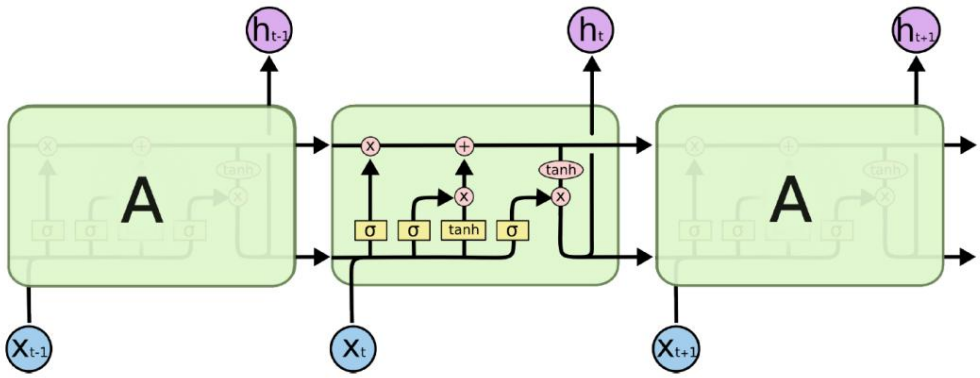

Figure 1.6: Example of an LSTM [6]

1.5.4 Convolutional Neural Network (CNN)

A convolutional neural network is a type of neural network belonging to the class of feed-forward networks which is inspired by the organization of the cortex visual.

In general it works like all other feed forward because it consists of an input layer, one or more hidden layers, which perform calculations through activation functions (for example RELU) and an output layer which carries out the actual classification. The difference with respect to the classic feed forward networks is represented by the fact that we do not have linear levels, but convolution levels.

Convolutional layers perform a very important job as they extract, through the use of filters, characteristics or features from the images whose content you want to analyze. Therefore, unlike a traditional feed forward that works “on the general information of the image”, a CNN works and classifies the image based on particular characteristics of the same. In other words, depending on the type of filter used, it is possible to identify different things in the reference image such as the contours of the figures, vertical lines, horizontal lines or diagonals.

Although it is a type of network that works in two dimensions, in the last

due to its power, it has also been used to work with series
financial time simply by adapting it to one-dimensional inputs.

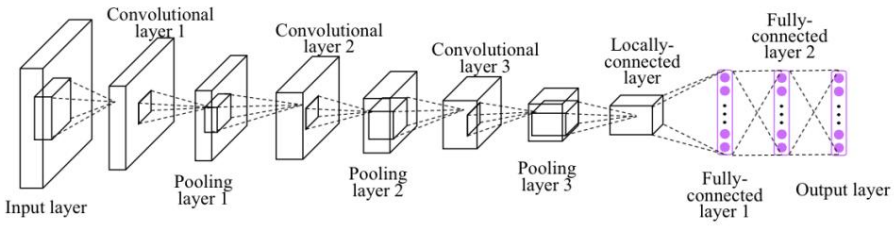

Figure 1.7: Example of structure of a CNN [7]

Chapter 2

Proposed framework

Before deep learning methodologies became popular, statistical models and machine learning were universally adopted for predicting financial time series [5] due to their good interpretative skills.

Among the most commonly used statistical models we find ARMA, GARCH and NARX whose use was aimed at verifying hypotheses concerning the financial market, where the forecasts were consequently based on these verified hypotheses. However, the chaotic behaviors that often appear with financial time series limit the predictive capacity of these models as they are unable to model the evolutionary process in financial systems.

In recent years, extensive studies have been undertaken to solve the forecasting problems of financial time series using deep learning methods thanks to their powerful expression ability.
RNNs developed specifically for sequenced data are very popular due to their superior performance in capturing nonlinear relationships.
However, traditional ones are insufficient in capturing long-term addictions due to the problem of gradient disappearance.
Based on the "memory cells", which were designed to hold information for a longer time, LSTMs were used which proved to be useful in forecasting equity returns. Therefore, LSTM is constantly used for the prediction of sequential or

financial time series and performs better than the classic RNN. However, the shortage of capturing long-term addictions still exists; that is, the performance of LSTM networks deteriorates rapidly as the length of the input sequence.
Also, due to the amplification of noise in the recurrence process of the model, the chaotic property of financial time series worsens the forecast performance of the model. Because the concept of attention represents the human intuition that certain portions of data are given more emphasis than to others, attention-based deep learning methods are widely used to learn the complex dependencies between
characteristics in the time series tasks.
Structure within the series is also considered important
temporal, that is the direct connection between distant temporal points than with the deep learning is still hard to exploit. For this reason the idea of using complex network methods for mapping dynamical systems derived from time series has become an active realm of non-time series analysis.
linear, which provided a guideline for addressing fundamental issues concerning long-term dependencies and chaotic property in forecasting
time series.
By transforming time series into graphs, researchers are able to measure the structural properties of time series and of capturing hidden structures embedded in chaotic time points. The fusion therefore of the use of networks complex and deep learning led to the creation of the framework that will come set out below to predict the future trend of asset prices.

2.1 Tools used

The tools used to manipulate the fra are listed below
mework, edit it and test it on your computer.

2.1.1 Python

Python is the programming language used to perform processing operations on data by mapping them into graphs and to create the
neural network. It is an object-oriented programming language, defined
of a higher level than most of the other languages belonging to it
to this category.

2.1.2 PyCharm

PyCharm is the integrated development environment (IDE) used for maneg jars and modifies the python code of the framework.

2.1.3 Anaconda

Anaconda is a distribution of the programming languages Python and
R for scientific computing (data science, machine learning applications, large-scale data processing, predictive analysis, etc ...), which aims to simplify
package management and implementation.
Package versions in Anaconda are managed by the
conda packages. This package manager was created as a separate open source package as it was useful on its own and for things other than
Python.
The Anaconda distribution comes with over 250 auto-installed packages and over 7,500 additional open source packages that can be
installed by PyPI, as well as by the conda package and by the virtual environment manager. It also includes a desktop GUI, Anaconda Navigator, which allows you to launch applications and easily manage conda packages, environments and channels without using command line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository. It is available for Windows, macOS and Linux operating systems.

2.2 Python libraries used

This section lists and briefly explains all the most important Python libraries that have been used for the development of the framework.

2.2.1 Pandas

Pandas [6] is a Python language library useful for processing and analyzing data, especially when you have to work with data in tabular format and heterogeneous data, that is, data presented as a matrix in which the values contained in the columns can belong to heterogeneous groups. It is equally valid when the data to be processed is proposed in the form of a time series.

2.2.2 NumPy

NumPy [7] is an open source library written in Python that offers support for large matrices and multidimensional arrays by providing a set of high-level mathematical functions to be able to operate efficiently on these structures
data.

2.2.3 Scikit-learn

Scikit-learn [8] is an open source library for the Python object language that provides regression, classification and clustering algorithms and other useful functions for machine learning.
Uses NumPy for linear algebra and high-performance array operations.

2.2.4 NetworkX

NetworkX [9] is a python package used for the study, creation and manipulation of the structure, dynamics and functions of complex networks.
Provides data structures for graphs and multi-graphs, standard algorithms for them
processing and analysis functions.

2.2.5 PyUnicorn

Pyunicorn [10] is an object oriented python package for advanced modeling and analysis of complex networks. It provides various functions for calculating standard and non-graph metrics. It also allows you to easily build networks from uni and multivariate time series and from data
events.

2.2.6 Matplotlib

Matplotlib [11] is a library created to represent data in the form of
graphics defined specifically for the Python language and the NumPy library.

2.2.7 Pytorch

Pytorch [12] consists of an external language module, particularly powerful, which provides valid support for the GPU and with various functions dedicated to the field of machine learning, deep learning and Natural Language Processing.

2.3 Algorithms and metrics

This section, on the other hand, specifically describes the algorithms and metrics that have been used in the process of forecasting the price trend of equities.

2.3.1 Visibility Graph Algorithm

The Visibility Graph (VG) algorithm is an algorithm used for mapping single time series in graphs. It turns out to be the most suitable when talking about financial time series because it is not influenced by algorithmic parameters and maps the time series in graphs without scale [13].
It is then used to map the raw data of market prices into time series graphs, that is, price graphs.
Thanks to the existence of explicit links between distant nodes in the converted graphs, the

Long-term dependencies in time series graphs can be captured directly through associations between time points. Furthermore, identifying the prominent content of the chaotic time series, the weights of nodes of the graph provide further knowledge for the learning of temporal attention in order to face the problem of the chaotic property of financial time series.

2.3.2 Collective Influence Algorithm

The collective influence algorithm aims to identify the number mi number of influential nodes [14].
The concept of influence is closely related to the concept of integrity of the net. The most influential nodes in a complex network form the minimal set la removal of which would dismantle the network into many disconnected and unconnected components extended. The extent of this fragmentation is the size of the cluster plus large of nodes, called the giant component G of the network.
Removing nodes with high collective influence is more effective at fragmenting and destroying a network than nodes selected by other methods.
The collective influence of a node in the network is given by the product of its degree reduced and the total reduced degree of all nodes at a distance d from it.

2.3.3 Graph embeddings with Struc2Vec

Struc2Vec is a graph embedding algorithm that aims to learn the structural properties of graphs as regards the associations between them time points [15].
According to the structural similarities based on the k-hop neighborhoods from the nodes, struc2vec is able to learn a vector representation that captures roles structural elements of nodes within a graph. Specifically, the method provides the execution of numerous random walks on the graph starting from each node.
It is based on two properties considered fundamental for a success approach toilet:

- The distance between the latent representation of the nodes should be for closely related to their structural similarity. Therefore, two nodes

which have identical local network structures should have the same latent representation, while nodes with different structural identities should be distant;

- Latent representation should not depend on any node or link, including node labels. Therefore, structurally similar nodes should have a near latent representation, independent of node attributes and links in their vicinity. The structural identity of the nodes must be independent of its “position” in the network.

From these definitions this algorithm has been developed which is basically based on four steps:

1. First of all, the structural similarity between each pair of vertices of the graph is determined for the different dimensions of the neighborhood. By doing this, a hierarchy is created in the measure of the structural similarity between the nodes, providing more information to evaluate the structural similarity at each level of the hierarchy.

2. Subsequently a weighted multilayer graph is constructed in which all the nodes of the network are present in each level and each level corresponds to a level of the hierarchy in the measurement of structural similarity. Furthermore, the weights of the arches between each pair of nodes within each layer are inversely proportional to their structural similarity.

3. The context for each node is created from the multilayer graph. In particular, a distorted random walk on the multilayer graph is used to generate sequences of nodes. These sequences are likely to include structurally more similar knots.

4. Finally, a technique is applied to learn the latent representation from a context given by a sequence of nodes, such as skip-gram

By incorporating price graphs into vector representations, struc2vec further enriches time point information by considering global dependencies in representations and improves the description of raw time series.

Long-term dependencies between time points can be captured thanks to the struc2vec feature in measuring structural similarity. The existing literature on complex networks and graph embedding implies that structural information extracted from time series graphs is capable of addressing the issues of long-term dependencies and chaotic property.

2.3.4 Local clustering coefficient

The local clustering coefficient of a node in a graph is a measure of how much its neighbors tend to form a clique (or a complete graph). A graph G = {N, L} formally consists of a set of nodes N and a set of edges L.

We define the arc lij that which connects the node ni with the node nj .
We also define the set Ni as the set of neighbors di ni of the nodes , that is the whole directly connected to the node ni . More formally

$$Ni = \{nj : \quad lij , lji \ddot{y} L^{2}\}$$

We define ki as the cardinality of Ni or the number of neighbors of ni .
The local clustering coefficient Ci is obtained from the ratio between the number of arcs actually present between the members of Ni and the number of arcs that could potentially exist between them.
Hence in an undirected graph, where the possible connections between the nodes of the set Ni is given by

$$ki (ki - 1) 2$$

, the local clustering coefficient for undirected graphs is calculated according to the following formula:

$$2 \cdot | \{ljk : nj ,, ljk1\ddot{y}L\} | nk \ddot{y} Ni ki (ki$$

2.3.5 Closeness centrality

Closeness centrality is a measure that focuses on the proximity of a node with respect to all other nodes in the set.
Nodes that occupy central positions according to the concept of proximity (close ness) can be very effective in transmitting information to other actors.
According to Sabidussi (Austrian mathematician) to quantify the concept of centrality as proximity, a node is said to be central in a network if the measurement of the number of steps that divide it from the other nodes is minimal. It is then calculated as the reciprocal of the sum of the length of the shortest paths between the node and all the other nodes of the graph. More formally:

$$C(u) = \frac{n - 1}{\sum_{v=1}^{n} d(u, v)}$$

The centrality is inversely proportional to the distance, since the shorter the distance between a node and the other nodes, the greater its centrality.
The maximum value that this index can assume is (g - 1) ÿ1 , when the node is adjacent to all other nodes. The minimum value is reached in its limit when one or more nodes are not reachable, i.e. we are in a disconnected graph.

2.3.6 Betweennes centrality

Betweenness centrality or centrality by interposition is a metric used to measure how important a node is by counting the number of shortest paths to which it belongs. For example, if the geodesic between nodes n2 and n3 is n2, n1, n4, n3, i.e. the shortest distance connecting these two actors must pass through nodes n1 and n4, we can say that the two actors in the middle have control over the interaction between n2 and n3. Hence, more correctly it can be said that the knots in the middle have a greater interpersonal influence on the others.

When between two nodes there are several geodesics connecting them, then the probability that one is used for communication is

$$\frac{1}{g_{jk}}$$

where gjk is the number of geodesics connecting ja k.
If, on the other hand, we define gjk (ni) as the number of geodesics that connect jake passing through i, and we establish that the probability of choosing one communication over another is the same, then the probability that a actor i is involved in this communication between j and k is given by

$$\frac{gjk\ (ni)}{gjk}$$

Hence the centrality of the node ni as an interposition is given by:

$$CB\ (ni) = X\ \underset{j<k}{gjk}\ \frac{gjk\ (ni)}{}$$

2.4 Structure and functioning of the framework

As previously pointed out, this paper intends to present
a new way to predict stock price trends.
To overcome the shortcomings of the financial time series forecasting methods previously introduced in the state of the art, a module based on complex network methods and the incorporation of graphs is introduced, to extract structural information from these, while trend predictions actions are obtained from different layers based on attention and from one layer of
fully connected classification.
Figure 2.1 illustrates the proposed structure, which is composed of two modules.

- Time series embedding module, the first module is a time series embedding module, which takes raw market price data as input, transforms them into graphs and extracts structural information, referring to the associations between time points and knot weights;
- Prediction module, the second module instead, is a module of predictions based on deep learning, where stock representations are learned from structural information by incorporating timelines for predicting stock trends.

Both modules are described more specifically below.

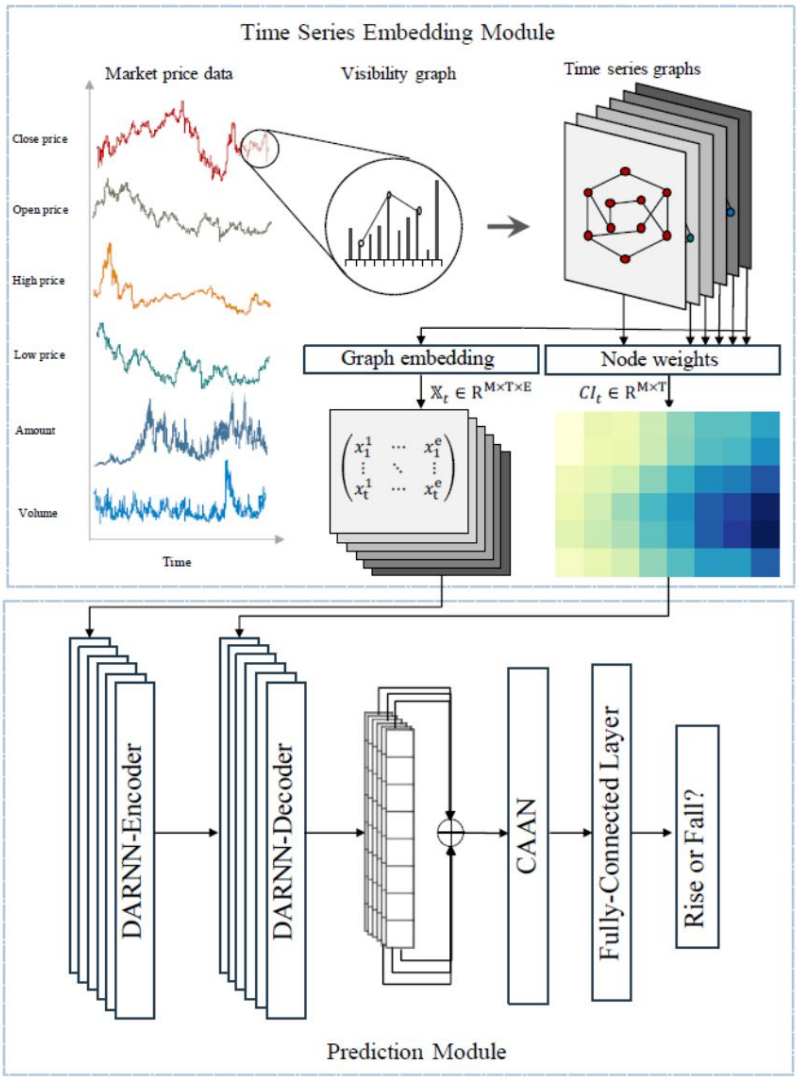

Figure 2.1: Framework structure [1]

2.4.1 Time series embedding module

The work carried out within this module consists at first in converting the raw data of market prices into time series graphs by measuring the weights of the nodes of the graph, while subsequently, for each of these, we go to explore the properties topological.

2.4.1.1 Time series graph

Generally there are three main approaches of complex networks to map single time series in graphs, but among these the most suitable and used when talking about financial time series is the VG algorithm since it is not influenced by algorithmic parameters and maps time series into graphs without
stairs.

Let us consider a series of stock prices pt with a length of T at time t. Each vertex in the converted graph corresponds to a data point in the original series. A link is established between the two points based on the principle that if the two data points can see each other in the histogram of the corresponding time series then they must be linked; Put differently, two time points in series can be connected by a straight line when there are no intermediate data heights that intersect this "line of sight".

More formally, a visibility graph can be created from a time series under the following visibility specification: given two points (ti, pi) and (tj , pj) where pi , pj > 0 in a time series, there exists a line of sight connecting the two points in the converted graph if and only if all the data points (tk, pk), such that they satisfy: ti < tk < tj ,

$$< pi + tj - ti \frac{tk - ti}{\quad}(pj - pi). pk \qquad (2.1)$$

For graphs converted from stock prices, each vertex represents the daily price under real trading circumstances. We can see the arcs of the graph as signs of the absence of abrupt price changes during the period between two time points.

Furthermore, on the basis of equation 2.1, we can deduce that the shortest path between any two time points in the converted graph is significantly shorter than their original time interval, which indicates that the series graphs temporal are suitable for capturing long-term dependencies because the information embedded in the initial prices can be obtained earlier from subsequent points without long-range transfers and without the information
vanish.

The VGs created are connected graphs and the construction process is invariant even if a series of basic transformations are performed on the original series, such as vertical and horizontal translations.

An example of the VG algorithm can be seen in figure 2.2, where we present a converted graph and the original time series with 20 data points. As you can see, time points are linked by links as long as they can see each other
each other.

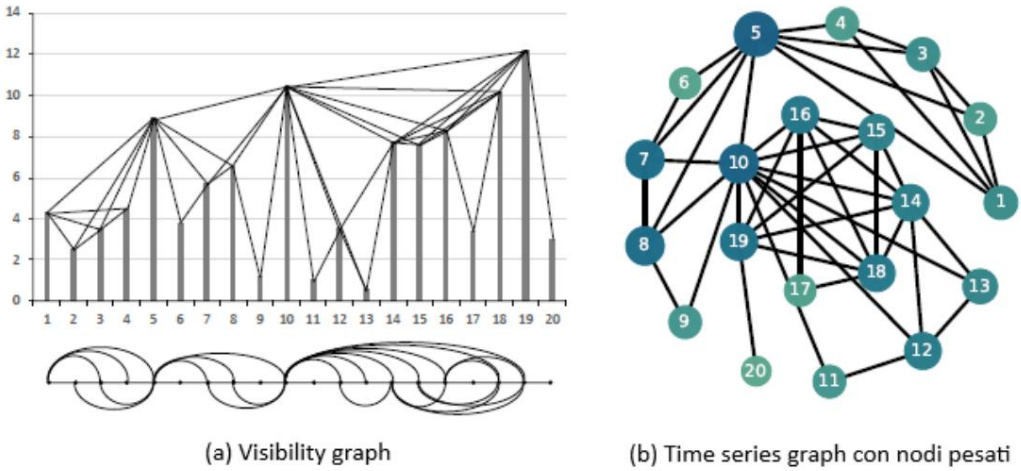

Figure 2.2: Example of a time series containing 20 time points and the graph derived from the VG2 algorithm

Subsequently, after having obtained the structure of the graph from the real time series, the weight of each node is measured to establish the chaotic property.

Traditionally in a complex network there are different metrics used to identify vital nodes, such as their degrees, k-cores and clu ster coefficients, but in this framework it was decided to use the collective influence algorithm since it has a low computational consumption and excellent performance in defining the influence of the node as a collective and not a local attribute. Such as

previously defined, in the collective influence algorithm a sphere of radius l (normally less than the diameter of the graph) is defined around

a node vi and its influence is calculated according to the equation:

$$CIl(vi) = (di - 1) \times \sum_{jÿÿBall(i,l)} (dj - 1) \quad (2.2)$$

where di is the degree of the node vi and l is a non-negative integer given according to the diameter of the graph and ÿBall (i, j) is the boundary of the sphere.
In figure 2.2 (b) there is an example of the time series graph with weighted nodes, where the weight corresponds to the collective influence index. Nodes with heavier weights are marked with deeper colors and larger sizes (example node 5).

In this phase, with the vector Pt we denote the historical state of an action at time t, where Pt ÿ R6 × T e consists of the raw data of the market price (the closing price, the high price, the low price, the opening price, quantity and volume), and T is the length of the lookback window of t.
Through the visibility graph algorithm, the converted graphs are obtained for each type of price (close, open, high, low, amount, volume) Gt , where is it

$$Gt = [G_t^{C.} \text{ OR } G_t^{H.} ; G_t^{L} \text{ TO } G_t^{V.}]$$

and each graph has T nodes. Furthermore, before extracting the structural information, we measure the weights of the CIt nodes of the graphs converted through the algorithm CI, where CIt ÿ R6 × T.

2.4.1.2 Graph Embedding

Once the graphs have been created, they must be transformed into vector representations and to maintain the structural properties regarding the associations between their time points, the struc2vec method is used which aims to learn a mapping

$$g: v ÿ V 7 ÿ R | V | \times d$$

.

This embedding method is designed to learn a representation of

ne which allows the estimation of the possibility that a node u shows itself together with other nodes in the sub-window of a short random walk:

$$\max_{g} \sum_{u \ddot{y} V} \log P\, r\,(N\,(u) \mid g\,(u)), \tag{2.3}$$

where N (u) represents the nodes close to node u. The probability that a vertex v appears together with u is calculated using a softmax function:

$$P\, r\,(v \mid u) = \frac{\exp\,(g\,(v)\, g\,(u))}{P._{vt \ddot{y} V} \exp\,(g\,(vi)\, g\,(u)).} \tag{2.4}$$

Furthermore, to measure the structural similarities between the nodes, struc2vec generates a set of assistant graphs weighted gk with k = 1, 2 ,. . . , k ÿ that derive from the captured graph to capture the structural similarities between the neighborhoods of the k-hop nodes, where k * is the diameter of the original graph. More concretely, each assistant graph is a complete indirect weighted graph.

Given Rk (v) defined as the ordered degree sequence of nodes which are at a distance k jumps from v, and wk (v, u), which measures the weights of the arcs in an assistant graph gk the weight of an arc is recursively defined as

$$wk\,(v \mid u) = wk\, \ddot{y}\, 1\,(v, u) + d\,(Rk\,(v), Rk\,(u)), \tag{2.5}$$

where w0 (v, u) = 0 and d (Rk (v), Rk (u)) measures the difference between Rk (v) and Rk (u). Struc2vec can generate sequences of vertices based on the weights of the edges in these weighted assistant graphs. Then, to maximize the probability that neighboring nodes in the local area appear together with the central node, a neural network using the skip gram architecture is trained.

The output of the hidden layer of the trained neural network is obtained as the embedding of the nodes. By incorporating price graphs into vector representations, struc2vec further enriches time point information by considering global dependencies in representations and improves the description of raw time series. Long-term dependencies between time points can be captured thanks to the feature

of struc2vec in the measurement of structural similarity.

At the end of this step, with the converted price graphs Gt tensor , we get the representations Xt for all graphs through struc2vec at time t, where

$$Xt = [X\,t\,;\,G.\,C.\,t\,;\,X\,t\,;\,X\,t\,;\,X\,t\,;\,X\,H.\,t\,;\,X\,t\,;\,X\,L\,t\,;\,X\,TO\,t\,;\,X\,V.\,t\,],$$

with Xt ÿ RT × E and E is the size of the embedding.

2.4.2 Prediction module

This paragraph describes the structure of the module prediction and its functioning which can be divided into two macro parts.
In the first part we have a series of artificial intelligence architectures, called DARNN [16] (Dual-stage Attention-based Recurrent Neural Net work), which deal with automatically extracting the representations of the input series by incorporating temporal information and structural information (the temporal information is obtained by organizing the model inputs in the order of the temporal sequences). Specifically, we have six DARNNs, one for each type of price, with weighted nodes for the temporal attention network.
The output of this model is a representation of the stock rt obtained by combining the output of the six DARNNs for each stock at time t.
In the second part, however, we have a cross-asset attention network, called CAAN, which is used to describe the relationships between the various securities.
Finally, the classification results are returned by a fully connected layer.
In figure 2.3 you can see the flow from the output to the input for each step of the module.

2.4.2.1 DARNNs model

As mentioned previously, the first part deals with learning the structural information through the use of six DARNNs that incorporate weighted nodes. The choice to use this type of architecture was made not only for its ability to select crucial variables and time points, but also for its excellent performance in predicting time series with respect to recurrent neural networks, simple LSTMs or convolutional networks.

Each DARNN is an encoder-decoder network and has the same learning process.

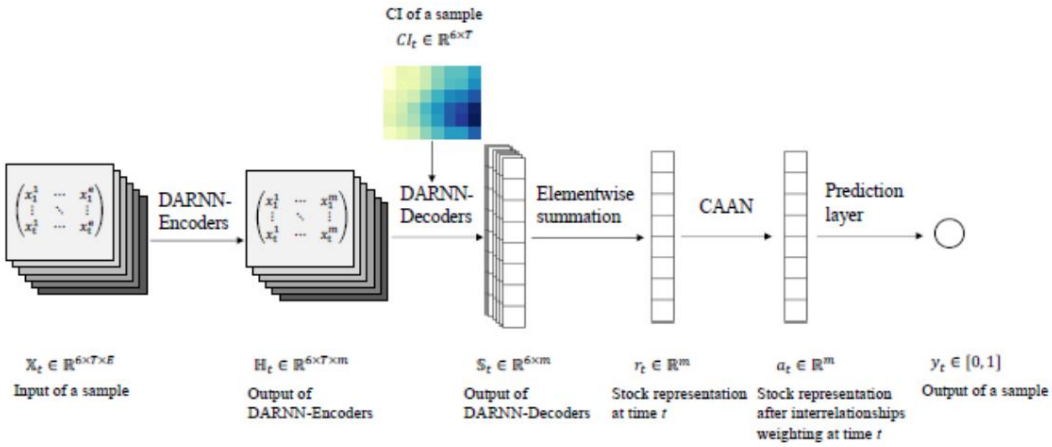

Figure 2.3: Computational flow of the prediction module 3

chin, but with different inputs. The encoder is basically an LSTM
which learns the input sequences as a hidden representation that considers the attention
to be given to the input data. For forecasting series
temporal, given X = (x1, x2, ..., xT) ÿ RT × E indicating the input sequence,
where xt ÿ RE and E is the size of the embedding graph, the encoder comes
used to recursively learn a hidden vector from X:

$$ht = LSTM (ht ÿ 1, xt), t ÿ [1, T], \quad (2.6)$$

where ht ÿ Rm is the hidden vector encoded by LSTM at the time step t
and m is the size of each hidden vector.
Since attention mechanisms are widely used in deep neural networks, a DARNN integrates
input attention in the encoding phase.
to automatically select the appropriate input series to give more and more phases to the
information characteristics obtained from the chaotic series.
Date $x^k = (x^k_{1,}\ xk_{2,\ ...,}\ xk_{T.})^{T.}$ ÿ RT as the k-th series in entry, attention
to be placed is calculated through a deterministic attention model, which
adopts the hidden state ht ÿ 1 and the state of the cell st ÿ 1 of the LSTM unit del
encoder:

$$c^k_t = v^{\ddot{y}}_c \tanh (Wc [ht ÿ 1; st ÿ 1] + Ucx^k) \quad (2.7)$$

And

$$\text{to}_t^k = \frac{\exp(c_t^k)}{\sum_{i=1}^{n} \exp(c_t^i)}, \tag{2.8}$$

where vc ÿ RT , Wc ÿ RT × 2m and Uc ÿ RT × T are learnable parameters.

At the time step t, the relative importance of the k-th series of inputs is estimated k from the coefficient a t k which is nothing but the coefficient c t normalized via a softmax. This function is used to highlight larger values and hide those that are significantly smaller than the value maximum.

The update of the input series with attention weights can be formulated as

$$\tilde{x}_t = (a_t^{1,1} x_{t,1}, a_t^{2,2} x_{t,2}, \ldots, a_t^{n,n} x_{t,n}) \tag{2.9}$$

Subsequently we will have that the update of the hidden state at time t will be

$$h_t = \text{LSTM}(h_{t-1}, \tilde{x}_t) \tag{2.10}$$

In this way it is possible to make an adaptive selection of the most decisive time series by calculating the attention weight each time and avoiding to consider all the input series in the same way.

The hidden features Ht = [HC HH HV] ÿ RG×T xm for the six prices can be obtained from the encoders of the DARNN architectures by passing the structural information Xt as input .

The decoder, on the other hand, incorporates a temporal attention to automatically select the appropriate hidden vectors of the encoder among all the time intervals, since these are able to capture the long-term dependencies within a time series. The temporal attention between the hidden states of the decoder is also calculated through a deterministic attention model, which adopts the state of the cell s $_{t-1}$ and the hidden state h_{t-1} both ÿ Rm, of the LSTM unit of the decoder with

$$d_t^i = v_d^T \tanh(W_d [h_{t-1}; s_{t-1}] + U_d h_i), \quad 1 \le i \le T \tag{2.11}$$

And

$$\ddot{y}\,t = \frac{\exp(d\,t)}{\sum_{j=1}^{T} \exp(d\,t)}, \tag{2.12}$$

where vd ÿ Rm, Wd ÿ Rm × 2m and Ud ÿ Rm × m are learnable parameters and m is the dimension of each hidden state.

As in the case of the encoder, ÿ the importance t is a weight of temporal attention that measures regarding the i-th hidden state of the encoder, thus re-balancing the information at each time point to capture long-term dependencies and filter out chaotic time points. Here, too, a softmax normalization is applied to the attention weights d Once the temporal attention is obtained, all the hidden states of the encoder h1, h2, ..., hT are added in a weighted way as a context vector t .

et :

$$et = \sum_{i=1}^{T.} \ddot{y}\,thi\,. \tag{2.13}$$

The context vector et varies at each time step.

In this basic decoder, the time weights modeled by equations 2.11 and 2.12 are learned directly from the encoder's hidden representation, but we can use a priori knowledge to improve the learning process of time weights. Given the node weights of the price graphs (i.e. the collective influence index), we can use them as the knowledge-based attention on time points to highlight information points and deal with the chaotic property of price series. Then, for each hidden state hi , we update the attention weight d t In the following way:

$$d\,t = v\,d\,\tanh(Wd\,[h\,t\,\ddot{y}\,1\,;\,s\,t\,\ddot{y}\,1\,] + Udh + wdCIi),\ 1\ \ddot{y}\ i\ \ddot{y}\ T \tag{2.14}$$

where wd is a learnable parameter and CIi denotes the weight of the node of time point i. By doing so, the relative importance levels of the time points in the price graphs are introduced as weights to improve the learning of attention and the regulation of the chaotic property. The nodes that obtain critical positions in the time series contribute more to the final representations of the sample and make more accurate predictions.

After re-weighing the temporal attention, the combination of the vector of with

et with the series of stock prices p1, p2, ..., pT ÿ1 is indicated updated text

such as:

$$p_{t\ddot{y}1} = w_p [p_{t\ddot{y}1}; e_{t\ddot{y}1}] + b_p, \quad (2.15)$$

where [pt ÿ et ÿ 1] ÿ Rm + 1 is a concatenation of the history of price and del

1; context vector while wp ÿ Rm + 1 and bp ÿ R are learnable parameters.

Consequently, the new pt ÿ 1 is used to update the function nasco = LSTM (h ÿ

sta of the LSTM unit in the decoder as h ÿ t t ÿ 1 , pt ÿ 1). The output of the

decoder of the DARNN architecture can therefore be formulated:

$$S = w [h_T ; e_T] + b, \quad (2.16)$$

where S ÿ Rm and [h ÿ eT] ÿ Rm + m is a concatenation of the characteristic na

T ; offset h 'e of the updated context vector w ÿ Rm × and at the last step of LSTM.

states of the decoder! 2m p. The parameters of the hidden b ÿ Rm map the concatenation to the dimension

With the hidden characteristics Ht learned from the DARNN encoder, the stock representations

St = [S t are computed through different DARNNs, where $ for six price series and the m decoder

size of each DARNN. Subsequently with an elemental sum layer, the six representations are

merged into one

$$r_t = \sum_{l \ddot{y} \{C, O, H, L, A, V\}} S_t^l \ddot{y} R^m \quad (2.17)$$

as the representation of the action at time t.

2.4.2.2 CAAN model

Once the representations of the actions are acquired, they are passed on

to the CAAN model, which relies on self-attention to use the inter-relationships between equities.

In particular, on the basis of the representations given in output by the decoder, three vectors

are calculated:

- q = Wqri , as a query vector;

• k = Wkri , as a key carrier;

i • v = Wvri , as a value vector;

where Wq, Wk, Wv are learnable parameters.

The interrelationships between the stock i and the other stocks within a lot are calculated using the query vector qi of stock i to query the key vectors of the other stocks:

$$l_{i,j} = \frac{q_i K_j}{\sqrt{D_k}} \tag{2.18}$$

And

$$\alpha_{i,j} = \frac{\exp(l_{ij})}{\sum_{j=1}^{T} \exp(l_{ij})} \tag{2.19}$$

where Dk is a scaling parameter and l is the size of each batch of samples. Then we use the interrelations of the value vectors vj in a representation of the attention for each point under

$$a_i = \sum_{j=1}^{T} \alpha_{i,j} v_j \tag{2.20}$$

The forecast of the future price trend of security i is calculated by a final feed-forward layer in which a sigmoid function is applied:

$$y_b = \text{sigmoid}(W_{fc} a + b_{fc}) \tag{2.21}$$

where Wfc and bfc are respectively the linear parameter and the bias to be learned.

Figure 2.4 summarizes the training and optimization process
tation of the prediction module just described.

2.5 Chaotic property and collective influence

Within this paragraph we want to give a more in-depth explanation of why the use of collective influence, assigned as the weight of the node of a graph within the decoder of the DARNN network, can improve the

Algorithm 1 Optimization of our prediction module within a sample

Input: input sample $\mathbb{X} \in \mathbb{R}^{6\times T\times E}$, corresponding $CI \in \mathbb{R}^{6\times T}$, historical price series $P \in \mathbb{R}^{6\times T}$, labeled target y

Output: predicted label $\tilde{y}$ of input

Denote all parameters of the prediction model as $\mathbb{W}$;
Initialize the parameters $\mathbb{W}$;
for $epoch \in [1, \ldots, maxIteration]$ **do**
  $\mathbb{H} = \text{DARNN-Encoders}(\mathbb{X})$, where $\mathbb{H} \in \mathbb{R}^{6\times T\times m}$;
  $\mathbb{S} = \text{DARNN-Decoders}(\mathbb{H}, CI, P)$, where $\mathbb{S} \in \mathbb{R}^{6\times m}$;
  $r = \sum_{i=0}^{6} \mathbb{S}_i$, where $r \in \mathbb{R}^m$;
  $a = \text{CAAN}(r)$, where $a \in \mathbb{R}$;
  $\tilde{y} = f_{prediction}(a)$, where $\tilde{y} \in \mathbb{R}$;
  // computing loss on a batch sample;
  $\mathcal{L} = \text{LossFunction}(y, \tilde{y})$;
  Update hidden parameters $\mathbb{W}$ with gradient decent $(\mathcal{L}|\mathbb{W})$;
end for

Figure 2.4: Prediction module process 4

chaotic property problem affecting the trend in the prices of securities
equity.

Taking into consideration the example in figure 2.5 (a), it can be seen that al
day 15 (highlighted in yellow) there is a sharp change in price which inside the neural
network could negatively influence the update
of the weight of attention giving it a considerable level of importance than in reality
does not possess. In fact, it can be seen from the graph constructed from that series
temporal, that the node that has the deepest size and color, and therefore
a higher collective influence value is day 18 and not day 15. This
demonstrates that, as can be seen in figure 2.5 (c), the model without regulation of
collective influence follows the variation of the price making itself
attract from the day that shows the most abrupt variation, as he mistakenly considers
it more informative than the others. If, on the other hand, collective influence is calculated
a possible solution to the problem can be obtained because, as can be seen from
graph 2.5 (d), it is possible to recognize which time points are the most
information for trend forecasting, removing the importance of day 15 e
giving it to days 18, 7 and 8. In the experiments conducted in this thesis, one wants

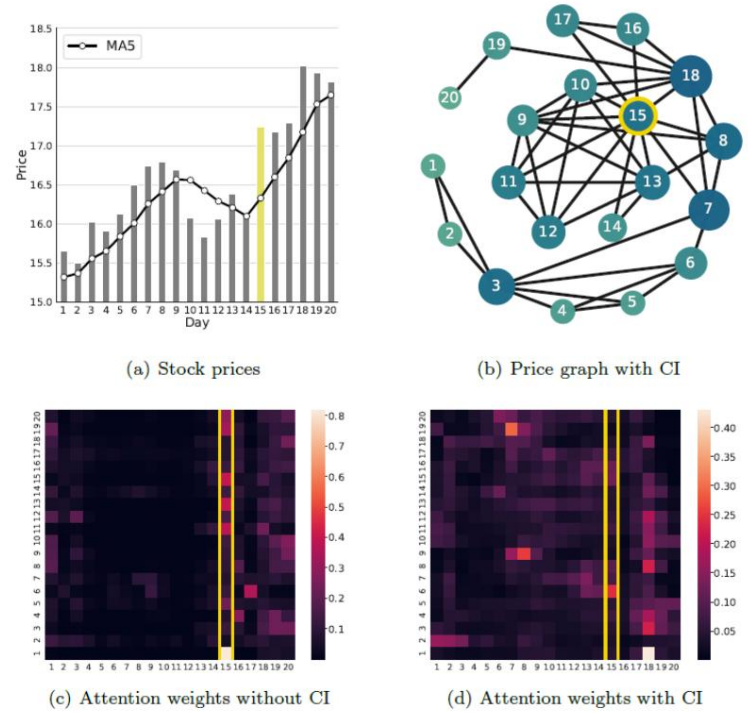

Figure 2.5: Example of how CI helps to capture multiple time points informative [5]

check if calculating other metrics on graphs can also help or improve the forecast of the stock trend.

Chapter 3

Development and implementation

Within the third chapter we briefly present what are the experiments that will be carried out using this structure and then yes
they look at the code level, some of the most fundamental steps for the process prediction, with the aim of understanding how a certain task was developed.

3.1 Changes made

In the previous chapter the operation of the framework object of this paper was described step by step in detail. In this paragraph instead, we want to introduce the changes made to the model to experiment
any corrections that perhaps could improve the accuracy of predicting the price development of a stock.
The idea always remains that of using complex networks and artificial intelligence, but going to calculate different metrics on the graphs to see
how the prediction module behaves and what results are obtained.
The dataset that will be used will not match the one containing the actions Chinese, but for reasons of size and computing devices available in
primis, and of personal interest then, will be constituted by the financial shares of nine publicly traded healthcare companies worldwide.
Specifically, it was decided to calculate three new metrics to use
to give weight to the graph nodes. The first two are very similar in meaning to collective influence, because they are two centrality indices that measure

how important a node is (central) within a network and are respectively the betweennes centrality and closeness centrality. The third instead corresponds to the local clustering coefficient which measures how much the nodes of the graph tend to form a clique with its neighbors, useful to understand how much the neighbors of a node tend to be dependent on each other.
Therefore, five different models will be compared:

1. the first remains the classic framework that has been proposed, which uses collective influence to give weight to the nodes;
2. in the second the weight of each node will be represented by its betweennes centrality;
3. in the third, the weight of the node will correspond to closeness centrality;
4. in the fourth, instead, the clustering coefficient will be calculated for each actor in the network;
5. in the fifth, the sum between the clustering coefficient of each node and the centrality index which produced the best accuracy in the first three models will be used as previous knowledge.

For the first four models what changes during the training is the value that is passed to the DARNN decoder, if in the first the value of the collective influence was passed, in the second, in the third and in the fourth they will be passed respectively the betweennes centrality, closeness centrality and the clustering coefficient.

In the fifth model, instead, a change is made to the DARNN decoder model. If before there was a single level in the network that took the value of the metric as input, now a linear level is added to take the two weights as input.

The outputs of each hidden state will then be added at the point where the attention weight update is performed.

3.2 Creation of visibility graphs

The creation of the visibility graphs from the financial time series comes produced within a file called price graph.py that is executed as a first step.

```
def make_visibility_graph (input_):
    file_, PI, T = input_
    vg_dir = os.path.join ('../ VG', PI)
    vg_file = os.path.join (vg_dir, '% s.pickle' % file _ [: - 4])

    df = pd.read_csv (os.path.join ('../ data', file_), index_col = 0)
    df.set_index (df.index.astype ('str'), inplace = True)
    vgs = {}
    for i in df.index:
        iloc = list (df.index) .index (i)
        if df.loc [: i,:]. shape [0] <T:
            continue
        time_series = df.iloc [iloc-T + 1: iloc + 1] [PI]
        if len (set (time_series.values)) == 1:
            continue
        net = timeseries.visibility_graph.VisibilityGraph (time_series.
            values)
        vgs [i] = net.adjacency
    with open (vg_file, 'wb') as fp:
        pickle.dump (vgs, fp)

if __name__ == '__main__':
    ...
```

As can be seen from the code shown, this function is called inside the main by passing it three parameters:

- file, the file containing the data relating to the price trend for a security financial;

- PI, the price to be taken into consideration between high, low, open, close, volume, amount / adjClose;
- T, the size of the lookback window, set by default to 20.

For each call, starting from January 1st 2018 and scrolling by one day at a time, the lookback window is set and the graph is created corresponding to the first day (line 16).
Subsequently for the time point in question it is saved in the array vgs the corresponding adjacency matrix, which will play a fundamental role in the continuation of the flow.
All adjacency matrices are then saved in files within the VG folder.

3.3 Calculation of metrics

The computation of the metric for all nodes of the graphs is the second fundamental step within the flow, because it allows to acquire that knowledge previous necessary for updating the weight of attention within the DARNN network decoder.

```
1 def vg_ci (_input):
2       file_, PI, T = _input
3       graph_file = os.path.join ('../ VG', PI, file_)
4       with open (graph_file, 'rb') as fp:
5             vgs = pickle.load (fp)
6       cis = {}
7       for d, adj in vgs.items ():
8             labels = np.array ([str (i) for i in range (20)])
9             G = nx.Graph ()
10            for i in range (T):
11                  vg_adjs = labels [np.where (adj [i] == 1)]
12                  edges = list (zip (* [[labels [i]] * len (vg_adjs), vg_adjs]))
13                  G.add_edges_from (edges)
```

```
14              cis [d] = collective_influence (G)
15          ci_file = os.path.join ('../ CI', PI, '% s.json' % file _ [: - 7])
16          with open (ci_file, 'w') as fp:
17              json.dump (cis, fp)
18
19  if __name__ == '__main__':
20          ...
```

Again the vg ci.py function called inside the main takes
in input the same parameters of the function seen previously.
From the visibility graphs created, for each day the respective adjacency matrix is taken, the graph is created and in line 14 the influence is calculated collective for each node of the input graph.
Finally the results are saved in files inside a folder called

THERE.

To calculate the metrics of the new experiments, the same function is used by changing only in line 14 the call to the index to be calculated.

3.4 Graph embeddings

The graph embeddings phase is another essential step for this fra mework because it identifies the moment in which we try to capture the structural information of all the graphs previously created by relying on Struc2Vec framework.

To do this, simply execute the price embeddings.py file which contains the function shown in the code below.

```
1 def struc2vec_embedding (input_):
2          file_, em_size, PI = input_
3          vg_file = os.path.join ('../ VG', PI, file_)
4          with open (vg_file, 'rb') as fp:
5              vgs = pickle.load (fp)
6          em_dir = os.path.join ('../ Struc2vec' , if not                PI)
7          os.path.exists (em_dir):
8              os.makedirs (em_dir)
```

```
9       ems = {}
10      for d, adj in vgs.items ():
11              labels = np.array ([str (i) for i in range (20)])
12              G = nx.Graph ()
13              for i in range (len (labels)):
14                      adj_nodes = labels [np.where (adj [i] == 1)]
15                      edges = list (zip (* [[labels [i]] * len (adj_nodes), adj_nodes])
                                )
16                      G.add_edges_from (edges)
17              model = Struc2Vec (G, walk_length = 10, num_walks = 80, workers = 40,
                          verbose = 40, stay_prob = 0.3, opt1_reduce_len = True,
                        opt2_reduce_sim_calc = True, opt3_num_layers = None, temp_path
                        = '. / temp_struc2vec_% s /' % PI, reuse = False) #init model
18              model.train (embed_size = em_size, window_size = 3, workers = 40,
                        iter = 5) #train model
19              embeddings = model.get_embeddings () # get embedding vectors
20              ems [d] = {k: v.tolist () for k, v in embeddings.items ()}
21      with open (os.path.join (em_dir, '% s.json' % file _ [: - 7]), 'w') as
                fp:
22              json.dump (ems, fp)
23
24 if __name__ == '__main__':
25          ...
```

Also in this case the parameters of the function are the same as the previous ones, with the only difference for the size of the embedding which is set default to 32. Each graph created by the adjacency matrix is passed to Struc2Vec model in line 17, setting among the most important parameters a length of walks equal to 10 and a total number of walks random equal to 80. In the next line the train is then launched, saving the embeddings produced inside the ems array in line 20. Everything is then saved in json files for each share in the Struc2Vec folder.

3.5 DARNN model

With this paragraph we move inside the prediction module an showing in detail how the DARNN network is structured.
As explained in section 2.4.2, six of these architectures are created attention-based artificial intelligence, each of which is divided into two stages: encoder and decoder.
From the part of the code shown below it is possible to know which are the levels that make up the encoder of the DARNN network, where specifically we have:

- 3 linear levels attn;
- a level represented by the Long short-term memory (LSTM);
- a drop out level.

```
class AttnEncoder (nn.Module):
    def __init __ (self, input_size, hidden_size, time_step, drop_ratio
        ):
        super (AttnEncoder, self) .__ init __ ()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.T = time_step

        self.lstm = nn.LSTM (input_size = input_size, hidden_size =
            hidden_size, num_layers = 1)
        self.attn1 = nn.Linear (in_features = 2 * hidden_size,
            out_features = self.T)
        self.attn2 = nn.Linear (in_features = self.T, out_features = self.T
            )
        self.tanh = nn.Tanh ()
        self.attn3 = nn.Linear (in_features = self.T, out_features = 1)

        self.drop_ratio = drop_ratio
        self.drop = nn.Dropout (p = drop_ratio / 100.)
```

```
16
17      def forward (self, driving_x):
18          # driving_x: batch_size, T, input_size
19          driving_x = self.drop (driving_x)
20          batch_size = driving_x.size (0)
21          # batch_size * time_step * hidden_size
22          code = self.init_variable (batch_size, self.T, self.hidden_size
                )
23          # initialize hidden state (output)
24          h = self.init_variable (1, batch_size, self.hidden_size)
25          # initialize cell state
26          s = self.init_variable (1, batch_size, self.hidden_size)
27          for t in range (self.T):
28              # batch_size * input_size * (2 * hidden_size)
29              x = torch.cat ((self.embedding_hidden (h), self.
                    embedding_hidden (s)), 2)
30              # batch_size * input_size * T
31              z1 = self.attn1 (x)
32              # batch_size * input_size * T
33              z2 = self.attn2 (driving_x.permute (0, 2, 1))
34              # batch_size * input_size * T
35              x = z1 + z2
36              # batch_size * input_size * 1
37              z3 = self.attn3 (self.tanh (x))
38              if batch_size> 1:
39                  # batch_size * input_size
40                  attn_w = F.softmax (z3.view (batch_size, self.input_size
                        ), dim = 1)
41              else:
42                  attn_w = self.init_variable (batch_size, self.
                        input_size) + 1
43              # batch_size * input_size (element dot multi)
44              weighted_x = torch.mul (attn_w, driving_x [:, t,:])
45              _,  states = self.lstm (weighted_x.unsqueeze (0), (h, s))
46              h = states [0] # 1, batch_size, hidden_size
```

```
47                  s = states [1]
48
49                  # encoding result
50                  # batch_size * time_step * encoder_hidden_size
51                  code [:, t,:] = h
52
53              if self.drop_ratio> 0:
54                  code = self.drop (code)
55
56              return code
```

The fordward process defined in the second function specifies the path that
the input data at time t in the form of tensors perform between the various levels of the encoder.
In the case in question, they are first passed to the dropout level which,
based on the given percentage, it randomly sets some values to 0 and then
at the three linear levels, where in the last one the activation function is calculated
of the hyperbolic tangent, the result of which will serve to establish the attention weight
attn w to be passed along with the incoming data to the LSTM (line 46).
The result of this level is first stored in the variables seh, che
respectively represent the cell and the hidden state at time t used in
computation of the next iteration for the data at time t + 1, and then it is passed
to the decoder.

The structure of the decoder, on the other hand, is slightly more complex because it is composed
from 6 linear levels and a level containing the LSTM. Of these six linear levels,
one corresponds to the output level and one takes as input the collective influence values
relative to the series given in input at time t. In the case in
which (as will be seen in the fifth experiment carried out) they should be taken into
consider two metrics for updating the attention weight then
an additional linear level will be added which is commented out on line 12.

```
1 class AttnDecoder (nn.Module):
2       def __init __ (self, code_hidden_size, hidden_size, time_step):
3             super (AttnDecoder, self) .__ init __ ()
4             self.code_hidden_size = code_hidden_size
```

```
        self.hidden_size = hidden_size
        self.T = time_step

        self.attn1 = nn.Linear (in_features = 2 * hidden_size,
            out_features = code_hidden_size)
        self.attn2 = nn.Linear (in_features = code_hidden_size,
            out_features = code_hidden_size)

        self.attn_ci = nn.Linear (in_features = time_step, out_features =
            code_hidden_size)
        # self.attn_cc = nn.Linear (in_features = time_step, out_features =
            code_hidden_size)

        self.tanh = nn.Tanh ()
        self.attn3 = nn.Linear (in_features = code_hidden_size,
            out_features = 1)
        self.lstm = nn.LSTM (input_size = 1, hidden_size = self.hidden_size
            )
        self.tilde = nn.Linear (in_features = self.code_hidden_size + 1,
            out_features = 1)
        self.output = nn.Linear (in_features = code_hidden_size +
            hidden_size, out_features = hidden_size)

    def forward (self, h, y_seq, cis):
        # h: batch_size * time_step * layer1_hidden_size
        # y_seq: batch_size * time_step
        # cis: batch_size * time_step
        batch_size = h.size (0)
        d = self.init_variable (1, batch_size, self.hidden_size)
        s = self.init_variable (1, batch_size, self.hidden_size)
        ct = self.init_variable (batch_size, self.hidden_size)

        for t in range (self.T):
            # batch_size * time_step * (2 * decoder_hidden_size)
            x = torch.cat ((self.embedding_hidden (d), self.
```

```
                embedding_hidden (s)), 2)
            # batch_size * time_step * layer1_hidden_size
            z1 = self.attn1 (x)
            # batch_size * time_step * layer1_hidden_size
            z2 = self.attn2 (h)
            # b * T -> 1 * b * T -> b * T * T -> b * T *
                layer1_hidden_size
            zci = self.attn_ci (self.embedding_hidden (cis.unsqueeze (0))
                )
            #zcc = self.attn_dc (self.embedding_hidden (ccs.unsqueeze (0)
                ))

            x = z1 + z2 + zci # + zcc

            # batch_size * time_step * 1
            z3 = self.attn3 (self.tanh (x))

            if batch_size> 1:
                # batch_size * time_step
                beta_t = F.softmax (z3.view (batch_size, -1), dim = 1)
            else:
                beta_t = self.init_variable (batch_size, self.
                    code_hidden_size) + 1
            # batch_size * layer1_hidden_size
            ct = torch.bmm (beta_t.unsqueeze (1), h) .squeeze (1) # (b, 1,
                T) * (b, T, m)
            # batch_size * (1 + layer1_hidden_size)
            yc = torch.cat ((y_seq [:, t] .unsqueeze (1), ct), dim = 1)
            # batch_size * (1 + layer1_hidden_size)
            y_tilde = self.tilde (yc)
            _, states = self.lstm (y_tilde.unsqueeze (0), (d, s))
            d = states [0] # 1, batch_size, hidden_size
            s = states [1]

        # batch_size * (hidden_size + last_layer_hidden_size)
```

```
61          dt_ct = torch.cat ((d.squeeze (0), ct), dim = 1)
62
63          # batch_size * hidden_size
64          return self.output (dt_ct)
```

In this forward process, the "previous knowledge" given by the weights of the nodes of the price graphs is substantially taken into consideration.
it is added to the outputs of the linear levels attn 1 and attn 2, which they take respectively as input the concatenation of the LSTM result instantly t-1 and the encoder output (line 40), with the aim of giving a better contribution to the attention to be given to that data. In the fifth experiment the weights from add will be two.
Once the outputs for all six DARNNs are calculated, the results come chained into one that is passed to the CAAN network.

3.6 CAAN model

Finally, in this last paragraph the structure of the last network is presented neural of the prediction module, used to calculate and use the possible interrelationships between the various securities in the prediction of the financial trend.
The network is composed of three linear levels used to determine query vectors, key vectors and value vectors.

```
1 class SelfAttention (nn.Module):
2       def __init __ (self, last_hidden_size, hidden_size):
3           super (SelfAttention, self) .__ init __ ()
4           self.last_hidden_size = last_hidden_size
5           self.hidden_size = hidden_size
6
7           self.wq = nn.Linear (in_features = last_hidden_size, out_features
                = hidden_size, bias = False)
8           self.wk = nn.Linear (in_features = last_hidden_size, out_features
                = hidden_size, bias = False)
```

```
9           self.wv = nn.Linear (in_features = last_hidden_size, out_features
                = hidden_size, bias = False)
10
11      def forward (self, h):
12          # h: batch_size * last_hidden_size
13          q = self.wq (h)
14          k = self.wk (h)
15          v = self.wv (h)
16
17          dk = q.size (-1)
18          z = torch.mm (q, kt ()) / math.sqrt (dk) # (b, hidden_size) * (
                hidden_size, b) ==> (b, b)
19          beta = F.softmax (z, dim = 1)
20          st = torch.mm (beta, v) # (b, b) * (b, hidden_size) ==> (b,
                hidden_size)
21          return st
```

In line 19 the possible interrelationships between the stock i and the other stocks are calculated, and once the weight has been normalized using the softmax function, the beta value for weighting the vectors v values.

The final value of the prediction will simply be a linear level in which it comes applied the sigmoid activation function and whether the result will be greater than or equal to 0.5 then it will be classified as a rise otherwise as a fall.

Chapter 4

Experiments and evaluations

Within this last chapter the experiments carried out using the framework in question will be discussed. In the first part, the tests carried out by the authors of the original model for predicting price trends using the CSI-300 index dataset are presented, while the second part will illustrate the results of five different tests for: past input data ,
calculated metrics and neural network model.

4.1 Experiment with the CSI-300 dataset

Once the proposed framework was defined, the equity data that make up the CSI-300 index of the Chinese market were taken into consideration to evaluate the efficiency. Specifically, the data that was collected goes from 1 January 2010 to 31 December 2019 as it was decided to avoid taking into consideration the values relating to the pandemic shock of COVID-19.
For each day, six prices were captured as follows:

- high, the highest price reached on that day;
- low, the lowest price reached on that day;
- open, the opening price on that day;
- close, the closing price on that day;

- amount, the total amount of money that circulated for that asset (volume * price) on that day;
- volume, the number of contracts of an asset, which were traded in a day. The volume is higher when we have significant price fluctuations in the markets, which usually occur following major financial news, macroeconomic announcements, political elections.
  and so on.

In order to be able to give an efficiency value of the proposed structure, the same data have been the subject of inputs from other types of neural networks present in the state of the art, but which do not exploit the structural information extracted from the time series graphs. For this reason the results obtained were compared with the results obtained by the following models:

- LSTM, a basic LSTM used to predict future trends
  stock prices based on its historical data;
- DARNN, this is a two-stage recurrent neural network based on incoming attention and temporal attention respectively in the encoder and decoder stages. This model is a part of our framework;
- DARNN-SA, is a sort of extension of the classic DARNN as it simply inserts an additional layer of attention between the DARNN output and the prediction layer;
- CA-SFCN, this is a convolutional network that incorporates cross attention, used as a two-stage attention for temporal and variable dimensions;
- MFNN, a multi-filter deep learning model that integrates convolutional and recurrent neurons for the extraction of characteristics with respect to financial time series and stock prediction.
- ARMA, acronym for autoregressive moving average model, is a type of linear mathematical model that provides an output value instant by instant based on the previous input and output values;

- GARCH, acronym for Generalized Self-Regressive Conditional Heteroskedasticity, is a statistical model used in the analysis of time series data in which the variance error is considered to be autocorrelated in a serial way. GARCH models assume that the variance of the error term follows an autoregressive moving average process.

4.1.1 Dataset splitting

In the phase of defining the training set and the test set, it was decided to make a division based on the years, i.e. all the data ranging from 2010 to 2018 were assigned to the training set, while the tests were carried out on the entire year of 2019 which, however, has been divided into four quarters 2019 (S1), 2019 (S2), 2019 (S3), 2019 (S4).

During the training process, 30% of the train set is randomly selected and assigned to the validation set to exploit historical information.
For each stock, historical information is taken with a window size of 20 days to predict the next day's price trend. The objective of the experiment is defined as follows:

$$y = \begin{cases} 1, & pc_{t+1} > pc_t \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where c_t is the closing price of the stock at time t.
In the table below (figure 4.1 it is possible to quantify the number of samples that were used to perform the training and subsequently the test.

4.1.2 Hyperparameter tuning

The optimal hyperparameters were obtained through a grid-search al
in order to obtain the best performance for all models.
The dimensions of the hidden states have been iterated between 32, 64, 128 and 256 while the dimensions of the minibatches between 32, 128 and 256.
The learning rate has been set by default to 0.001 while as an optimizer

Dataset		#Sample	↓		↑	
			#Sample	(%)	#Sample	(%)
Train/Val	2010 - 2018	530,284	276,533	52.15	253,751	47.85
Test	2019(S1)	16,992	7,591	44.67	9,401	55.33
	2019(S2)	17,765	9,534	55.36	8,231	46.33
	2019(S3)	19,488	10,789	55.36	8,699	44.64
	2019(S4)	18,300	9,142	49.96	9,158	50.04

Figure 4.1: Composition of the training, validation and test set [1]

Adam was chosen.

The optimization of the hyperparameters was done thanks to the use of the library PyTorch, which allows you to automatically refine the parameters that can be learned through backpropagation, to implement our
forecast. For all guidelines the models were end-to-end trained from the raw odds data with a z-score normalization function
using the standard deviation and mean calculated for each sample. For
the parameters in struc2vec, the path length has been set to 10 and the
size of the window at 3. The classical approaches of the time series, that is ARMA and GARCH, have been optimized through Auto-Arima in Python.

4.1.3 Results

To evaluate the results obtained by the proposed framework the authors have compared the percentage of accuracy achieved, both in the training phase and
in the testing phase, with those obtained from the neural network structures present in the state of the art, naturally passing him the same set of samples.
Normally, models with greater representational power should obtain greater accuracy in the training phase than those who do not.
use, and in fact, as can be seen from the graph in figure 4.2, the proposed structure manages to reach the higher percentage, probably because some
Stock data properties are captured from structural information derived from price graphs.

Furthermore, at first glance, it is easy to notice the clear difference in learning between this framework and the classic LSTM, which lately was very used for data prediction, but which apparently fails on its own to be competitive. So from the results obtained in the training phase not

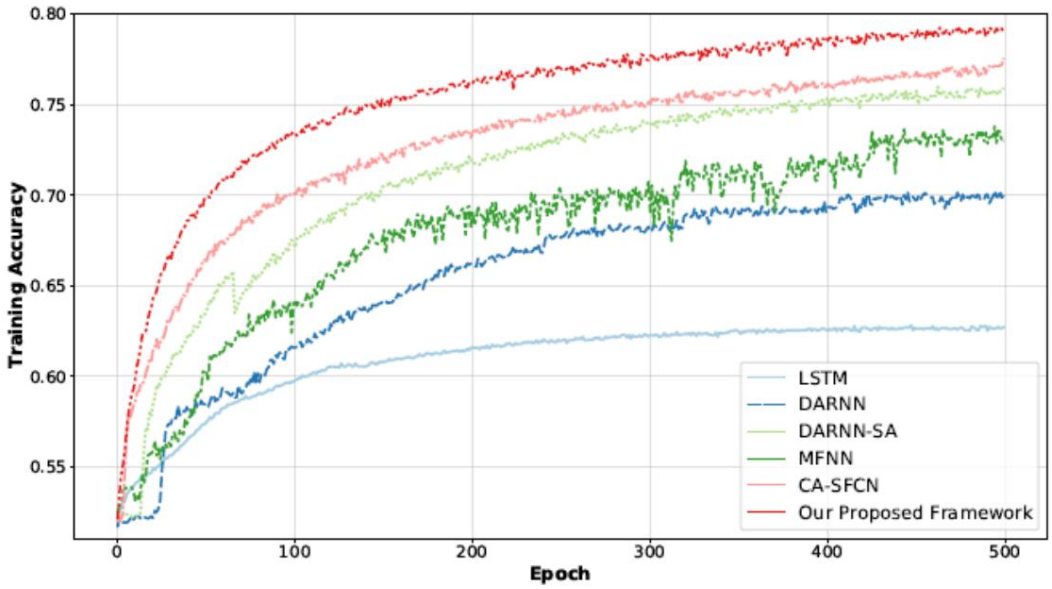

Figure 4.2: Training set results [2]

it is utopian to expect that the proposed structure will be able to accurately capture some properties useful for a better generalization. Indeed, like shows Figure 4.3 (which summarizes in table form the percentages obtained in the tests on the four quarters of 2019), also in this case the accuracy greater is given by this framework.
In addition to accuracy, three other metrics were calculated to help evaluate performance:

- Precision, indicates the accuracy with which the machine learning system predicts positive classes. It is calculated by making the ratio between the true positives and the sum of true positive and false positive.

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

- Recall, indicates the ratio of positive instances that are correctly identified by the machine learning system. It is calculated by making the ratio between true positive and the sum of true positive and false negative.

$$\text{Recall} = \frac{TP}{(TP + FN)}$$

- F1, is the harmonic mean of recall and precision. A harmonic mean gives more weight to small values. This causes a classifier to get a high F1 score only when precision and recall have high values. The calculation is defined as follows

$$F1 = \frac{2\ (\text{precision}\ \text{recall})}{(\text{precision} + \text{recall})}$$

It can be deduced from the table in figure 4.3 that the performances on the test set are therefore in line with those of the training set and this reveals that the structural information contained in the time series of actions are able to address fundamental issues concerning dependencies at long term and chaotic property. There are only a few metric values where the framework in question is slightly exceeded, namely the recall value in the second quarter of 2019 and the precision and F1 values in the third quarter of 2019.

	2019(S1)				2019(S2)				2019(S3)				2019(S4)			
	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
ARMA	50.15	54.96	42.81	48.13	50.75	46.61	42.51	44.46	49.89	44.26	39.91	41.97	50.07	49.40	41.77	45.26
GARCH	50.28	54.90	44.68	49.26	50.66	46.74	45.74	46.23	50.43	45.05	41.76	43.34	50.36	49.75	42.24	45.69
LSTM	57.94	64.88	52.26	57.89	59.88	58.98	44.04	50.43	56.71	52.23	35.33	42.16	54.75	55.97	44.92	49.84
DARNN	60.87	68.27	54.69	60.73	62.03	61.48	48.29	54.09	60.62	58.10	61.33	59.67	61.54	68.34	63.31	65.73
DARNN-SA	64.32	71.72	58.63	64.52	66.23	66.60	54.40	59.89	65.47	65.00	59.09	61.9	65.63	72.34	68.92	70.59
MFNN	61.21	68.28	60.81	64.33	63.00	65.30	51.40	57.52	62.74	67.68	58.75	62.9	64.69	67.19	57.52	61.98
CA-SFCN	65.51	72.82	60.10	65.85	67.21	67.61	**73.52**	70.44	66.10	**77.81**	68.32	**72.76**	67.30	70.24	73.38	71.78
Our framework	**67.48**	**75.24**	**61.45**	**67.65**	**68.46**	**69.81**	71.67	**70.73**	**68.34**	67.86	**73.77**	68.09	**67.91**	**77.51**	**73.78**	**75.60**

Figure 4.3: Test set results

4.2 Experiments with the health care societies

The experiments conducted to evaluate and test the framework just presented were carried out on a different and significantly smaller dataset, containing the stock trends of nine well-known companies in the healthcare sector from around the world. The choice to change the set of samples to be used was made mainly for two reasons, the first of personal interest while the second due to the need to reduce processing times, other minds too long for the devices and computer equipment that you had them available.

The idea of working on financial data that in recent years have seen significant growth due to the Sars-Cov-2 pandemic, as well as being of cultural interest, could be a good test to verify the capture of those addictions to long term and those chaotic properties which have been mentioned previously and which are often the cause of inaccurate prediction.

The data downloaded from the yahoo finance platform [17], in the period from 1 January 2018 to 31 December 2021, belong to the following companies sanitary:

- Agilent Technologies, Inc (A);
- Abbott Laboratories (ABT);
- AmerisourceBergen Corporation (ABC);
- Amgen Inc. (AMGN);
- Boston Scientific Corporation (BSX);
- Biogen Inc. (BIIB);
- Johnson & Johnson (JNJ);
- Humana (HUM);

- Pfizer (PFE).

For each of these listed companies and for each day included in the period indicated above, the following price values were taken:

- high, the highest price reached on that day;
- low, the lowest price reached on that day;
- open, the opening price on that day;
- close, the closing price on that day;
- adjClose, abbreviation of adjusted close price, is the closing price of a share modified to reflect its value after taking into account factors such as dividends, share splits and also the possible issue of new shares. It is often used when looking at historical returns or doing detailed performance analysis

  pass;
- volume, represents the number of contracts of an asset, which are traded in a day.

So in addition to the dataset, a type of share price has also been replaced, namely the adjClose instead of the amount.
All these data will then be transformed into graphs using the visibility graph algorithm to obtain the structural information and perform the same process described in chapter 2.

4.2.1 Dataset splitting

For the data splitting it was decided to use the same method presented in paragraph 4.1.1, in which the data of the years 2018, 2019 and 2020 were used for training, reducing the greatness of the validation set (due to the few samples available); while the data used for the test set were taken from the year 2021, always divided into four quarters. It is good to specify that in the event that they were carried out, for example

for example, tests on the third quarter, data from previous quarters (in this case first and second quarters) would be placed in the training set.
In the histogram in figure 4.4 it is possible to notice that the partition made for the three sets is almost perfectly balanced for each class.

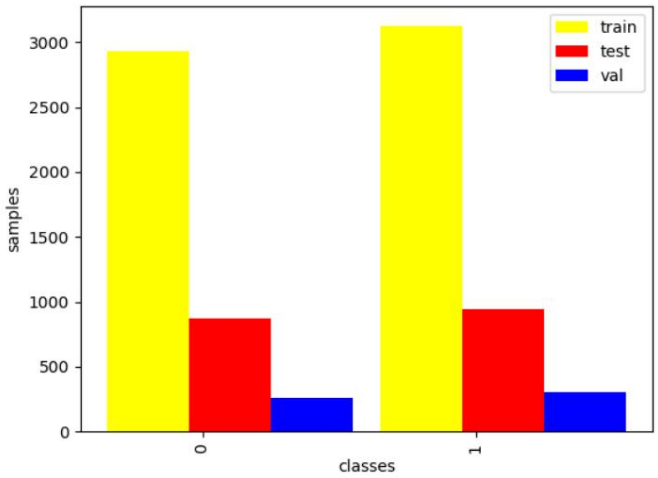

Figure 4.4: Splitting dataset histogram [4]

4.2.2 Hyperparameter tuning

The tuning of the hyperparameters was the most difficult phase of the thesis work as the available equipment, in particular the GPU that had little memory, did not allow to perform some configurations between minibatch size and hidden size .

The parameters considered and their functioning within an artificial neural network are introduced and explained below.

4.2.2.1 Learning rate

The learning rate represents the speed with which the learning model learns. While the direction of descent is usually determined by the gradient of the loss function, the learning rate determines how large a step is in that direction. Too high a learning rate will do

skip learning beyond the lows, but a learning rate too much
low will take too long to converge or may get stuck in an undesirable local minimum.

In this paper the learning rate has been set by default to 0.001 with
some tests on a value of 0.01 which however did not allow the learning network
keep maintaining very high values of the loss function.

4.2.2.2 Activation function

The trigger function is used to map the weighted sum of the input input to the output. This activation feature helps the
neural network to learn complex relationships and patterns in data.
The choice of the type of function is considered a real hyperparame tro, for this reason it is a good idea to test several of them during training for
see which one allows us to have better results.
Within the DARNN model the activation function used was the
hyperbolic tangent, but we also wanted to test the functioning of the activation function Relu (Rectified Linear Unit) which, however, did not produce any
good results. Probably the reason lies in the fact that this last map
the values in an interval that goes from 0 to 1 while the tanh function maps them within an interval that goes from -1 to 1. The possibility of having negative numbers
in this type of learning it probably best describes the importance
to be given to each incoming time series when this is to be the most
low as possible.
In the fully-connected level, on the other hand, it was used to establish the final output the sigmoid function, which produces a value in the interval [0,
1]. If this result is less than 0.5 it will mean that on that day
there will be a decrease in the price, while if it is higher then we will have a increase.

4.2.2.3 Dropout

It is a form of regularization which is applied during the training of the net. The idea is to switch off neurons at each step.

In this way the network is forced to change every time the neurons it uses to represent a given set of data that it is observing according to a certain probability.

In this case a dropout with a percentage of 50% and a percentage of 0% was tested.

4.2.2.4 Optimization method

The optimization methods are required, through an iterative process, for minimize or maximize an objective function called the loss function.

Also in this case it was decided to continue to use the Adam method, acronym for ADAptive Moment estimation, which is basically a RMSprop (Root Mean Square Propagation) with the addition of the moment.

Generally this is the most used optimization method where the moment update is converted into an exponential moving average and it is not necessary to change the learning level when dealing with ÿ. As in RMSprop, an exponential moving average of the gradient squared is calculated, but unlike this it certainly produces less noise.

4.2.2.5 Configurations tested

Once the parameters listed above were established, during the training and testing phases, different combinations of values were tested with regard to the dimensions of the minibatches and hidden sizes.

After a brief analysis it was found that after a certain number of epo that the network was no longer able to learn useful characteristics to increase the accuracy in the test set, so by default the number of epochs was set to 100.

The table below shows all the combinations that exist can you test.

n ° epoch	learning-rate	minibatch	hidden-size
100	0.001	128	32
100	0.001	64	32
100	0.001	32	32
100	0.001	32	64
100	0.001	32	128
100	0.001	64	64

The best results were obtained with an equal minibatch size
at 32 and a hidden size at 128.

4.2.3 Results

As previously described, the model is tested on a different dataset from the one proposed by the Chinese university to see how it behaves.
with equity trends influenced by the pandemic.
In addition, various metrics on the graphs originating from the series will be calculated temporal, to understand if collective influence is the best weight to use for
dealing with chaotic property or whether others can be taken into consideration.

By initially analyzing the training phase among all the models, it can be deduced that the percentage of accuracy achieved is very similar between all five
models proposed, touching an average of 90%. This probably means that collective influence is a great way to combat property
chaotic of financial time series, but so too are the financial time series
other metrics calculated on the graphs that are able to give further information useful for updating the attention weight inside the decoder of the
DARNN.

The results obtained for each model with a con will be listed below
comparison and a final evaluation on the values obtained.

4.2.3.1 Model results with Collective Influence

In this first test, as already introduced previously, we want to reuse the same model produced by the Chinese authors on a dataset different, much smaller in terms of number of samples, which however unlike the Chinese one, it is made up of equity data in the full pan period demico.

Taking into account the shares of health care companies in a period of this type, it allows you to go to work on time series that are not stationary and decidedly oscillating, becoming an excellent test for mapping long-term dependencies and dealing with chaotic property.

According to the results obtained in the training phase, this is also expected case of achieving a high percentage of accuracy that it is capable of map similar periods in the trend between the train and the test set.

From the four confusion matrices present in figure 4.5 we can read here the results obtained for each quarter of 2021 respectively.

diagonal of each matrix, from the upper left cell to the lower cell on the right, you can see the correct predictions that have been made since model on the performance of equities.

The total number of 0 indicates how many times the stock has dropped since the day previous to next day, while the total number of 1 indicates the number of rises.

In quarters 2, 3 and 4 the accuracy percentage is around 79%, while in the first quarter at 71%.

4.2.3.2 Model results with Betweenness Centrality

In the second test we want to experiment with the use of a metric calculated on graphs as the weight of the nodes. The basic idea remains to switch to network one previous knowledge which however no longer corresponds to the calculation of the collective influence, but to the betweennes centrality for each actor within a graph.

Starting from the idea that centrality indices evaluate, as the collective influence, how much a node within the various paths is important for the passage information, calculating the centrality by interposition could be

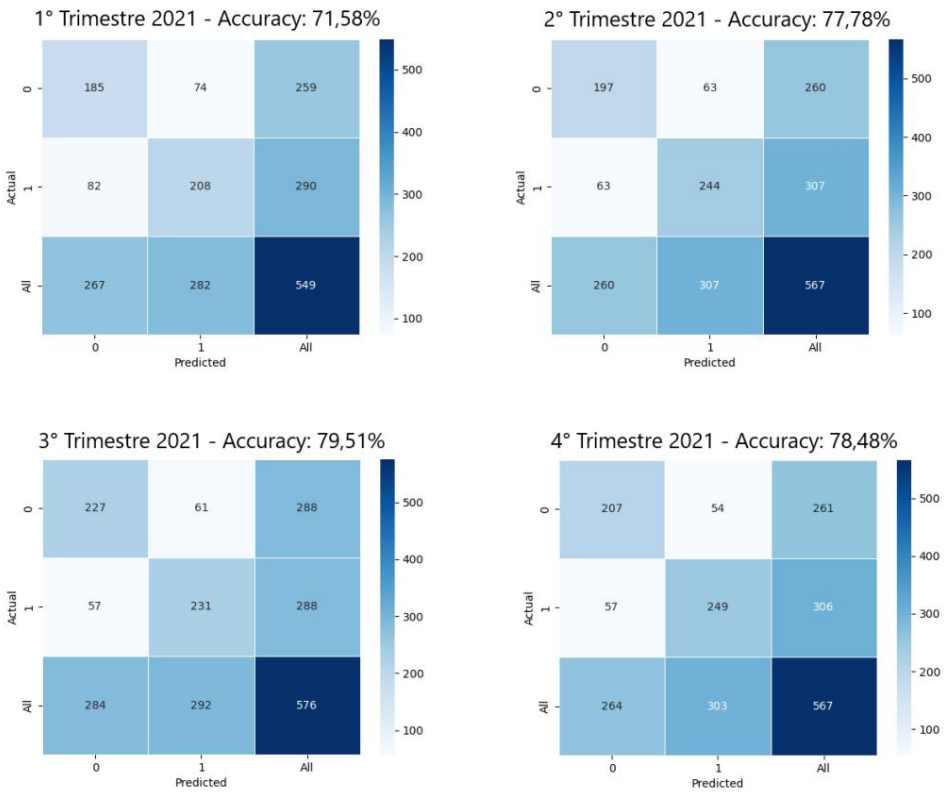

Figure 4.5: Confusion matrices for the first test

I help to identify which days are really to be taken into consideration for the trend in the price of the financial security since without them the destination nodes would not be influenced by the origin nodes.

It is good to specify that with this metric only the shortest paths between two nodes are taken into consideration, so the importance of the node could be even higher if all the possible paths within the
net.

From the results present in the confusion matrices of figure 4.6, it would seem that the ability to map the input into the expected output is as valid as that of the previous test, obtaining in the first quarter an accuracy percentage higher than three percentage points and remaining on the same level for the other three quarters.

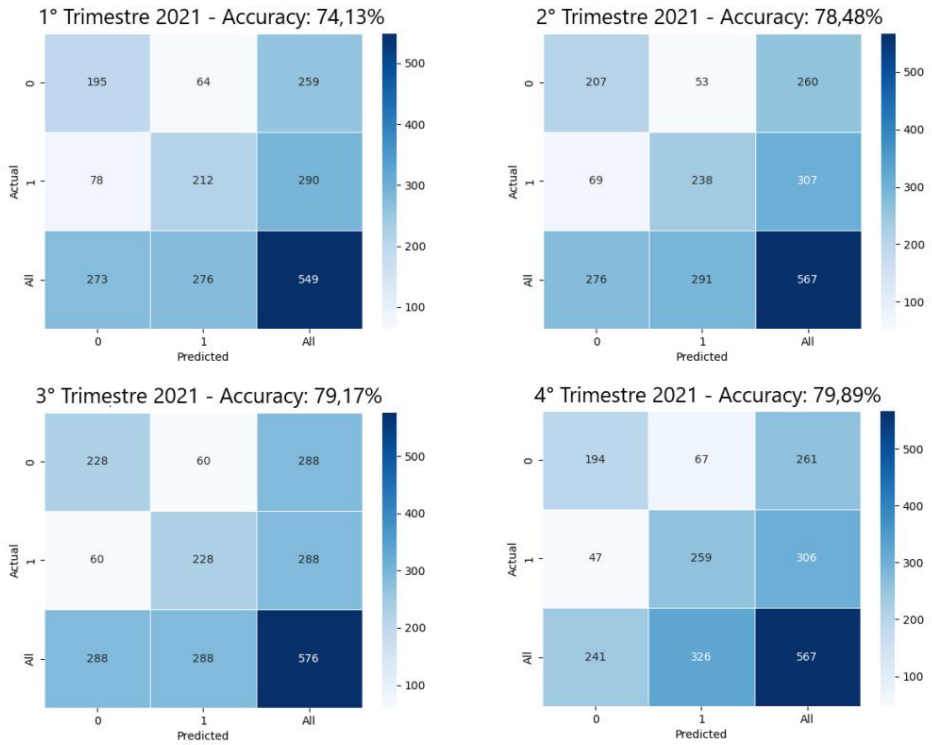

Figure 4.6: Confusion matrices for the second test

4.2.3.3 Model results with Closeness Centrality

In this third test, the idea always remains the same as the previous one, that is, always take into account a centrality index as a metric, which however in this case corresponds to closeness centrality. Unlike the previous index, closeness centrality focuses on the proximity of one node to all the others, thus obtaining an actor's weight high would mean that node directly affects all nodes close to he.

In the analysis of a time series in which we want to understand how much the price of a day can influence the price of the days immediately preceding or immediately following this, it can be done using this index.
From the matrices present in figure 4.7 it can be seen that also in this case the prediction obtains very encouraging results. Although in the first quarter, as in the previous tests, it is more difficult to obtain a good function of mapping, in the other quarters the percentages of accuracy calculated are similar or slightly higher than those of the previous tests. Probably this it means that slightly more features can be identified significant and specific that allow you to earn that percentage point in terms of forecast accuracy.

4.2.3.4 Model results with Clustering Coefficient

The fourth test involves the use of an additional metric to be used as previous knowledge for updating the attention weight, that is the clustering coefficient.

The idea of calculating this index on graphs constructed from series temporal, was born from the visualization of the small networks that were created. In the image in figure 4.8 there are four examples of graphs created by series temporal, where the number of nodes is given by the size of the loo kback window (as defined in the previous chapters). As can be easily seen, i knots tend to form a lot of triangles. From this observation it is therefore thought it might be interesting to calculate its own for each node of the graph local clustering coefficient, which is a measure of how much i

Figure 4.7: Confusion matrices for the third test 7

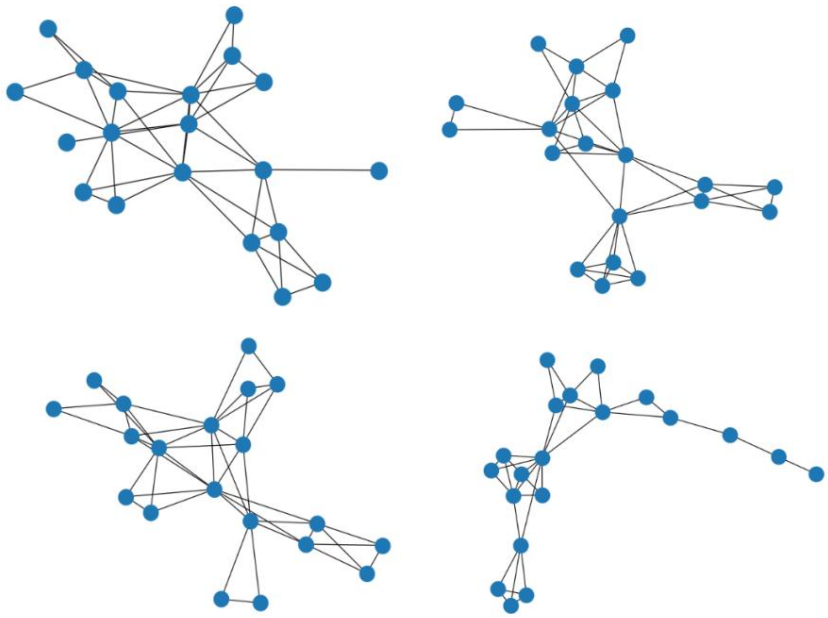

Figure 4.8: Graphs generated from the time series [8]

its neighbors tend to form a complete graph.

In fact, from the results present in the confusion matrices in image 4.9, the the percentage of accuracy obtained in the various quarters is by no means delu tooth.

In the first three months, the story does not change, the accuracy remains the lowest and in this case it is significantly lower than the rival tests. Even in the second and in the fourth quarter the percentage is lower than the other tests, but it approaches greatly to their values. The best result is instead given by the third quarter in which a percentage close to 81% is obtained, equalizing that of the test. done with closeness centrality.

4.2.3.5 Model results with Clustering Coefficient and Betweennes Centrality

Finally, in the last test we wanted to experiment with the use of two metrics for try to give the network a stronger imprint than the time series at the time t in inputs that necessarily had to be considered.

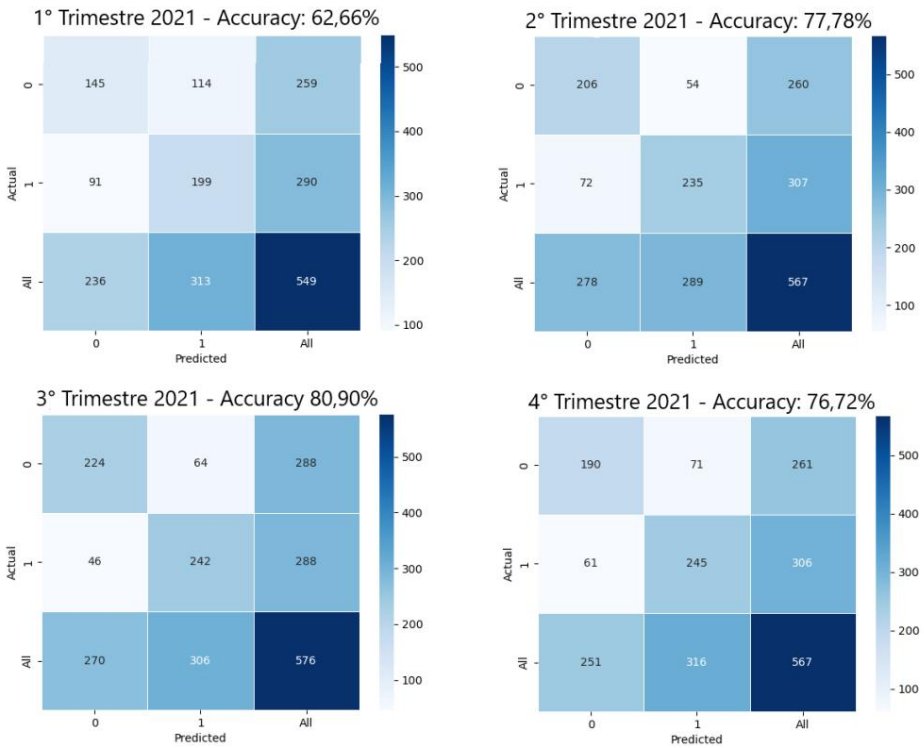

Figure 4.9: Confusion matrices for the fourth test 9

The motivation that led the most to attempt this test too was the desire to try to achieve the accuracy of the first one as well.
quarter of the fourth experiment to a percentage close to all its contending. For this reason the idea was to try to further mark the
value of the clustering coefficient, adding to it the value of the centrality index which in the previous tests had produced the best result. Consequently
it was decided to add in the update the attention weight of the decoder
of DARNN is also the value of the centrality by interposition of the current node at time t.

This experimentation made it necessary to change the network model inside the decoder, by inserting an additional hidden layer that
it dealt with taking the value of the second metric as input.
From figure 4.10 below it can certainly be said that the goal was not
reached, indeed a sort of contraindication has occurred, because the capacity
forecasts for the first quarter improved slightly and that for
second quarter has worsened considerably. A reason for this disappointing result can be given by the signs of the values of the two metrics, because
two signs both negative or both positive would respectively reinforce the fact that that day should not be considered or yes, but
two values with opposite sign could alter the update producing
more false negatives or false positives. To confirm this hypothesis, it is enough
check the respective confusion matrices where the quadrant numbers (false
positive and false negatives) are higher than those of the test matrices
previous.
For the other two quarters, however, excellent results were obtained, so perhaps the idea could also be right, but it is the method of
weight update that should be changed.

4.2.3.6 Final comparison

In this section we want to make a summary of the results obtained from these five experiments trying to give it meaning by comparing, in addition to accuracy, also the values of Precision, Recall and F1.

By examining the table in figure 4.11, it can be seen immediately

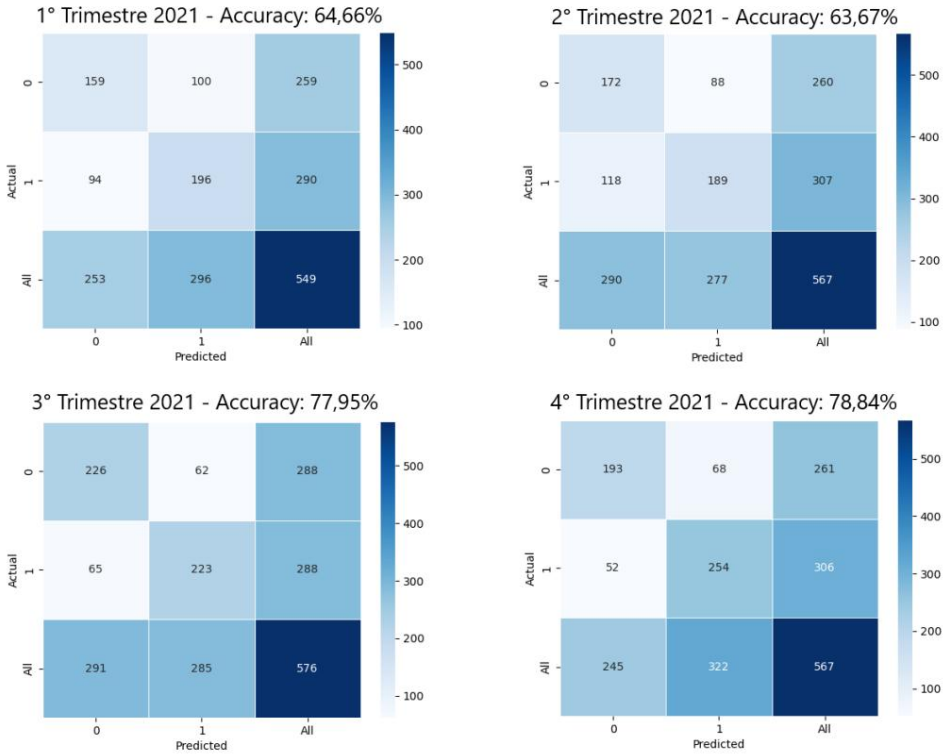

Figure 4.10: Confusion matrices for the fourth test 10

	2021(S1)				2021(S2)				2021(S3)				2021(S4)			
	Accuracy	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
1° Test (CI)	71,58%	71,72%	73,76%	72,73%	79,54%	80,71%	81,76%	81,23%	79,51%	80,21%	79,11%	79,66%	78,48%	76,47%	82,39%	79,32%
2° Test (BC)	74,13%	73,31%	76,81%	74,91%	78,48%	77,52%	81,79%	79,60%	79,17%	79,17%	79,17%	79,17%	79,89%	79,45%	84,64%	81,96%
3° Test (CC)	70,13%	75,20%	64,83%	69,63%	79,89%	80,63%	82,74%	81,67%	80,90%	81,79%	79,51%	80,63%	79,54%	81,88%	79,74%	80,79%
4° Test (CL)	62,66%	63,58%	68,62%	66,00%	77,78%	81,31%	76,55%	78,86%	80,90%	79,80%	84,03%	81,48%	76,72%	77,53%	80,07%	78,78%
5° Test (CL + BC)	64,66%	66,22%	67,59%	66,89%	63,67%	68,23%	61,56%	64,73%	77,95%	78,25%	77,43%	77,84%	78,84%	78,88%	83,01%	80,89%

Figure 4.11: Confusion matrices for the fourth test 11

for each evaluation index which tests have reached the highest percentage (cells highlighted in yellow).
In the first and fourth quarters, the use of centrality by interposition definitely prevailed over all, while in the other two quarters it was more efficient to consider closeness centrality and the clustering coefficient as previous knowledge. of each node.

The first consideration that can be made concerns the results obtained from the collective influence. For each quarter a value was obtained in line with the best results of the other tests, but the highest value was never reached and this means that there are probably computable metrics on graphs that can address the property to chaotic in a slightly more efficient way, also improving the capture of short and long term dependencies.
Secondly, it is clear to ask why in the first quarter the numbers reached are lower than in the other quarters. One reason can undoubtedly be that the test set has fewer samples due to lack of prices for a few days. Secondly, the train set on which the network was trained also had less data and this is due to the fact that the quarters prior to the test one were incorporated into the train set to try to further improve the mapping input function. - output.

Another aspect that may seem strange looking at the table is that the highest precision never belongs to the test that has reached the highest accuracy, but this is not a problem since in this case it is more It is important to have a high recall which defines for a class how many correct predictions it can identify. Apparently in these cases more positive are predicted than are actually true.

In conclusion, it can be said that there has not been a winning test, but more or less they have all been equal. Surely to have more certainties, it will be necessary to take a dataset containing a number of samples in the hundreds of thousands to try to obtain a more generalized training and perhaps reach considerable differences in terms of accuracy between the various experiments.

Chapter 5

Conclusions

In conclusion, within this thesis a new framework has been presented, created by a Chinese university, for the prediction of the trend of financial securities based on the structural information captured by the time series graphs converted from the data of the market prices.
In a first part it was shown that this structure, collaborating with different deep learning techniques, achieves competent performances and shows practical abilities in predicting actions by solving two of the problems that cause greater difficulty in the prediction phase. In fact, using models with attention mechanisms, it has been discovered that, through structural information, the long-term dependencies of the values in the time series can be better captured, while using the collective influence of the nodes of a graph as knowledge additional to the time attention, one is able to deal also with the chaotic property of financial time series.

In the second part, on the other hand, we wanted to experiment with different tests using the same structure.
First of all we wanted to test the framework with a smaller dataset, in terms of number of samples, and different from the one containing the Chinese indices because it is composed of data taken in a period in which the stock market was very unstable due to the Sars-Cov-2 pandemic and therefore the probability of having many fluctuations in the price was greater.
Subsequently, three other metrics (closeness centrality, betweennes centrality and clustering coefficient) were calculated on the graphs instead of the influence

collectively to see if they could cope more efficiently
the chaotic property.
Finally, in the last test we wanted to modify the structure of the network decoder
DARNN to allow him to consider not just one metric as the weight of the knot, but two.

The results obtained were quite satisfactory because in terms of percentage accuracy on the test
set, with the second, third and fourth tests, we
values close to or even slightly higher than the first are often reached
experiment. This probably means that collective influence
it is not the only type of weight that can help the network understand which nodes are
more significant to take into consideration.
Furthermore, the achievement of high percentages for the values of accuracy, precision,
recall, f1 demonstrates that even with a dataset affected by many fluctuations
in price, the mapping of long-term dependencies is significant
mind.
The results of the last experiment are more disappointing, since in two of the test quarters the
accuracy percentage remained significantly lower.
compared to the others and this suggests that perhaps the summation is not the most correct
operation to carry out if one wants to take into account values of
different metrics to update the attention weight.
However, it is necessary to specify that due to the inefficient equipment available, the results
obtained from the experiments must be taken with a grain of salt,
because they are the result of an elaboration produced on very small datasets.
On the other hand, however, they lay the foundations to encourage a future study
training the network on hundreds of thousands of samples and maybe even on
equities from different financial sectors.
In any case it can be said that the performances obtained by this framework
and from these tests enrich the existing financial research study for
prediction of actions thanks to the representation of time series
across complex networks.

5.1 Future developments

The future developments that can be made starting from this work could take many different directions, certainly one of the most important will be to replicate what has been done on a dataset rich in financial stocks trends to verify the effective help that metrics used in this thesis can contribute to the neural network model for the prediction of the price flow.

A further experiment could involve the calculation of other metrics on graphs, such as global efficiency or eigenvector centrality, where the latter, unlike betweennes centrality, takes into consideration all the possible paths for a certain node and not just the shorter ones.

Finally, it might be interesting to abandon financial time series to try to use this framework on totally different fields that exploit time series.

Bibliography

[1] J. Wu, K. Xu, X. Chen, S. Li, and J. Zhao, "Price graphs: Utilizing the structural information of financial time series for stock prediction," Information Sciences, vol. 588, pp. 405–424, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025521013104

[2] H. Ebel, J. Davidsen, and S. Bornholdt, "Dynamics of social networks," Complexity, vol. 8, no. 2, pp. 24-27, 2002.

[3] P. Ongsulee, “Artificial intelligence, machine learning and deep learning,” in 2017 15th International Conference on ICT and Knowledge Engineering (ICT & KE). IEEE, 2017, pp. 1–6.

[4] Legaldesk. [Online]. Available: https://legaldesk.it/blog/artificial-intelligence-application-fields

[5] C. Chen, L. Zhao, J. Bian, C. Xing, and T.-Y. Liu, “Investment behaviors can tell what inside: Exploring stock intrinsic properties for stock trend prediction,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2376-2384.

[6] Pandas. [Online]. Available: https://pandas.pydata.org/

[7] Numpy. [Online]. Available: https://numpy.org/

[8] Scikit-learn. [Online]. Available: https://legaldesk.it/https://scikit-learn.org / stable / index.html

[9] Networkx. [Online]. Available: https://networkx.org/

[10] Pyunicorn. [Online]. Available: https://pypi.org/project/pyunicorn/

[11] Matplotlib. [Online]. Available: https://matplotlib.org/

[12] Pytorch. [Online]. Available: https://pytorch.org/

[13] JF Donges, J. Heitzig, B. Beronov, M. Wiedermann, J. Runge, QY Feng, L. Tupikina, V. Stolbova, RV Donner, N. Marwan et al., "Unified functional network and nonlinear time series analysis for complex systems science: The pyunicorn package," Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 25, no. 11, p. 113101, 2015.

[14] F. Zhu, "Improved collective influence of finding most influential nodes based on disjoint-set reinsertion," Scientific Reports, vol. 8, no. 1, pp. 1–12, 2018.

[15] LF Ribeiro, PH Saverese, and DR Figueiredo, "struc2vec: Learning node representations from structural identity," in Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 385–394.

[16] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual stage attention-based recurrent neural network for time series prediction," arXiv preprint arXiv: 1704.02971, 2017.

[17] Yahoo. [Online]. Available: https://finance.yahoo.com/

Acknowledgments

Finally, I would like to thank all the people who helped and supported me in completing this course of study.

A heartfelt thanks goes to my supervisor Stefano Ferretti for his infinite availability, kindness and timeliness to my every request and for helping me in the drafting of this thesis.

I will never stop being grateful to my parents and my brother who gave me the opportunity to enroll in university, encouraged me and encouraged me in every difficult moment of this course of study and that without their teachings today I would not be what I am.

I sincerely thank my fiancée Sara for giving me peace of mind, courage and lightheartedness. Thank you for all the time you have dedicated to me, for the patience you have had and for always having been there.

Finally, a sincere thanks also goes to my two colleagues, as well as friends Gianluca and Giacomo, with whom I spent these two years between lessons, exams and video calls, where their help was also fundamental to reach this goal.