POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

An LSTM-based model to Trading Energy Stocks

Supervisors
Prof. Barbara Caputo
Dr. Giuseppe Rizzo

Candidate
Alberto Benincasa

December 2020

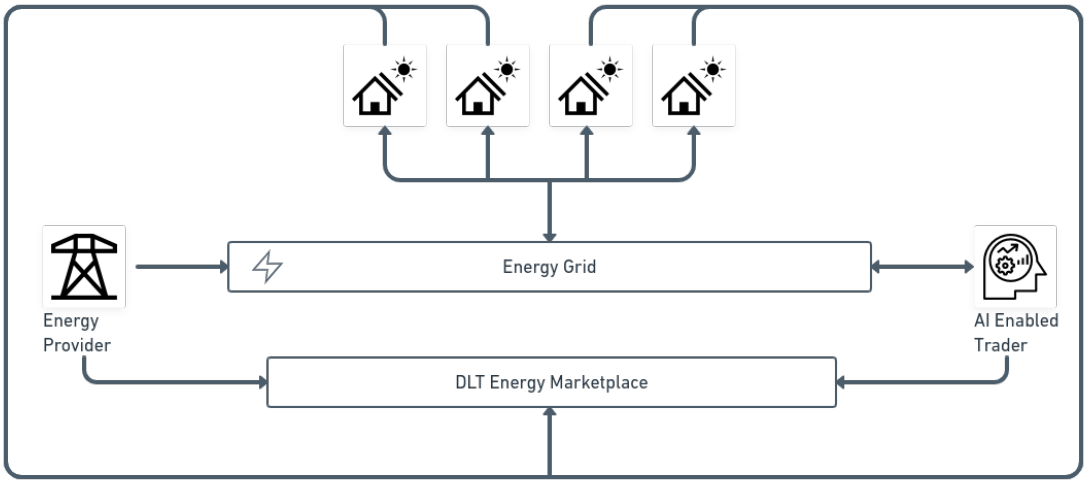
Summary

When we talk about Artificial Intelligence, in the collective imaginary, we often think about the "prediction of the future": this is probably a result that we could achieve in the future, and Time Series Forecasting are the closest models to this ideal. Today, the price prediction field is one of the most important in AI, both from a theoretical and a practical perspective. Indeed, the interest of equity market investors towards this type of solutions has increased more and more in recent years, as not only prediction models are capable of storing and processing more data that a person can manage, but also a lot of time and money is already spent in order to collect these data. Moreover, an AI-based tool can find correlations otherwise impossible to see in the same time frame. On the other hand, from the researcher's perspective, this field offers many challenges that improve the quality of predictions more and more. This is because the stock market is essentially dynamic, nonlinear, complicated, nonparametric, and chaotic in the financial time series prediction process.

The work presented in this thesis is inserted in this context. In fact, the scenario we faced, schematized in Figure 1, is the following: after having collected the data relating to the price of energy and information regarding the weather and energy consumption of a collection of houses, our goal was to manage the contribution of the energy grid automatically, selling energy when it was not needed and buying it when needed. All this to optimize not only energy waste but also the economy of the complex.

To do this, we used three different datasets: the first collects historical data relating to the price of energy between 2010 and 2013, the second and the third, relating to the same period, collected data on the climate (precisely, exposure solar energy and amount of rain) and those of energy consumption, in particular for every 30 minutes, we have a value that indicates whether we are in surplus, which means the use of energy is lower than the produced one, or in deficit; the data derive from a collection of 300 households that

Figure 1: Households grid structure.



have rooftop solar systems and a gross metered solar system installed. Then we implemented four different algorithms, using both Machine Learning and Deep Learning approaches:

- Facebook Prophet: that is a procedure for forecasting data, and it is based on a decomposable time series model with three main model components: a linear or logistic growth curve for modeling data trends, one for modeling periodic changes, one for the effects of holidays and the last one for catching the unusual changes.

- XGBoost: eXtreme Gradient Boosting is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It consists of a tree structure which aims to make decisions and discriminate between different classes. XGBoost analyzes the data and maximizes the decision making of this tree through the use of machine learning techniques, where the final prediction for a given example is calculated as the sum of the gradient statistics on each leaf, and then apply the scoring formula to get the quality score.

- Long-Short Term Memory (LSTM): these Networks are a special kind of Recurrent Neural Networks, explicitly designed to avoid the so-called long-term dependency problem. Their main feature is the capability of removing or adding information to the cell state, remembering them for long periods of time regulated by structures called gates.

- Stacked LSTM: a variant of LSTM implemented by us, where we took all the additional information, concatenated them into one normalized vector and fed it into the LSTM Cell during the forward phase together with the hidden inputs derived from the price data. In this way, we kept intact the importance of the information on the price but still adding more information that can be used for the prediction.

From the results, we were able to see how the Deep Learning-based approach has produced generally better results, confirming how the greater complexity and adherence of structures such as LSTMs to data encoded in time series help this type of problem. Furthermore, although in the literature we can find similar task solutions with better results (albeit with different data and contexts), the important take-home message we have derived is that using different features that characterize price fluctuations has a considerable impact on prediction. Therefore, in order to be able to identify with more certainty and precision other possible factors that determine these variations, the proposed solution can obtain even better results by merely changing or extending the external data. As for the architecture we tested, it is possible to extend it using bidirectionality, which in the training phase allows the network to obtain information both from the past states of the cell (backwards) and from future ones (forward) at the same time t, increasing the number of inputs known to the network.

Abstract

This work presents an intelligent tool developed utilizing a deep learning technique to trade energy stocks. The tool forecasts energy prices and recommends when to buy and how much energies from different households. First, we introduce the world of time series, showing examples of fields where they are being successfully used as well as their growing importance nowadays. We then describe how they have been classified through the years and the different technologies that have been experimented with for solving forecasting problems in different ways.
Then, we discuss the world of recommending systems and how they are becoming more and more important in the context of classification.
We will explain the problem that this work tries to solve, which consists of the creation of an intelligent tool capable not only of analyzing and processing the past data of the electricity market in order to predict the price trend of the following two days, but also to select the best time to buy and sell energy in that time frame. Such transactions are recorded on the blockchain (thanks to the Accenture team I have worked with). The tool thus created has the purpose of optimizing the management of the grids that supply energy to groups of houses.
After using some of the most popular machine learning regression libraries, we decided to implement a particular Recurrent Neural Network called Long-Short Term Memory (shortened LSTM). We then implemented a Stacked LSTM: the attributes that can affect the price value (such as information on the time of day and the weather) are not given in input to the LSTM cell immediately but are inserted later. All the data used refer to the electricity market in the south of Australia. The approach we propose resulted in an improvement in the results that could be appreciated using only machine learning and the basic implementation of LSTM.

Acknowledgments

It is mandatory for me to give thanks to the people who have supported and accompanied me during these years.

Thanks first to my family, who has always supported me and my decisions without ever making me feel the pressure for my studies. They made everything easier.
Thanks also for tolerating me when I was under stressful periods and for believing in me even when I was staggering.

Thanks to my friends, the ones I met at university and the ones I already knew. You helped me when I had difficulties, you listened to me when I needed it, you put up with me and you managed to make difficult periods easier to bear.

A final thanks also goes to Giuseppe, with whom I had and still have the pleasure of working: a person with whom I found myself comfortable since the very beginning, understandable, competent and a great motivator.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The purpose of this thesis is to develop a solution to a Energy Market Prediction problem, more precisely our value proposition is to build a tool able to predict the price trend of electricity and to optimally manage its purchase and sale. To achieve our goal we used Time-series Analysis techniques in order to generate a model able to extract and use relationships between price data and external information.

1.1 The team

The project described in this thesis is the result of the combined work of two different companies: LINKS Foundation and Accenture Labs. Since the project brings together two of the most critical modern topics, Deep Learning and Blockchain, the work has been divided as follows: as regards LINKS Foundation, with which I worked with the support and supervision of expert figures of the sector, such as Giuseppe Rizzo (senior researcher at LINKS), took care of the development of the Artificial Intelligence tool described in Chapter 4 together with the API interfaces needed to communicate data externally. Accenture Labs instead has the task of presenting the predictions made and managing the transactions through Blockchain. We will focus on the work done by LINKS Foundation.

The union of these two modules defined a prototype, which has been submitted to the Patent Commission and is currently being under evaluation.

1.2 Why Deep Learning?

Deep Learning is a new technology that is obtaining more and more interest in recent years, especially in fields like Computer Vision, Natural Language Processing and Time Series Processing. Deep Learning means using a neural network with several layers of nodes (called neurons) between input and output. These layers are able to compute relevant features automatically in a series of stages, just like the process of an human brain.

1.3 Results

This work compares the Deep Learning models (vanilla LSTM and Stacked LSTM) with those of Machine Learning (XGBoost and FacebookProphet). The Deep Learning approach has obtained better results for all the metrics used (Mean Squared Error, Mean Absolute Error, and Mean Absolute Percentage Error), confirming how much the use of architectures based on Neural Networks leads to more accuracy in the price predictions in the field of forecasting.

Chapter 2

State of the art

In the following chapter, we describe how Time Series Forecasting has been studied and classified, with an additional analysis of the technologies that are obtaining good results in this field.

2.1 Time Series Forecasting

Time series forecasting has a crucial difference from the standard regression problems: the constraint of the order. In fact, the chronological order of the data makes it harder for an estimator to learn an overall model that can be used, as patterns may appear for some time and then disappear, or even the entire distribution of the data might change. Furthermore, the main reason we analyze Time series is forecasting, where we try and use past observations to predict the future. To do this, we have to consider two main variables:

- The past values of the series called **endogenous variables**.
- The external factors that may be correlated with the distribution of the data. These factors have to be deterministic; otherwise, we are in multivariate Time series forecasting, where these values have to be predicted. Anyway, this kind of variables takes the name of **exogenous variables**.

As for the predicted values, we can decide to predict the value after the considered time:

$$\hat{y}_{t+1} = f(y_0, y_1, \ldots, y_{t-1}, y_t) \tag{2.1}$$

Or, starting from knowledge at time t we can decide to predict data which it is not at time $t+1$ but at time $t+x$. This can be done in two ways: the first

one is the direct approach, which uses the past values to forecast directly the future value:

$$\hat{y}_{t+x} = f(y_0, y_1, \ldots, y_{t-1}, y_t) \tag{2.2}$$

The second one is the iterative approach, where in order to compute the value at time $t + x$, all intermediate points are predicted and used as a basis for the next prediction. in this case, however, it must be emphasized that the error is propagated for all elements, which can cause poor accuracy for predictions very far in the future:

$$\begin{aligned}
&\hat{y}_{t+1} = f(y_0, y_1, \ldots, y_{t-1}, y_t) \\
&\hat{y}_{t+2} = f(y_0, y_1, \ldots, y_t, \hat{y}_{t+1}) \\
&\vdots \\
&\hat{y}_{t+x-1} = f(y_0, y_1, \ldots, y_t, \hat{y}_{t+1}, \ldots, \hat{y}_{t+x-2}) \\
&\hat{y}_{t+x} = f(y_0, y_1, \ldots, y_t, \hat{y}_{t+1}, \ldots, \hat{y}_{t+x-2}, \hat{y}_{t+x-1})
\end{aligned} \tag{2.3}$$

2.1.1 Sliding Window

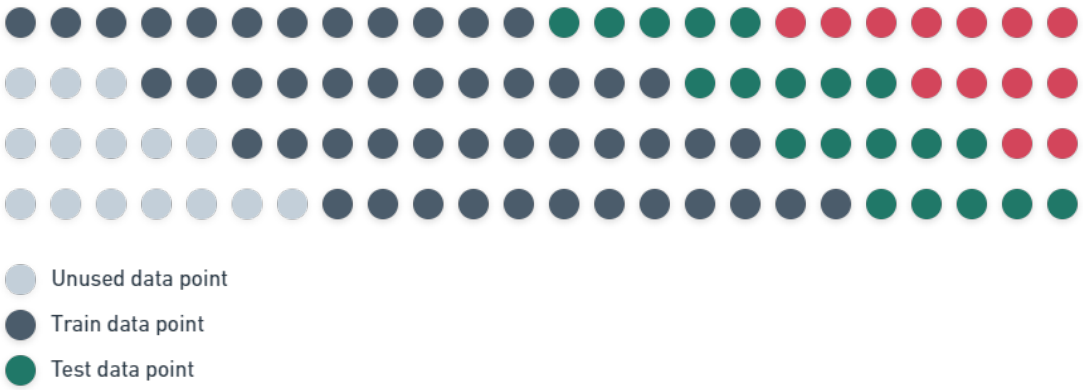
The traditional approach is to train the model on a portion of the dataset (the training dataset) and compute the performance on the test dataset (Figure 2.1). The problem with this strategy is that we cannot assume that the distribution of the series never changes, and it is not necessary that some variables need to be used or modeled. To prevent these problems, it is often used a sliding window approach.

Figure 2.1: Standard forecasting approach utilizing train and test datasets.

The sliding window strategy is the state of the art implementation for Time Series Forecasting as it can provide better predictions. The mechanism

can be found schematized in Figure 2.2, and it can transform the time series dataset into a supervised learning problem, which is when the process of training is *supervised* because we know the correct answers, so that we can correct errors on predictions done on training data.
To achieve this, we change the structure of the data to see, at the same time t the input and the next time step output, keeping intact the order between the observations. Furthermore, we can set a different width of the sliding window to consider more previous time steps.

Figure 2.2: Sliding window mechanism.



2.2 Machine Learning

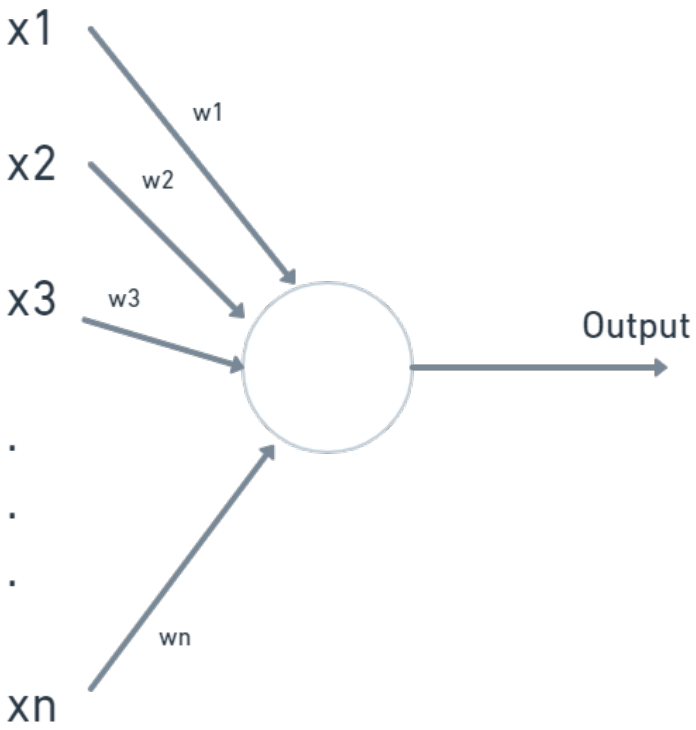
Machine Learning is a subset of AI algorithms that enable the machine to learn through experience, simulating human intelligence. Machine Learning applications are several, such as image recognition, speech recognition, multimedia platforms, and much more.
The perceptron was the first architecture that implemented the principle of taking different vectors as input and adjusting the internal weights considering the predicted and the expected value, of which we can see the structure in the Image 2.3, where the output is calculated as follows:

$$Output = \sigma(w * x + b) \tag{2.4}$$

$$\sigma = \frac{1}{1 + e^{-t}} \tag{2.5}$$

Lately, the Machine Learning has been classified in two subgroups:

Figure 2.3: Perceptron structure.



- **Unsupervised Learning**: here we extract features and information from a set of data with no labels. Examples of unsupervised learning are Clustering process or Principal Component Analysis.
- **Supervised Learning**: this is the most common type and consists of predicting the correct label taking as input the item features.

2.3 Deep Learning

Over time, having a greater number of data led to the need to have facilities that can handle it. This need was felt particularly in the context of time series and benefited from the birth of Neural Networks, which are a concatenation of multiple perceptrons where every layer extract features from the previous one.
One of the best architecture that can handle the problem of Time Series Forecasting is the Recurrent Neural Network because it can take advantage

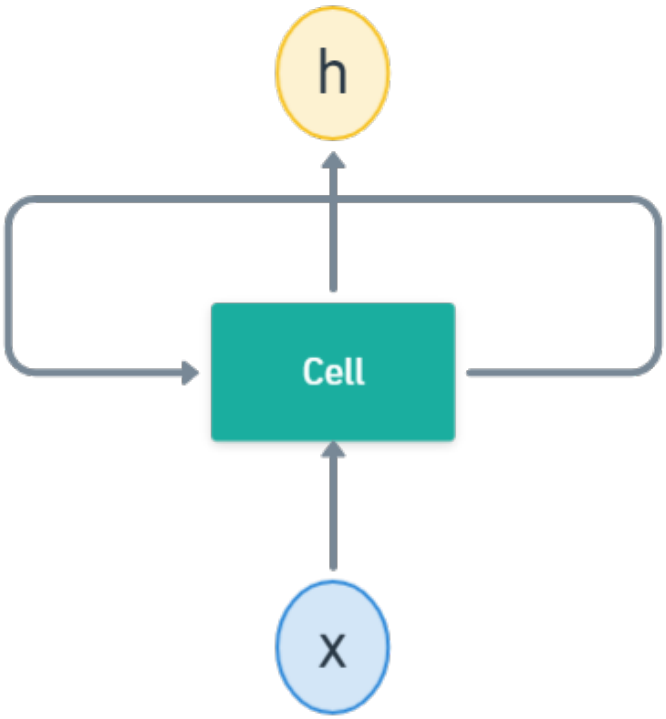
of the order in the data. Recurrent Neural Networks are a special type of Neural Network with the peculiarity of looking at the current input and the previous ones. We can see a simplified structure in Figure 2.4. The Cell is a Feedforward neural network, and we compute the output using the following computations:

$$A_t = g(W_{ax}X_t + W_{aa}A_{t-1} + b_a) \tag{2.6}$$

$$H_t = g(W_{ya}A_t + b_y) \tag{2.7}$$

A is the output of the hidden layer, $g(t)$ is the activation function, W is the weight matrix, X is the input, and H is the output at time step t, and finally, b is the bias.

Figure 2.4: A Recurrent Neural Network.

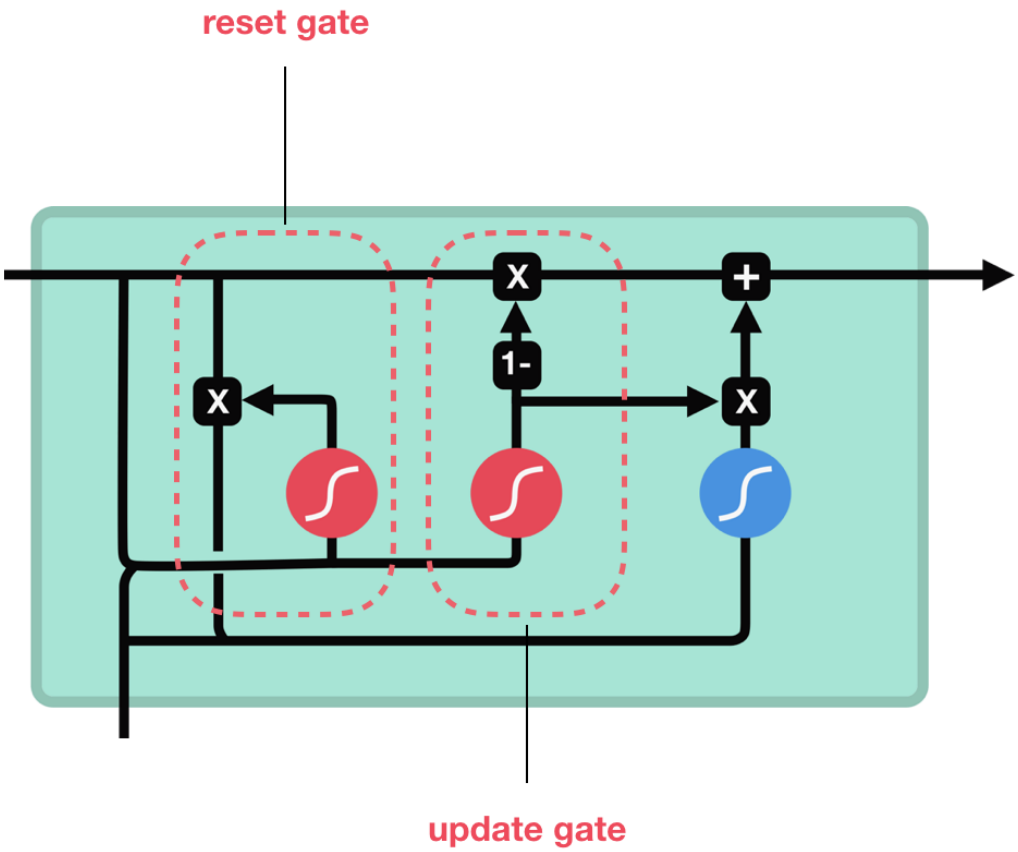


Out of the RNN units present in literature, the most useful and most widespread for the problem we will presenting in Chapter 3 is the Long Short Term Memory, that we will describe in-depth in Section 4.2. But Another architecture could fit our problem well, and it is the Gated Recurrent Unit (GRU, Figure 2.5).

This type of Neural Network is simpler than the LSTM one, which makes the computations faster. It uses structures called gates in order to manipulate the current data and the previous ones, and we have two gates in GRU:

- **Update Gate**: based on the new information, it decides which features are to be kept and which not.
- **Reset Gate**: it decides which features from past events are to be forgotten as they are not relevant for the output.

Figure 2.5: GRU Cell Structure. Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf2



2.4 ARMA Models

One of the models that fits the problem of Time Series Analysis the most is the autoregression model, where the prediction is done after the linear combination of the past values. The autoregressive model can be written as:

$$y(t) = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t \tag{2.8}$$

$$y_t = c + \epsilon_t \sum_{k=1}^{p} \phi_k y_{t-k} \tag{2.9}$$

where ϵ_t is the noise, ϕ are the parameters, and p is the order of the model, from which the notation **AR(p)** is born that means $autoregressivemodelof orderp$ [2].

Similar to autoregressive models, there are the Moving Average Models. The main difference is that the intercept is the average of the most recent q values, so we regress on the difference between the average and the observed values.

$$y(t) = c + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots + \phi_q \epsilon_{t-q} \tag{2.10}$$

As for the autoregressive model,ϵ_t is the noise, ϕ are the parameters, and q is the order of the model and the notation is **MA(q)** [4].
The MA component is the linear combination of latest residuals with $\phi_{i\in[1..q]}$ the weight of each error correction.

The combination of AR(p) and MA(q) generates the ARMA(p,q) model, which is the sum of the two models mentioned. The most important weakness of these models is the constraint to have **stationary** time series, which means that there are features that do not depend on the time at which the series is observed [10]. This, in real life challenges are not true in almost every case: so we need an operation to stazionarize the series. With the **ARIMA(p,d,q)** an operator called $back - shift$ is introduced [6]:

$$\Delta y_t = y_{t-1} \tag{2.11}$$

$$(1 - \Delta) y_t = y_t - y_{t-1} \tag{2.12}$$

$$\Delta^d y_t = \Delta^{d-1} y_{t-1} = y_{t-d} \tag{2.13}$$

With the introduction of this operator, we can handle non-stationary time series, with d being a variable that defines the number of differencing performed on datapoints. Still, we have the problem that ARIMA models are

not able to handle seasonal data as it does not model this component in its mathematical formulation.

For this reason, the ARIMA model was generalized by Box and Jenkins in 1970 with the introduction of $SARIMA(p,d,q)(P,D,Q)^s$ in order to manage the seasonality components of time series, by introducing another differentiation adding other parameters:

$$\phi_P(\Delta^s)(1-\Delta^s)^D z_t = \epsilon_t \phi_q(\Delta^s) \tag{2.14}$$

- $\phi_P(\Delta^s)$ is the Seasonal AR.
- $(1-\Delta^s)^D z_t$ is the Seasonal Differencing.
- $\phi_q(\Delta^s)$ is the Seasonal MA
- ϵ_t is the noise.

with the SARIMA model, we can handle the non-stationary time series and the seasonal components.

Chapter 3

Problem Setting

In this chapter, we will briefly talk about the reasons that convinced us to investigate the use and the importance of time series in the world of Deep Learning.
Furthermore, we will explain the different application areas this type of solution can be applied; finally, we will discuss the use case that the solution presented in this work tries to solve.

3.1 The importance of Time-Series Forecasting

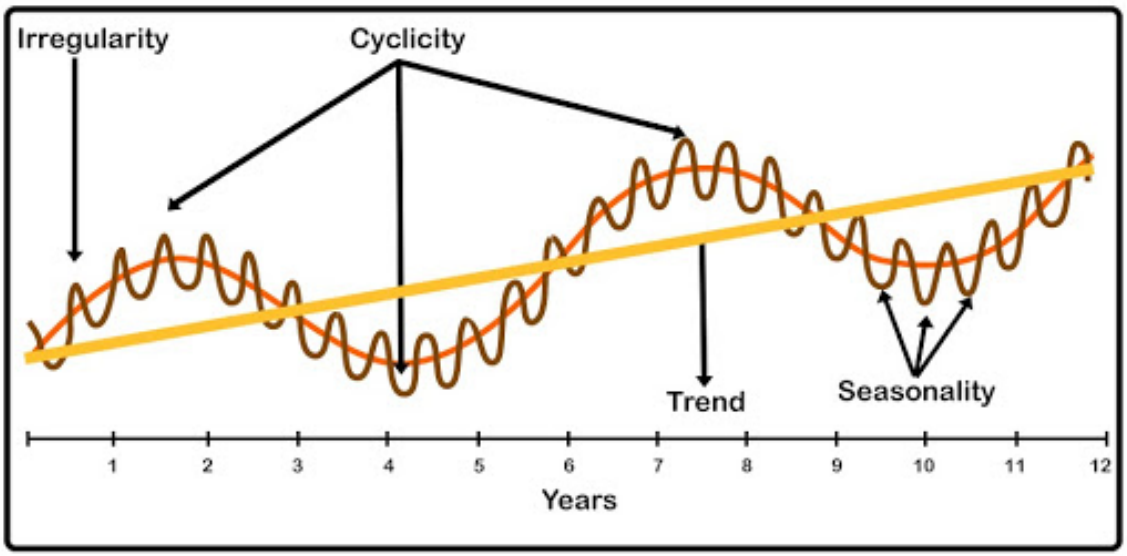
When we talk about Artificial Intelligence, in the collective imaginary, we often think about the "prediction of the future": this is probably a result that we could achieve in the future. Today, Time Series Forecasting are the closest models to this ideal.
These solutions' objective is to predict a datum, such as a value or a class, knowing the data's history in the considered field.

To better understand these models' functioning and how they evolved until today, we have to define what a time series is: a time series is a sequence of data taken at a given time interval. By analyzing these data, it is possible to obtain various information, such as trends, cyclicity, seasonality, and irregularities. All this information allows us to understand and model which factors can influence the variation of the value considered and predict future points.

In literature, this type of information is called *pattern*. In order to improve the understanding of the differences between them, we can define different

type of patterns (3.1):

Figure 3.1: Derivable patterns from Time Series.



- **Trends:** a trend is defined as an increase or decrease in a value over a given period of time. This variation can occur in a *deterministic* way; therefore, it has a reasonable cause and can be considered useful information, or *stochastic*, so we cannot consider it for our models.
- **Seasonal:** often, seasonality is a fixed and know type of frequency. For some types of data, such as the one considered in this thesis, there is the possibility that it influences the value trend. Therefore it is appropriate to consider the seasonality data: in these cases, we talk about seasonal patterns, which are sometimes also called *periodic* time series.
- **Cyclic:** a cyclic pattern exists when data exhibit rises and falls that are not of a fixed period.

The single measure can be defined as follows:

$$y(t) = T(t) + S(t) + C(t) + \epsilon(t) \tag{3.1}$$

$$y(t) = T(t) * S(t) * C(t) * \epsilon(t) \tag{3.2}$$

Where $y(t)$ represents the measure at time step t, $T(t)$ is the overall trend of the series, $S(t)$ the seasonal aspect of the time series, $C(t)$ describes the

cyclic component of the observation and finally, $\epsilon(t)$ represents the irregular pattern within the series, called residuals. The two equations describe both the additive and multiplicative decomposition.
The first is more appropriate if the magnitude of the seasonal fluctuations does not vary with the time series level; the second one instead has to be used when the variation appears to be proportional to the level of time series. With the economic time series, multiplicative decompositions are the best.

The analysis of a time series could be done differently for different objectives; this is why time series models occur in many fields and got a wide application in several areas, such as economic forecasting, population studies, biomedical science, quality control, and many more.

3.2 Price Forecasting

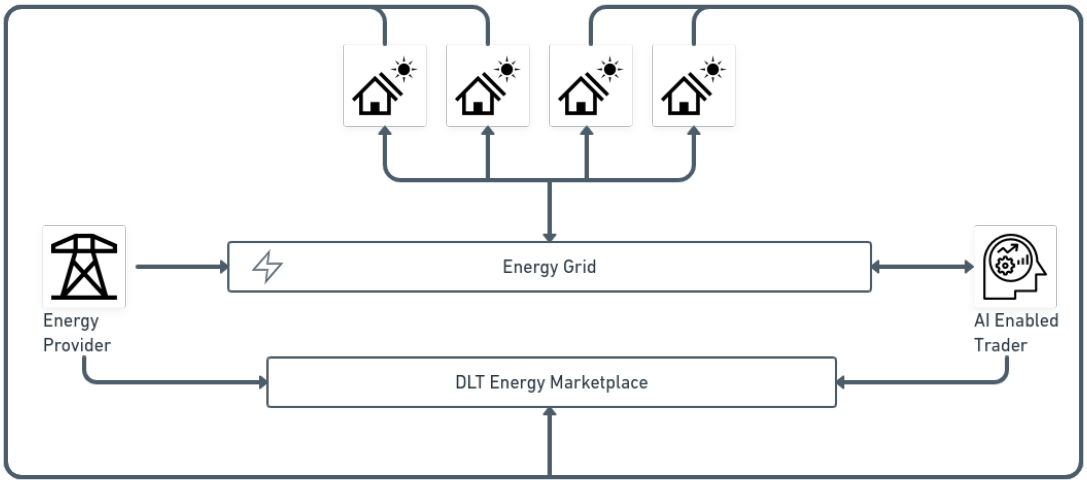
Today, the price prediction field is one of the most important in AI, both from a theoretical and a practical point of view. Indeed, the interest of equity market investors towards this type of solutions has increased more and more in recent years, as not only prediction models are capable of storing and processing more data that a person can manage, but also a lot of time and money is already spent in order to collect these data; an AI-based tool can be able to find correlations otherwise impossible to see in the same time frame.
On the other hand, from the researcher's point of view, this field offers many challenges that improve the quality of predictions more and more. This is because, in the financial time series prediction process, the stock market is essentially dynamic, nonlinear, complicated, nonparametric, and chaotic [1]. Besides, there are market-affecting factors that are difficult to both track and predict, such as political events, general economic situation, rising interest rates, investors speculations, institutional investors choices, and even psychological factors of investors.
The sum of this and the investments in the area mean that stock prediction is and will be a field that, in the future, will have greater importance.

3.3 Problem Formulation

The use case that the solution presented in this thesis tries to solve is the automatic management of the energy present in a grid.
In fact, in the structure of the case considered (3.2), there were several ele-

ments:

Figure 3.2: Households grid structure.



- **Household:** house equipped with solar panels.
- **Energy Provider:** energy provider company.
- **Energy Grid:** energy collection and distribution grid. The most important mechanism to underline here is that the grid is not able to store energy.
- **DLT Energy Marketplace:** element whose function is to manage the transaction of energy. Here, the energy provider can buy surplus, and the grid can buy energy when in deficit. This element is not directly connected to the Energy Grid.
- **AI Enabled Trader:** intelligent element capable of predicting the price trend energy from the data for the next two days, and between the predicted values, prepare purchase/sale for the DLT energy marketplace.

The procedure that follows the system, of which we can find a graphic example in figure 3.3, is the following:

1. The data present in the DLT Energy Marketplace are extracted and contain the requests for the sale of surplus or energy requests from

households. In this case, we have two types of agreements: the Energy Provider buys surplus, and the households buy energy from Energy Provider.

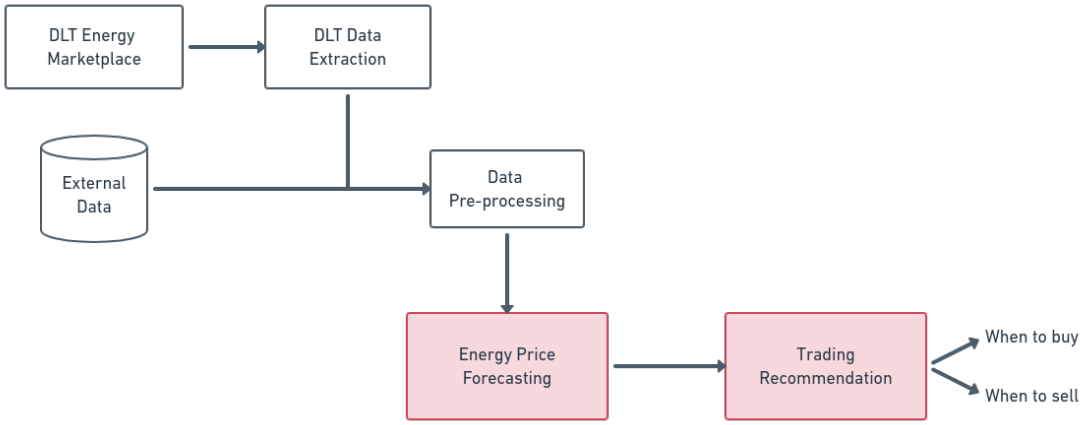
2. In the External Data source (for more detail about all the data, see 5), the information referring to the historical weather data and the future climate forecast are downloaded.

3. The data is then pre-processed to be inserted into the network. Here, in addition to aligning all the data according to the time span considered, the normalization, the transformation into tensors, and the different sources' concatenation is also performed.

4. Once the data is ready, it is fed into the network, and predictions for the next two days are calculated (the detailed structure explained in 4).

5. Based on the predicted values, one or more recommendations are made. They can be of two types: when to buy or when to sell.

Figure 3.3: Learning from DLT to recommend energy stocks.



A system with this type of automation has several advantages. First of all, remembering that the energy grid cannot store energy, selling the surpluses reduces waste. Furthermore, there is also an advantage in monetary terms: both the sale and the purchase will be made when the maximum and minimum peak of the price curve, respectively.

Chapter 4

Solution Description

In this chapter, the core architectures and technologies that have been used for the development of the solution presented in this work are reported.

4.1 Baselines

4.1.1 Facebook Prophet

Facebook Prophet [12] is a procedure for forecasting time series data developed and open-sourced by Facebook in 2017. It is developed both in Python and R (a "data science" programming language) and has the advantage of proposing a series of parameters that are easy to understand and use to improve results: for this reason, even people without in-depth knowledge of the models used for the analysis of time series have the opportunity to approach this area of application. In addition, this model is robust when there are missing data or outliers.

The operating principle of this model is based on additive regression models, in which seasonal components play an essential role. The equation that puts them together is the following:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon t \tag{4.1}$$

- **y(t):** is the forecast.
- **g(t):** represents the trend models, which are non-periodic changes. In order to provide more flexibility, Prophet relies on Fourier Series to model the seasonality effects.
- **s(t):** these are the periodic changes, such as week, months, and years.

- **h(t):** defines the effects of holidays with irregular schedules, and it is a user-provided list because holidays and events are an essential factor that has to be taken into consideration depending on the place considered.

- **εt:** any changes which are not detected by the models represent the error term.

Even if the models can be written or extended (according to specific requirements), Facebook Prophet standardly provides two models. The first one is the Logistic Growth Model, which is particularly effective with non-linear data and it is described by the following equation:

$$g(t) = \frac{C}{1 + e^{-k(t-m)}} \tag{4.2}$$

Where C is the carry capacity, k is the growth rate and m is an offset parameter.

The second model, used by default by Facebook Prophet, is the Piece-Wise Linear Model, best option to choose when data shows linear properties and fitted by the following statistical equations:

$$y = \begin{cases} \beta_0 + \beta_1 x & \text{if } x <= c \\ \beta_0 - \beta_2 c + (\beta_1 + \beta_2)x & \text{if } x > c \end{cases} \tag{4.3}$$

We have decided to use Facebook Prophet as one of the baselines because it fits our problem particularly: in fact, market variations often follow the seasonal trend, and it is easy to find other cyclic patterns.

4.1.2 XGBoost

XGBoost, the acronym for eXtreme Gradient Boosting, is a widely used supervised learning algorithm in machine learning. It was developed as a research project at the University of Washington in 2014 and presented at the SIGKDD (Special Interest Group on Knowledge Discovery and Data https://www.kdd.org/) conference by Tianqi Chen and Carlos Guestrin. This algorithm has gained popularity not only because it allows us to obtain excellent results and performances but also because its implementation brings with it several advantages, such as:

- Portability: it runs smoothly on the main operating systems, such as Windows, Linux, and OS X.

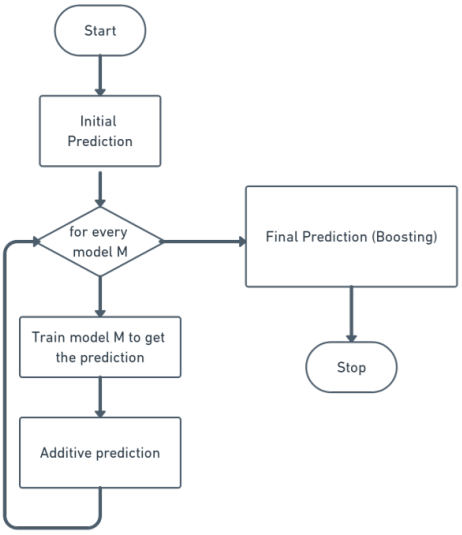
- Languages: it supports the main programming languages.
- Applications: it can be used to solve a wide range of problems, such as regression, classification, and forecasting problems.
- Cloud: it supports AWS clusters, Azure, and similar ecosystems.

XGBoost is a particular implementation of the Gradient Boosting model [7] with the peculiarity of the focus on the speed and the algorithm efficiency; in fact, all the operations can be parallelized, and this is the main reason why this library is one of the most used in the context of Data Science challenges.

The heart of the operation of this algorithm is the decision tree model. It has a tree structure, where each of the leaves corresponds to a class, and each node represents an attribute. On the other hand, the concept of boosting refers to the mechanism of converting weak models to strong ones through its combination and doing the training sequentially so that in each iteration model tries to correct the error done in the previous iteration. Finally, Gradient Boosting is a technique wherein each of the interactions the new predictor is constructed to fit on the pseudo-residuals of the previous predictor.
The schema of the solution can be find on 4.1. Here, we can find this operations:

1. The Mean of target values is calculated from the initial predictions and the corresponding initial residual errors: $p_o(x) = Mean(value)$ and $predicted_value = value - p_o(x)$.
2. Each model M is trained with independent variables and residual errors as the data to get the predictions.
3. The additive predictions and the residual errors are calculated with some learning rate from the previous output predictions obtained from the model.
4. Computation of the prediction value at i-th iteration: $p_i(x)$
5. Additive prediction is done by the sum of the previous predictions and a parameter α called Learning Rate: $F_i = h_o(x) + \sum_i h_i(x)$
6. The final prediction is the additive sum of all previous predictions made by the models.

Figure 4.1: Flowchart for the Boosting framework.



4.2 Long-Short Term Memory

Long-Short Term Memory (LSTM) networks are a special architecture of Recurrent Neural Networks (RNN) designed by Hochreiter and Schmidhuber in 1997 and set accuracy records in multiple applications domains [8]. LSTMs were specifically designed to overcome the long-term dependency problem faced by recurrent neural networks RNNs due to the vanishing gradient problem. This problem is often encountered when training neural networks where the weights are updated by a value that is proportional to the partial derivative of the error function with respect to the current weight in each iteration of the training; in some cases, this value (called gradient) can result in a vanishingly small amount, and in the worst case it can stop the neural network from further training.
The LSTM structure implements feedback connections in addition to the feedforward ones of the classic RNN; in this way, each of the data present in the sequence is not treated independently concerning the other points, thus allowing to predict the following data while maintaining useful information from the previous ones. This mechanism is why LSTMs are mainly used to analyze any data sequence, be it time-series, text, or speech.

Let us now go into the details of its operation: the main heart of an LSTM cell (Figure 4.2) is composed of the state and the gates: the state is the element that allows preserving the information along with the different it-

erations of the sequence processing, effectively acting as a "memory "of the network. On the other hand, Gates serve to discriminate which information must be saved or deleted from the state. The gates are based on the Sigmoid activation function (in the figure, with letter S), which allows you to normalize the values between 0 and 1: if the information is forgotten, it will be multiplied by 0. The other activation function present in an LSTM cell is the hyperbolic tangent (tanh, in the figure with letter T): in fact, the values that are introduced into the network are transformed by mathematical operations many times; in order to prevent the explosion of the magnitude of the data, we use the tanh function to keep the values between -1 and +1. The graphs of both these functions can be seen in Figure 4.3.

Figure 4.2: Overview of the LSTM cell structure.

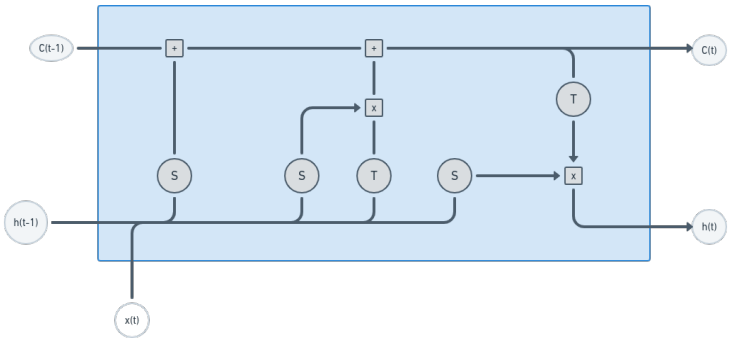
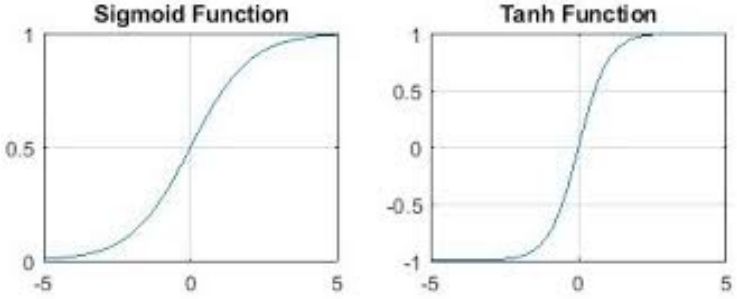


Figure 4.3: Hyperbolic tangent and Sigmoid activation functions graphs.



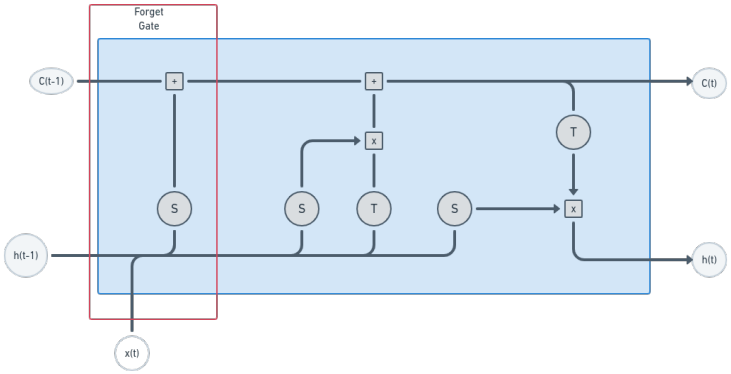
4.2.1 Forget Gate

Once the two input parameters, namely our input at time t and our output at time $t-1$, have entered the network, the first element they encounter

is the forget layer ref figure: 4.4 . In this phase, the decision taken by the network is to "remember" the important elements and "forget" those that are not: this operation is done through a Sigmoid activation function, whose outputs are values between 0, which means "forget this element" and 1, which instead means "keep this element": these values are stored in a variable called f_t (Figure 4.4):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{4.4}$$

Figure 4.4: Forget Gate.



4.2.2 Input Gate

The input data, which we underline to be the same processed by the Forget Gate, are passed in input to two other elements, whose interaction we need to calculate which new information must be saved in the state of the cell (Figure 4.5):

1. New Memory Vector: the purpose of this element, whose equation is the following:
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4.5}$$
is to generate the vector for updating the state of the cell, combining the previous state and the current inputs: it is done through an activation function tanh (whose values range from -1 to +1), as possible negative values can reduce the impact of specific components on the cell state

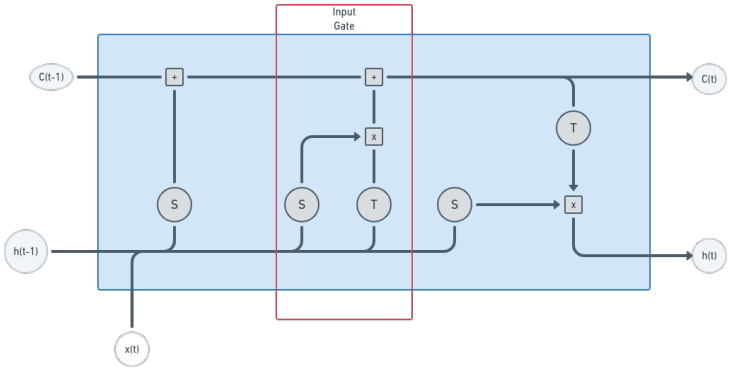
2. Input Gate: since the values processed by the New Memory Vector do not check the actual importance of the data, we need an element

that does this work: it is the input gate, which is a Sigmoid activated network that acts as a filter, whose operating principle is the same seen and described in the Forget Gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.6}$$

3. Ensembling: the outputs of the two blocks described so far are put together through a pointwise multiplication. Therefore, the values decided by the New Memory Network are regulated by those of the Input Gate. This operation gives us the values that allow the updating of our long-term memory (cell state).

Figure 4.5: Input Gate.

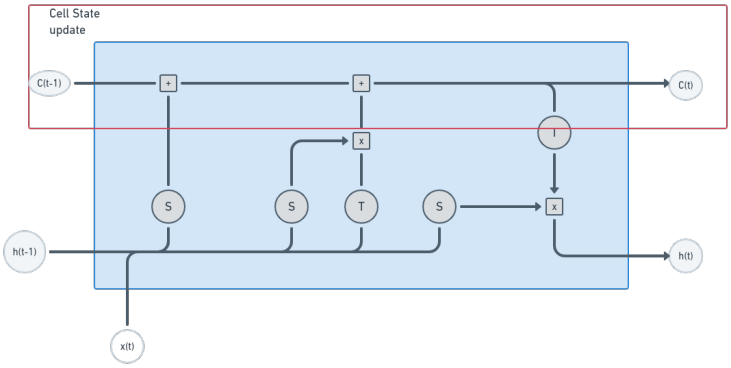


4.2.3 Cell State Update

The information we have calculated up to this point is enough to perform the status update operation. This operation brings together the results of the previous ones, so we have that the new state is given by the sum of that of the previous iteration, of which we have forgotten the unnecessary information, and the important elements calculated from the inputs of the current iteration (Figure 4.6).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.7}$$

Figure 4.6: Update of the Cell State.



4.2.4 Output Gate

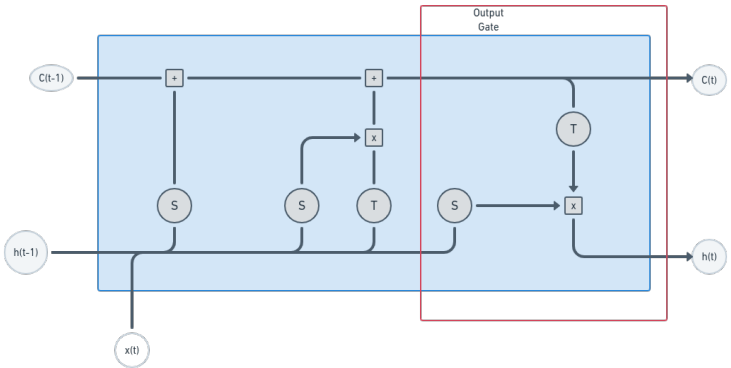
The final operation is to decide what the output of the current iteration will be. To do this, first of all we have the Output gate, which works like the input gate and the forget gate, and which serves to filter the input data, and which therefore is always a Sigmoid. However, this filter is now applied to the new state of the cell to make sure that only the necessary information is sent to the output.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{4.8}$$

However, before applying the filter, we make sure that the data is normalized between -1 and +1 via an Hyperbolic Tangent activated network. The multiplication of these two results (o_t and $tanh(C_t)$) results in the new hidden state.

$$h_t = o_t * tanh(C_t) \tag{4.9}$$

Figure 4.7: Output Gate.



4.3 StackedLSTM

The solution we have explored and proposed does not modify the basic structure of the LSTM explained in the previous section but modifies how the data is inserted into the cell. The phenomenon that we have noticed is that by inserting a huge set of data into the network, the importance of the information relating to the price was lost as a whole. This behavior caused very wrong predictions.

To overcome this, we decided to manipulate the external data we wanted to use for price prediction. First, we took the climate data and brought it into a time interval comparable to what we had for price and energy consumption: in this case, the information on solar exposure and rainfall was given to us on a daily basis, which introduced some noise into the data and which can certainly be a point that, by being able to collect data every 30 minutes, can improve the accuracy of predictions.

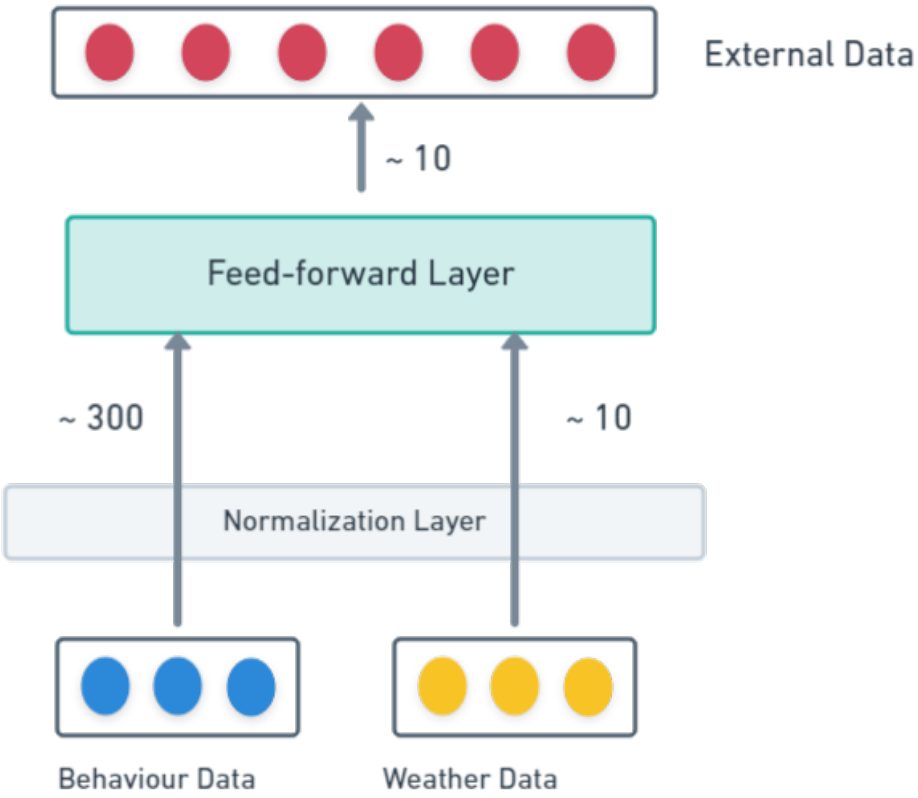
In any case, the second operation carried out on the data is on those of energy consumption: from the dataset in our possession, every 30 minutes, we had the information of the energy produced or consumed in excess, called respectively surplus and deficit. Since this is an exclusive type of data, i.e., if there is a surplus, there is no deficit, and vice versa, what we did was select only the useful information. Once the information was collected, we normalized it according to the following formula:

$$X_{standardized} = (X - X_{min})/(X_{max} - X_{min}) \tag{4.10}$$

In this way, all the input data are brought in a value range between 0 and 1, without modifying its distribution. It is important to remember that the use of a MinMaxScaler, like the one done for external data and price values, allows us to normalize each feature's data independently. Furthermore, since the transformation is done using the maximum and minimum of the distribution, particular attention must be paid to the outliers, which can introduce unpleasant errors: in this case, it is advisable to clean the data, or use tools such as the RobustScaler, which scales the values using statistical operations that are robust to outliers, such as not using the average value but the so-called quantile range (IQR = the range between the 1st quartile and the 3rd quartile).
The rationale behind standardization/normalization operations is that having many features, usually measured with different scales of values, cannot contribute to the model building, in the same way, creating potential bias.

Furthermore, during the backpropagation operation, having scaled data allows to speed up and stabilize the calculations compared to using non-scaled ones.
All the values thus obtained are then concatenated, and we called them *externaldata*. The procedure is schematized in Figure 4.8. The last thing to do is pass this data into a Feedforward layer: this is done because a Feedforward layer has the goal to approximate some function. In our case, we gave as input all of our data, with an input size of 310, and output a vector of size 10, which keep being representative of all the information.

Figure 4.8: Concatenation and normalization process.

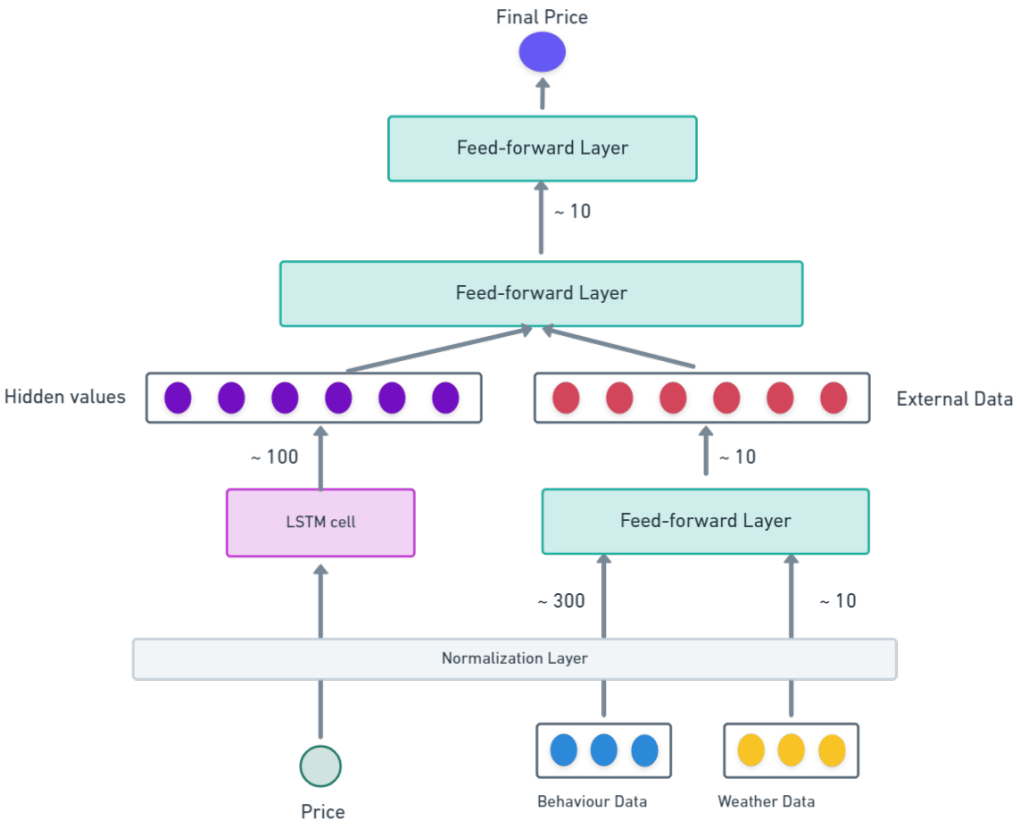


The tensor containing the external data is inserted in the forward phase of the LSTM after being concatenated with the hidden values tensor generated by the LSTM cell. This tensor has a size of 100 (a value that we found to be an excellent trade-off between results and performance) starting from

input price. In this way, the price information is preserved without giving up the important information given by external data.

However, having a total dimension tensor of 110, we need to reduce the dimensionality as we had previously done with the external data. This is done using two Feedforward layers to obtain a single output value, representing our predicted price. We have inserted two layers in this case because passing from a dimension 100 to 1 in a single Feedforward would probably have caused errors due to the impossibility of approximating a vector of more than one hundred elements in a single value: therefore, we have the first Feedforward that transforms the vector from 100 to 10, and then the second that further transforms it from 10 to 1. We can see the complete structure designed in the Figure 4.9.

Figure 4.9: Stacked LSTM structure schema.



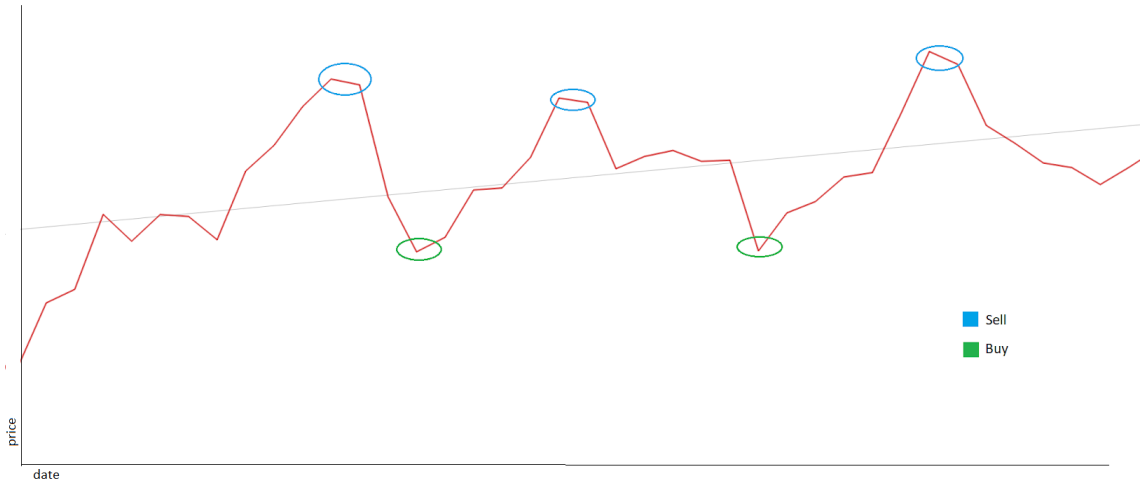
4.4 Trading System

To complete the solution to the problem presented, we implemented a system of analysis and selection of time windows to recommend purchasing or selling energy stocks (Figure). We consider the data generated by the Energy Price Predictor: among them, we can find maximum and minimum peaks, which correspond to the main points where sales or purchases can take place, respectively. However, we must consider a significant factor: which of these periods to select? We have said that energy cannot be stored in the grid for very long and that more supply is needed when the grid is empty. For this, the selection of the period must also take into account the data relating to energy consumption.
In our case, what we did was take the data relating to all the households and, for every 30 minutes, calculate the total energy value present in the grid. Therefore, considering this value and the possible transaction periods, we choose the most suitable one.

Finally, the predictor and this trading system were packaged together in order to be called through the REST API:

- predict: an object is instantiated that represents our Energy Price Predictor which, starting from the selected period, loads the reference trained model and predicts the energy price for the next two days.
- bestBuyStock: select the best periods to make a possible purchase from the results mentioned above.
- bestSellStock: select the best periods to make a possible sell from the results mentioned earlier.

Figure 4.10: Example of possible choices for the purchase or sale of energy stocks.



Chapter 5

Dataset

In this chapter, we will describe the datasets used in detail, along with some statistics that allowed us to fully understand the data. For the work described in this thesis, we used multiple datasets, which includes different types of data and information:

- A dataset containing the historical series of the price of energy.
- A dataset containing weather information.
- A dataset containing the information of the energy produced and stored by different households in Australia.

5.1 Energy Price Data

This dataset, which can be viewed and downloaded via the address http://www.nemweb.com.au/#mms-data-model, collects the history of electricity prices. The range of period that has been analyzed goes from 01/01/2010 to 31/12/2013.

The downloaded dataset had a lot of information, such as looking at the price in different currencies and different geographic areas. Still, many of them have not been used, which can be a useful source of data for this work's future extensions.
The data relating exclusively to the $NSW1$ area has been considered by us, which corresponds to the southeast of Australia, where we have the households' data. Furthermore, we also took the price of energy in dollars. All these information are listed in table 5.1.

Finally, there is also a small consideration to make: what we are discussing in this thesis is the price of solar energy; this value depends on many factors, some of which we have taken into consideration because they directly affect the price, such as the weather data. There are many other factors that influence its trend indirectly, such as the price of oil, a significant raw material in the modern market; a possible addition that can be made to this work is to feed the network with as much information as possible, having an awareness of selecting the most correlated one to avoid that the network does not know how to recognize the essential elements useful at the moment of prediction.

DVD TRADINGPRICE	SETTLEMENT DATE	REGION	RRP
PRICE	2010/07/01 00:30:00	NSW1	29.36
PRICE	2010/07/01 01:00:00	NSW1	29.26
PRICE	2010/07/01 01:30:00	NSW1	32.62
PRICE	2010/07/01 02:00:00	NSW1	31.89
PRICE	2010/07/01 02:30:00	NSW1	28.88

Table 5.1: Example of the Energy Price dataset.

5.2 Weather Data

This dataset, which can be found at the following address http://www.bom.gov.au/climate/data/, contains data relating to the weather conditions in the South West of Australia.

The data was loaded from two *.csv* files, containing information about sunny days and the other containing information relating to rainy days. In both of them, the time span considered is the day: this has created some discrepancy on the time base considered for the price of energy and the behavior, which is per half an hour. A more granular dataset in line with the quantization of the other evaluated data could increase the efficiency of the models that have been described in chapter 4.

Climate data are vital in this context. The energy produced by the houses considered derives from photovoltaic plants, which make solar exposure the necessary condition to produce more energy.
This premise leads to the following aspect: on a rainy day, since the energy produced is naturally deficient, the energy consumption will likely be higher than the one produced. Consequently, it may be necessary to purchase more. On the contrary, on a very sunny day, we can have a surplus of energy, and

in this case, we can sell this surplus. These operations have an impact, albeit minimal, on the variation of the energy price.

In table 5.2 and 5.2, we have a small extract of the data contained in the solar exposure and rainfall datasets. The data are divided as follows:

- **Product Code:** it is a datum inserted at the moment of data acquisition, irrelevant for the work presented.
- **Bureau of Meteorology Station Number:** although it is a data not considered by the architectures presented in this thesis, it indicates the data acquisition station, which therefore was geographically located in the area considered, that is the southeast of Australia.
- **Year:** indicates the year
- **Month:** indicates the month.
- **Day:** indicates the day.
- **Daily Global Solar Exposure (MJ/m*m):** the solar exposure values of the considered station.

Product Code	Bureau of Meteorology Station Number	Year	Month	Day	Daily Global Solar Exposure (MJ/m*m)
IDCJAC0016	66037	2010	1	1	141
IDCJAC0016	66037	2010	1	2	158
IDCJAC0016	66037	2010	1	3	81
IDCJAC0016	66037	2010	1	4	143
IDCJAC0016	66037	2010	1	5	291

Table 5.2: Example of the Solar Exposure dataset.

5.3 Household Behavior Data

Through this dataset (available at the following address https://www.ausgrid.com.au/Industry/Our-Research/Data-to-share/Solar-home-electricity-data#.Vl-RUhNCpOs), we have access to information concerning the behavior of a group of 300 randomly selected households that have rooftop solar systems

Product Code	Bureau of Meteorology Station Number	Year	Month	Day	Rainfall Amount (mm)	Period	Quality
IDCJAC0016	66037	2010	1	1	0.0	1	Y
IDCJAC0016	66037	2010	1	2	0.0	1	Y
IDCJAC0016	66037	2010	1	3	11.6	1	Y
IDCJAC0016	66037	2010	1	4	1.2	1	Y
IDCJAC0016	66037	2010	1	5	0.0	1	Y

Table 5.3: Example of the Rainfall dataset.

and a gross metered solar system installed. So, we have, for every 30 minutes, a value that indicates whether we are in surplus, which means the use of energy is lower than the produced one, or in deficit. Therefore it means that the value of energy production is lower than the used one.

We have two files: *surplus.csv* and $deficit.csv$, which already indicates the type of data we are evaluating. In the table 5.3 and 5.3 we have a small excerpt of the elements considered. In particular:

- **Household:** indicates the house taken into consideration, of a total of 300.
- **Datetime:** indicates, in $YYYY-MM-DDHH:MM:SS$ format, the date to which the data refers.
- **Net:**
- **Value:** indicates numerical value of the surplus/deficit.

Obviously, for the same DateTime, we have only one data in one of the two files.

5.4 Data Statistics

All the analyzed data were not chosen by us after a careful review of the literature, but followed a process similar to notarization so that the data could not be modified. For this reason, during a first preliminary analysis, we noticed no strange behaviors such as lack of values (encoded in pandas with NaN Not a Number). However, there is the presence of some outliers, which are points that have a value that is not in line with the distribution

Household	Datetime	Net	Value
h1	2010-07-01 00:30:00	-4.072	-0.11955392
h216	2010-07-01 00:30:00	-0.13	-0.0038168
h215	2010-07-01 00:30:00	-4.378	-0.128538
h214	2010-07-01 00:30:00	-0.126	-0.00369936

Table 5.4: Example of the deficit dataset.

Household	Datetime	Net	Value
h146	2010-07-01 01:00:00	0.011999	0.0003511
h146	2010-07-01 01:30:00	0.038	0.001239559
h146	2010-07-01 02:00:00	0.076	0.00242364
h146	2010-07-01 02:30:00	0.158	0.00456304

Table 5.5: Example of the surplus dataset.

being analyzed. As we can see from Figure 5.1, where the outliers have already been eliminated, the prices that rise above the value 200 and below 0 are 138 on the total number of 52608 datapoints, corresponding to 0.2%.

Furthermore, from the data relating to the weather, we note how the two distributions of rainfall and solar exposure are complementary (Figures 5.2) while the remaining information are all related to better specify the date considered (and they are $hour$, $dayofweek$, $quarter$, $month$, $year$, $dayofyear$, $dayofmonth$, $weekofyear$)

Finally, we have the data relating to energy consumption, which are exclusive, so they will be shown in a single graph where they are combined to calculate the total contribution. These data can be viewed in the Figure 5.3. We can observe that almost all of the data shows deficits, and this is due to the fact that the energy produced by the houses is not enough for the sustenance of the group of households considered. With more energy produced, managing the grid becomes more challenging.

Figure 5.1: Distribution of price data.

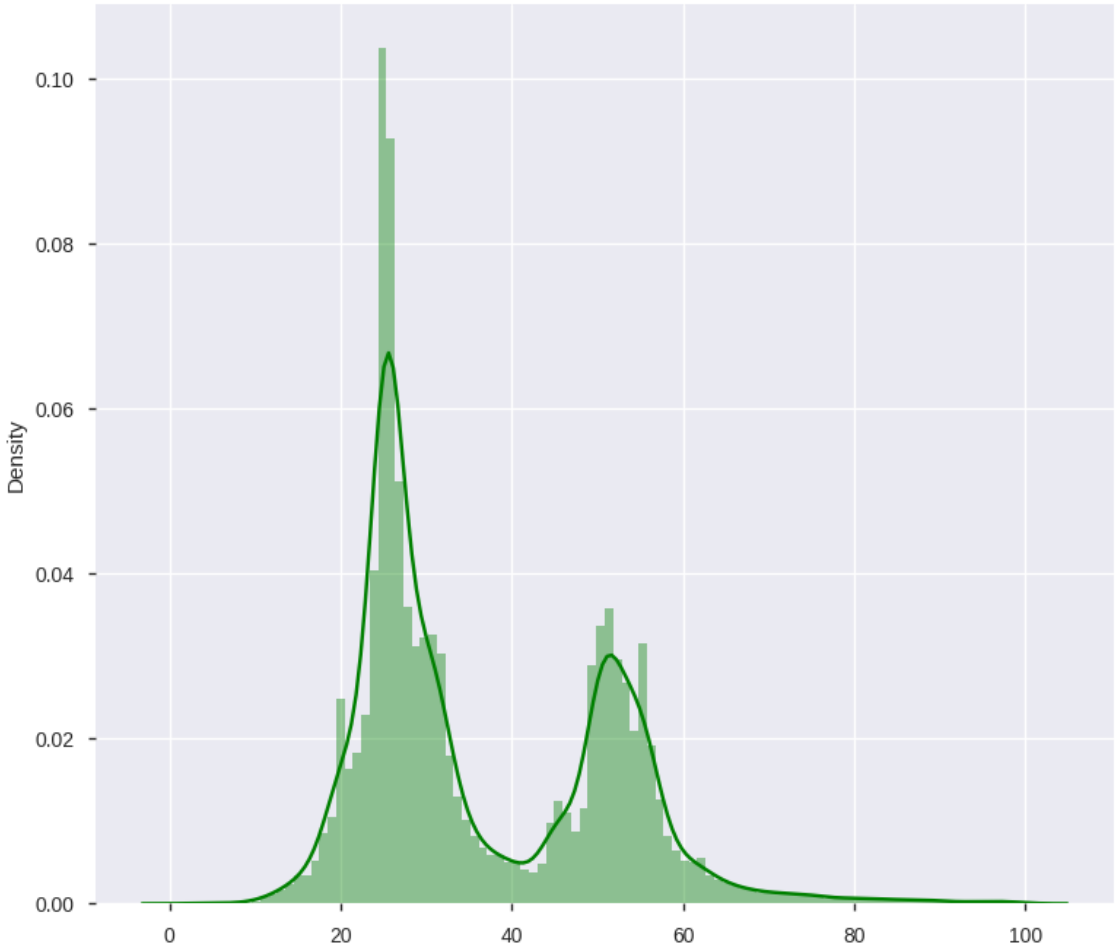


Figure 5.2: Distribution Daily global solar exposure (MJ/m*m) and Rainfall amount (millimetres).

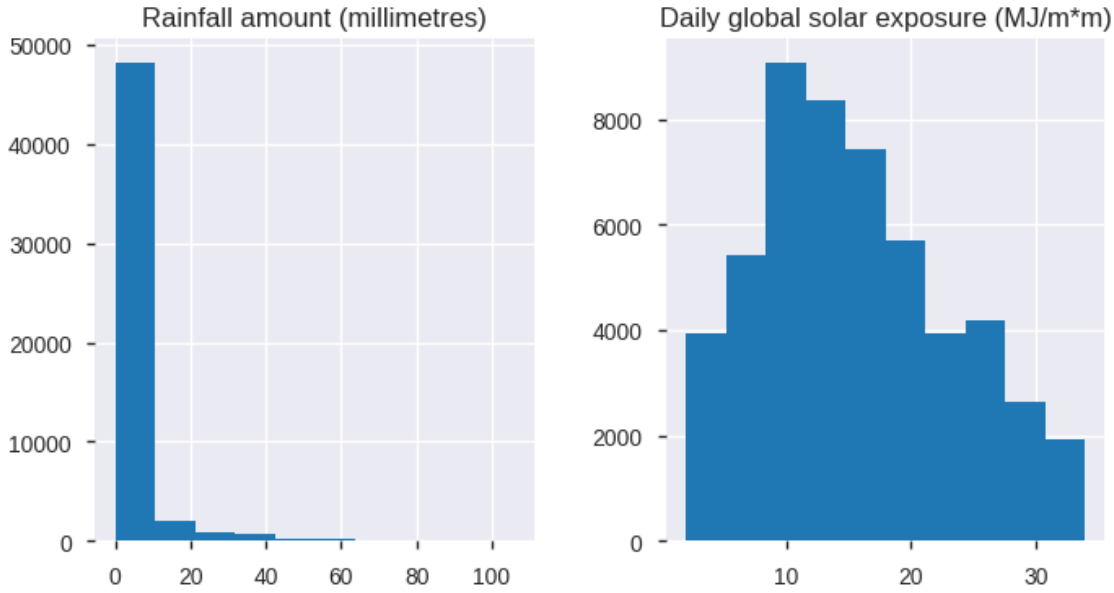
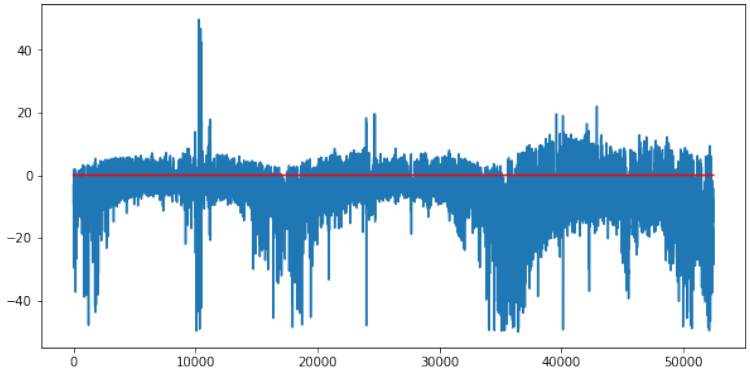


Figure 5.3: Distribution of energy consumption.



Chapter 6

Experimental Setup

In this Section, we will discuss the tools we used to implement the solution designed in section 4, together with the reasons for which we chose them, as well as the methods used to evaluate the results we obtained.

6.1 Framework and libraries

The programming language we chose for our experiments is Python. The reasons for this choice are several: first of all, it provides excellent libraries for big data management and analysis, such as Pandas (Subsection 6.1.1). In addition, Python offers a lot of advanced frameworks for Deep Learning, such as PyTorch or TensorFlow (Subsection 6.1.2). Finally, it is an easily approachable language thanks to its simplicity regarding the syntax and the management of the various entities, so this language is gaining more and more popularity. However, compared to other languages, it certainly does not shine for its optimization of resources management.

6.1.1 Pandas

We made extensive use of the Pandas library, an open-source library released in 2009 and based on data analysis for the development of our solution. We used Pandas to import our datasets into structures called DataFrames: they wrap several file formats, such as CSV, JSON, and Microsoft Excel, in a row/column table, allowing us to apply a large number of useful functions, such as sorting data with respect to values of a specific column, the application of lambda functions by single row or operations of grouping data by columns.

In addition to this, Pandas is incredibly easy to use and implement a

very useful optimization in data management: the modification of the data is done by parallelizing the operations, which excludes the need to sequentially iterate all the data and therefore makes the computations decidedly faster.

All these advantages make this library excellent for time series management, which is the subject of this work, as well as being the main reason why we opted for its use.

6.1.2 PyTorch

The advent of Deep Learning led to the need to have frameworks that can support creating the architectures necessary for both applicative and research fields. For this reason, tech giants like Facebook and Google released several frameworks making it easier to learn, build, and train different types of neural networks.

TensorFlow is an open-source framework released by Google in 2015 (https://www.tensorflow.org/) widely used by companies and startups and consists of two main blocks:

1. A library that has the purpose of defining computational graphs, together with mechanisms that allow the execution on different hardware configurations.

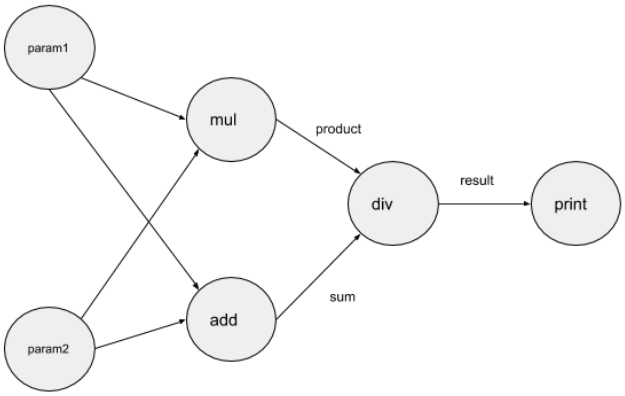
2. A computational graph.

A computational graph is an abstract way to describe the flow of execution of a program through a directional graph. Graphs, born from graph theory [5], are structures composed of two elements: the nodes, which represent various entities, and the edges, which have the function of relating two nodes; the edges can be with or without verse. In a computational graph, nodes are variables or operations that have to be performed. At the same time, edges define the inputs of the nodes, which can be either a variable or the result of another node. When we run code in TensorFlow, the computation graph is defined statically [9]. For example, let's consider these simple operations:

```
param1 = 5
param2 = 4

product = param1 * param2
sum = param1 + param2
result = product / sum
print(result)
```

Figure 6.1: Result of the static generation of a computational graph. Here, we can observe what defines nodes and edges and how the workflow is structured.



Then, the statically generated computational graph will be as follows:

The advantage of this mechanism is parallelism, which makes training faster and more efficient. On the other hand, to use parallelization, we have to code and fine-tune every operation manually. Moreover, everything related to the neural network must be declared a priori, and TensorFlow does not support the debugging with the extremely powerful Python tools and does not allow to add modularity through modules, making the creation of a new project slow and repetitive.

PyTorch is one of the latest deep learning frameworks, developed by Facebook and open-sourced in 2017 (https://pytorch.org/). Similar to TensorFlow, PyTorch is composed of two main blocks:

- Dynamic building of computational graphs.
- Autograds.

The *autograd* package is the main core of all neural networks written with PyTorch: it provides automatic differentiation for all Tensors operations. These structures are a generalization of those usually defined in linear algebra. *torch.Tensor* is the central class of the *autograd* package. In this way, the computational graph is generated dynamically during execution; that is, the instantiation of the graph is done whenever the $forward$ method is called, making the application much more scalable and research-oriented.

Moreover, the implementation of PyTorch is very connected with the Python language, sharing the same advantages described at the beginning of this Section. Finally, it also supports modules and the use of multiple CPUs and GPUs, along with good documentation and plenty of existing projects made with the support of PyTorch. These reasons led us to decide to adopt this as the framework for the development of our solution.

6.2 Google Colaboratory

Given the nature and the size of the dataset, the computing resources we needed were not particularly expensive. Since we did not have to rely on extremely powerful calculators, we decided to take advantage of a free to use product released by Google called Colaboratory. Colab is based on the open-source project Jupyter; the code is executed on a remote virtual machine connected with your account. It can be modified directly online and subsequently downloaded if you want to have a local copy. In this regard, both GitHub and Google Drive are integrated into Colab, making versioning operations even more straightforward and effective. For what concerns the hardware level, Colab provides limited resources to allow the free use for all people; it uses different GPUs: Nvidia K80s, T4s, P4s, and P100s. Using the resources provided by this tool, we were able to carry out all the experiments without incurring any problems and with good execution times. For these reasons, I thank Google since with this product, they managed to bring more and more people close to the world of Machine and Deep Learning and make this process more very simple thanks to its ease of use.

6.3 Flask

To provide our tool the opportunity to communicate with the outside world, we decided to use Flask. It is a *micro*-framework (which means it is composed of a single core that can be extended) for web development in Python; extensions can add functionalities to an application as if they were implemented by Flask, which in his core still offers several useful tools such as the possibility of making RESTful requests, server and debugger for development and excellent documentation.

We opted for Flask because we needed a light and secure framework for writing API calls to pass the data predicted to the visualization tool created by the Accenture Labs colleagues.

6.4 Evaluation Metrics

In the world of Machine and Deep Learning, the choice of the evaluation metric is fundamental: it allows you to influence how the performance of a model is measured and compared; moreover, they also influence the importance that you choose to give to different characteristics of the model on the results.

6.4.1 Mean Squared Error

The Mean Squared Error (shortened MSE) is calculated using the following formula:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{6.1}$$

Therefore, the sum of the squares of the prediction errors, which are equivalent to the difference between the real values and the predicted one, is divided by the number of predicted data. Using a very large scale of values, it is possible to incur very high MSE values, so the variant Root Mean Square Error is often used, which inserts the root on the MSE.

For The work presented in this thesis, we did not have problems like the one mentioned above (for detailed information about the data used, see the section 5), so we chose to use the classic MSE.

6.4.2 Mean Absolute Error

The Mean Absolute Error (shortened MAE) is calculated using the following formula:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{6.2}$$

Again, MAE is a widely used function over regression problems, and it is calculated as the average measure of the errors (which is the difference between the predicted values and the real ones), regardless of its direction.

The main difference between the two loss functions presented is that in the MSE, having the square of the difference, we have that the higher the error, the higher the MSE value. Therefore, we tell our network that we do not admit values that are very different from the real ones.

6.4.3 Mean Absolute Percentage Error

The Mean Absolute Percentage Error (MAPE) is defined by the formula:

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{6.3}$$

It is calculated as the average of the absolute percentage errors of forecast [11], so the error is defined as the difference between the observed value and the forecasted one; the errors are then summed to compute MAPE.
This measure is easy to understand because it provides errors in terms of percentage and is widely used in forecasting problems. The smaller the MAPE, the better the forecast.

Chapter 7

Results

In this chapter the results of the all the solution we described so far are presented, along with our comments on them.

7.1 Facebook Prophet

The results obtained by the Facebook Prophet implementation are shown in Table 7.1. way: all the data, except for the last 30 days, were used for the training phase of the models. The remaining data were then used for price prediction and assessment of the quality of the predictions. From what we see in the Figure 7.2, we can understand how the quality of the predictions is not excellent, despite the model having a loss with a decreasing trend, an important sign that the model is learning. Although it is possible to carry out further parameter tuning operations or even implement additions such as in the management of timestamps (since Facebook Prophet is unable to manage them and a conversion to a DATE type format is needed), the quality of the observed results may certainly be due to the few data considered. Specifically, considering the size of our dataset, the data we handled turned out to be an adequate number regarding the other solutions we tested, but what distinguishes Facebook Prophet is its strong orientation to model data that are very season-dependent. and cyclicity, information that is more easily obtained by having information available with a broader timespan (of 12 hours or even of the day) but with a much wider time window. As we also see in the temporal components extracted in the Figure 7.1, we can see how there is no clear definition of some types of components and that with more data, it could have been more defined

Figure 7.1: Facebook Prophet components.

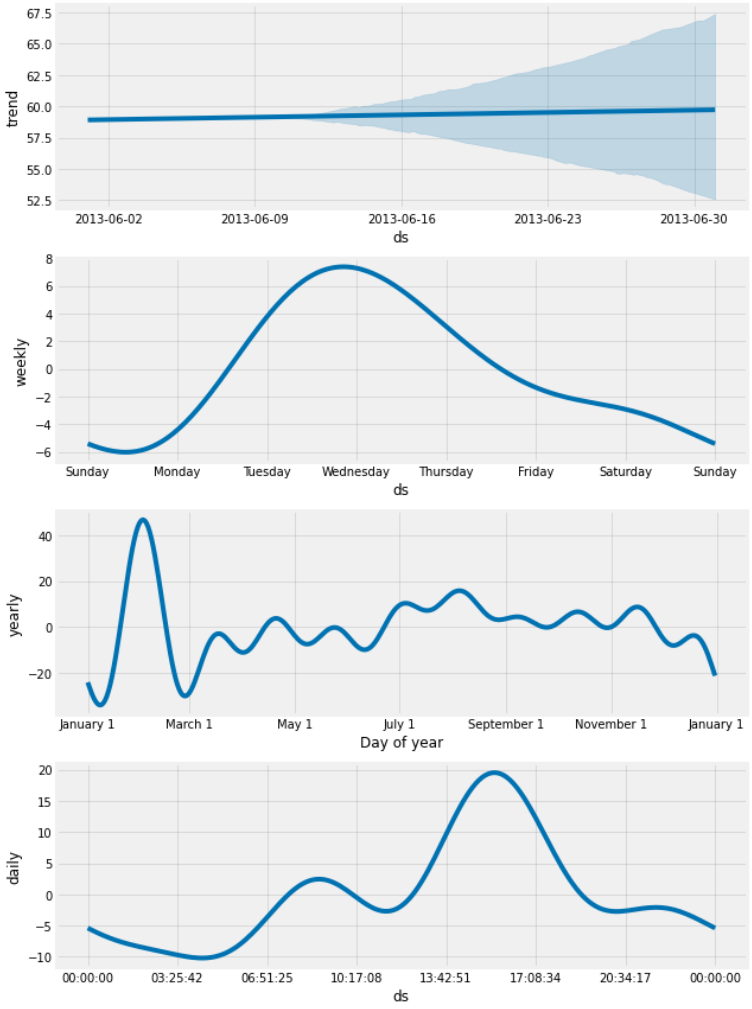
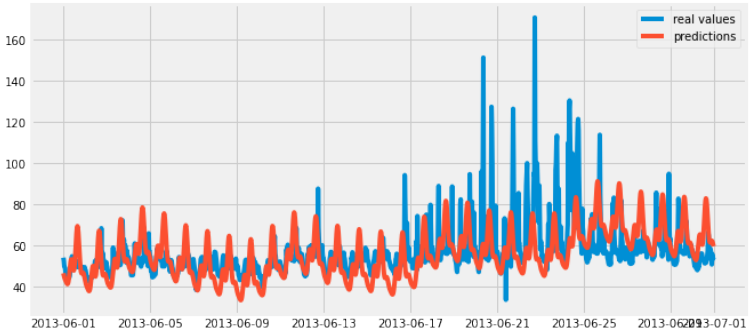


Figure 7.2: Real values compared with Facebook Prophet predictions.



MSE	MAE	MAPE
231.559	11.187	14.924

Table 7.1: Facebook Prophet Results.

7.2 XGBoost

The results obtained by the Extreme Gradient Boosting implementation are shown in Table 7.2. XGBoost also provides us with important information on the features present in the dataset and its importance in price prediction: we can see the scores in the Figure 7.3. From it, we can make some important deductions, such as the poor score for the *month*, *quarter*, and *year* features is probably caused because our dataset is very granular (remember that the price information is recorded with 30 minutes) and so, the importance of information that is more dilated over time is lost. In confirmation of this assumption, we see how the *week* and *day* acquire more importance. Finally, we also have a high score on solar exposure: meanwhile, the information on rain is complementary, so the two features could also be combined by admitting to considering negative values for rain. Furthermore, considering that what we are analyzing is the price of electricity produced by households through solar panels, we can understand why solar exposure is important.
Here too, as for the Facebook Prophet model, the predicted values compared with the real ones and visible in the Figure 7.4, we do not have very high precision. However, although they have a very large optimization margin, we consider that both models were used without investing too much time in tuning but to be used as a baseline for the other solutions presented.

Figure 7.3: XGBoost features importance.

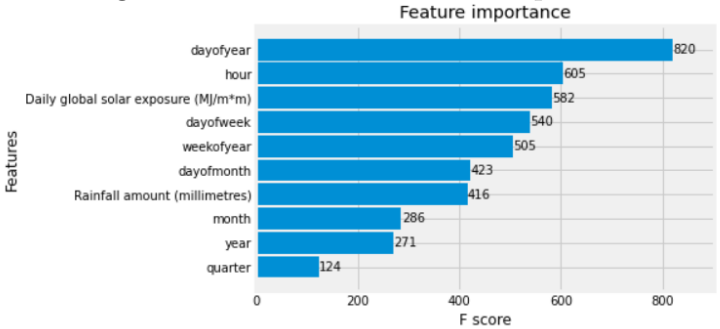
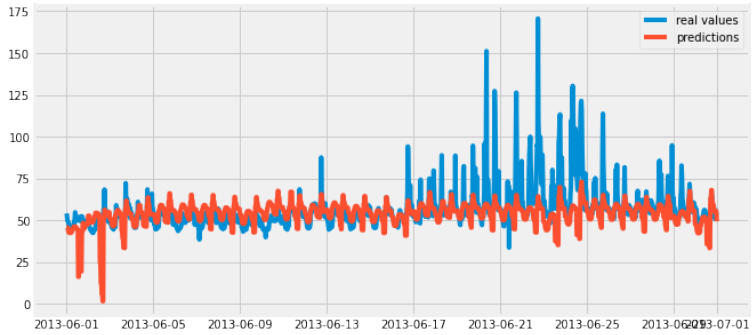


Figure 7.4: Real values compared with XGBoost predictions.



MSE	MAE	MAPE
143.939	7.05	11.272

Table 7.2: XGBoost Results.

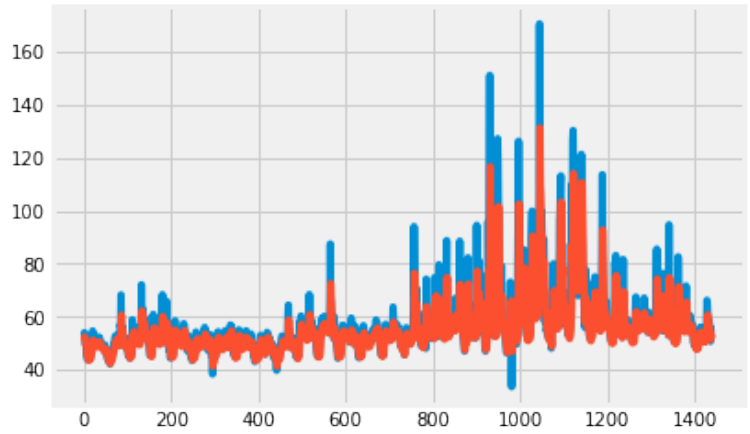
7.3 Vanilla LSTM

The results obtained by the vanilla LSTM implementation are shown in Table 7.3. The implementation of the vanilla version of LSTM needs a necessary premise. The results shown were generated using only the price value data. The rationale behind this choice is that entering external data as input allowed the loss to have a decreasing trend, but the network loses information on the data to predict. What happened was to have enormously high MSE and MAE values. Furthermore, the network training was carried out for 1000 epochs, a fairly high number but which in terms of performance did not negatively impact the data, albeit numerous (to be precise, the number of total data points is 52608), were just numbers. As we can see from the Figure 7.5, the results of the vanilla version of LSTM are appreciable, as also indicated in the table 7.3. This behavior is certainly due to the greater complexity of the neural structure.

MSE	MAE	MAPE
69.754	4.446	6.073

Table 7.3: Vanilla LSTM Results.

Figure 7.5: Real values compared with vanilla LSTM predictions.



MSE	MAE	MAPE
55.642	4.361	3.764

Table 7.4: Stacked LSTM Results.

7.4 Stacked LSTM

The last results obtained by the Stacked LSTM implementation are shown in Table 7.4. As we will see in the section 7.5, the version we implemented obtained the best results among the various implementations. Both the vanilla version of LSTM and the Stacked LSTM use the sliding window mode introduced in the section 2.1.1 and trained for 1000 epochs. The sliding window's width has been set to 4, which we have empirically found to be a good compromise between results produced and processing time. Furthermore, we set 100 as the hidden layer dimension, to which two feedforward layers are then applied to return to the unit size. Furthermore, as we see in the figure 7.6, we trained the models on a variable dimension of training data, taking, for each figure, six months of extra points and making the following month's prediction. We notice that as the number of data increases, the quality of the prediction and the exact reconstruction of the price trend increases. Using only six months as input, we have completely wrong values as the network is not yet ready to be able to recognize patterns and trends from the data.

Figure 7.6: Some grouped images

Setup	MSE	MAE	MAPE
FB Prophet	231.559	11.187	14.924
XGBoost	143.939	7.05	11.272
LSTM	69.754	4.446	6.073
Stacked LSTM	55.642	4.361	3.764

Table 7.5: Overall Results.

7.5 Solutions breakdown

From the results listed so far, we can draw some conclusions. The first is that the Deep Learning-based approach has generally achieved better results: this is a trend recognized in the literature and has been further verified in this case. Machine Learning algorithms scale badly with the number of data, that is, after a certain number of data, which depends on the algorithm and the data considered, if you add others, the performance does not change. Furthermore, using traditional machine learning techniques, many of the features must be identified in order to reduce the complexity of the data and better emphasize the patterns during training. This procedure is what is instead learned from Deep Learning algorithms. Obviously, the hardware on which the neural networks do the calculations and the training times are much greater, but as we see, the results justify the wait. Finally, we see that concerning the MSE and MAE metrics, the two solutions based on deep learning do not differ much, also considering the range of prices. However, the value of the MAPE decreases from 6,073 to 3,764, which we remember tells us the error made on average as a percentage of the predicted values. This result is an important take-home message. It confirms how external factors to the price data help the network reconstruct the trend more accurately and predict future values more accurately. All the results are listed in Table 7.5.

Chapter 8

Conclusions and future works

Working on this thesis, we had the opportunity to study some structures and topics that I had only dealt with in class and independent studies. Furthermore, being a significant area, many researchers work on these issues year after year, producing increasingly satisfactory results; in fact, although the solution presented has produced valid results, there is significant room for improvements.

In this thesis, we presented our LSTM solution for the problem stated and obtained different results from different approaches. We introduced the world of Time Series Analysis, explaining how it works and showing examples of fields where they are being used nowadays. We also described the different implementations experimented through years for solving different types of problems.

Starting from this knowledge, we tried our solution for a specific problem belonging to the Time-series Forecasting, practically experimenting on different datasets, including the dataset of the historical data on energy prices per day from 2010 to 2013, the dataset containing the historical data on climate, considering rainfall and solar exposure and finally the data concerning energy consumption that are derived from a collection of 300 households located in Australia with rooftop solar systems installed and with a gross meter that records the total amount of solar power generated every 30 minutes, so that we have the amount of energy surplus or deficit produced or needed by individual homes.
We experimented two Machine Learning solutions: Facebook prophet, that is a procedure for forecasting data, and it is based on a decomposable Time-series model with three main components that are able to model trends and periodic changes from a Time-series and XGBoost that is an optimized dis-

tributed gradient boosting library: it consists of a tree structure which aims to make decisions and discriminate between different classes. XGBoost analyzes the data and maximizes the decision making of this tree through the use of machine learning techniques.
After that, we implemented a Deep Learning approach using the vanilla version of the Long-Short Term Memory network. Then, an extended version we created, in which the external data are fed into the network in the forward phase and not in the input one.

From the results, we were able to see how the Deep Learning-based approach has produced generally better results, confirming how the greater complexity and adherence of structures such as LSTMs to data encoded in time series help this type of problem. Furthermore, although in the literature we can find similar task solutions with better results (albeit with different data and contexts), the important message we have derived is that using different features that characterize price fluctuations has a considerable impact on prediction. Therefore, in order to be able to identify with more certainty and precision other possible factors that determine these variations, the proposed solution can obtain even better results by merely changing or extending the external data.

8.1 Future Works

The work presented in this master thesis has a lot of possible improvement because some part of the problem was not analyzed thoroughly.

8.1.1 Variants of LSTM

The first possible extension is to implement the so-called bidirectional LSTM [3]. In the structure described in fact, we used a unidirectional LSTM, which preserves the past information because the only inputs received are from the past.
This is done not only by running the inputs forward but also backward in the LSTM. In this way, we preserve the information from the future, and combining the two hidden states, we are able at any point in time to maintain information from both past and future. The Bidirectional structure has applications in the Natural Language Processing field because to understand the meaning of a word, it is important to know both the word that comes before and the one that comes after.

Another possible LSTM variant that fits the problem of analyzing Time Series is the Gated Recurrent Unit (GRU), a simplified version of LSTM that does not have separated memory cells but still controls the flow of information, improving time performance. The architecture is explained in Section 2.3.

8.1.2 Recommender System

Another possible extension is to rethink how buying and selling management can be more efficient. This improvement can certainly be made by implementing a recommendation system; with the advent of the internet, we have had access to a constantly growing number of information. In addition to the need for tools that allow access and manipulation of large amounts of data, systems have also become necessary that recommend the best data to users. For this reason, recommendation systems have been created, which in recent years have offered increasingly interesting challenges and applications in everyday life (just think of the systems used for online purchases or the advice given to us in multimedia platforms).

In our case, the implementation of a recommender system can help us select precisely the correct period in which to make a purchase or sale, taking into consideration, in addition to the price data to decide between and the history of past transactions.

Bibliography

[1] Y.S. Abu-Mostafa and A.F. Atiya. "Introduction to financial forecasting, Applied Intelligence 6.3". In: 1996, pp. 205–213.

[2] Hirotugu Akaike. "Fitting autoregressive models for prediction". In: *Annals of the institute of Statistical Mathematics* 21.1 (1969), pp. 243–247.

[3] Marco Basaldella et al. "Bidirectional LSTM Recurrent Neural Network for Keyphrase Extraction". In: Jan. 2018, pp. 180–187. ISBN: 978-3-319-73164-3. DOI: 10.1007/978-3-319-73165-0_18.

[4] Michael A Benjamin, Robert A Rigby, and D Mikis Stasinopoulos. "Generalized autoregressive moving average models". In: *Journal of the American Statistical association* 98.461 (2003), pp. 214–223.

[5] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736-1936*. Oxford University Press, 1986.

[6] Javier Contreras et al. "ARIMA models to predict next-day electricity prices". In: *IEEE transactions on power systems* 18.3 (2003), pp. 1014–1020.

[7] J. H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. February 1999.

[8] Schmidhuber Jürgen Hochreiter Sepp. ""Long Short-Term Memory". Neural Computation." In: (1997-11-01). URL: https://doi:10.1162/neco.1997.9.8.1735.

[9] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". In: (2015). Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[10] G. E. Salcedo et al. "A wavelet-based time-varying autoregressive model for non-stationary and irregular time series". In: *Journal of Applied Statistics* 39.11 (2012), pp. 2313–2325. DOI: 10.1080/02664763.2012.702267. eprint: https://doi.org/10.1080/02664763.2012.702267. URL: https://doi.org/10.1080/02664763.2012.702267.

[11] "MEAN ABSOLUTE PERCENTAGE ERROR (MAPE)". In: *Encyclopedia of Production and Manufacturing Management*. Ed. by P. M. Swamidass. Boston, MA: Springer US, 2000, pp. 462–462. ISBN: 978-1-4020-0612-8. DOI: 10.1007/1-4020-0612-8_580. URL: https://doi.org/10.1007/1-4020-0612-8_580.

[12] Letham B. Taylor SJ. "Forecasting at scale." In: (2017). URL: https://doi.org/10.7287/peerj.preprints.3190v2.