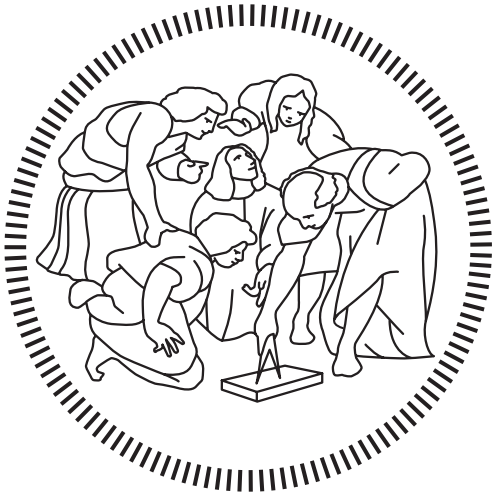
Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science – Mathematical Engineering



Mid-Term Probabilistic Load Forecasting with Recurrent Neural Networks

Supervisor
Prof. Roberto BAVIERA

Co-Supervisor
Michele AZZONE

Co-Supervisor
Prof. Paolo BRANDIMARTE

Candidate
Pietro MANZONI – 921050

Academic Year 2019 – 2020

Ringraziamenti

Desidero in primis esprimere un sentito ringraziamento al Prof. Baviera e a Michele Azzone per la grandissima disponibilità, competenza e passione con cui mi hanno accompagnato nella scrittura di questa tesi. Il loro supporto è stato fondamentale e ogni momento di confonto illuminante.

Ringrazio la mia famiglia per avermi aiutato e sostenuto in tutte le scelte che mi hanno portato ad arrivare dove sono oggi. Grazie ai miei genitori per essere sempre stati al mio fianco e per avermi incoraggiato nei momenti più difficili, e a mio fratello Nicolò perchè più di tutti mi ha sopportato in questi anni.

Un grazie speciale va poi a Luca, Danny, Gianluca, Daniele e a tutti i BC(S), compagni di mille avventure che non hanno mai fatto mancare il loro sostegno. E ringrazio ovviamente tutti gli amici del Politecnico - in particolare Matteo - con i quali, giorno dopo giorno, ho avuto il piacere di condividere questa bellissima esperienza.

Infine, vorrei ringraziare Laura, Francesco e Simone, i miei veri compagni di viaggio in questi anni. Loro e le lunghe chiacchierate sul treno mi mancheranno certamente moltissimo (i sedili scomodi e la curiosa fauna che popola il Lecco-Milano un po' meno).

La realtà è che odio gli addii e i cambiamenti, ma sono sicuro che - ovunque mi porterà il futuro - ogni volta che ripenserò agli anni dell'università lo farò con gioia, e questo lo devo a tutti voi. Grazie di cuore.

Sommario

In un mondo sempre più caratterizzato da un alto fabbisogno di energia elettrica la capacità di predire accuratamente i consumi futuri è fondamentale. Le tecniche convenzionali sono mirate alla generazione di previsioni puntuali; tuttavia negli ultimi anni si è osservato un crescente interesse per il campo delle previsioni probabilistiche. Ciò è principalmente dovuto alla loro capacità di offrire informazioni sull'incertezza, un elemento cruciale per prendere decisioni ottimali.

Lo scopo essenziale di questa tesi è l'introduzione di due modelli per la previsione probabilistica su base oraria; entrambi considerano un orizzonte temporale di un anno e fanno uso di Reti Neurali Ricorrenti. Le prestazioni di tali modelli sono valutate sulla serie storica 2009-2015 dei consumi elettrici del New England, USA. I risultati mostrano come i modelli proposti siano in grado di ottenere una maggiore accuratezza in termini di previsione puntuale rispetto ad alcuni modelli utilizzati tipicamente nel settore; in aggiunta si riscontra anche una maggiore appropriatezza delle densità di probabilità generate.

Parole chiave: Previsioni probabilistiche, medio termine, reti neurali ricorrenti

Abstract

In a world characterized by a huge need for energy, the ability to predict accurately the future electrical demand is crucial. Conventional techniques are aimed at generating point forecasts; however, in recent years, probabilistic forecasting is becoming increasingly widespread. This is mainly due to the fact that probabilistic forecasts can offer some information about uncertainty, a fundamental element to make optimal decisions.

The essential aim of this thesis is the introduction of two models for forecasting the probability distribution of the hourly load; both consider a time-horizon of one year and make use of Recurrent Neural Networks. The performances of these models are assessed by using the time-series 2009-2015 of the electrical demand of New England, USA. The results show that the proposed models are capable of achieving a greater accuracy with respect to some standard models used in the sector; in addition, even the predicted probability densities are found to be more appropriate.

Keywords: Probabilistic forecasting, mid-term, recurrent neural networks

Contents

Notation and Symbols

$\dfrac{\mathrm{d}f}{\mathrm{d}\underline{x}}$	Jacobian matrix, defined in such a way that $\left[\dfrac{\mathrm{d}f}{\mathrm{d}\underline{x}}\right]_{ij} = \dfrac{\mathrm{d}f_i}{\mathrm{d}x_j}$.
∇f	Gradient, always defined as a column vector.
$\nabla^T f$	Transpose gradient (row vector). If f is a scalar field, $\nabla^T f \equiv \dfrac{\mathrm{d}f}{\mathrm{d}\underline{x}}$.
$\| \, \|_p$	L^p norm of a vector or of a matrix (induced norm).
$\odot$	Elementwise product (Hadamard product).
$\oslash$	Elementwise division.
δ_{ij}	Kronecker delta.
$\mathbb{E}$	Expected value of a random variable.
$\varphi(x, \mu, \sigma)$	PDF of univariate Gaussian distribution with mean μ, variance σ^2 evalutated in x.
$\mathscr{L}$	In NNs, loss function associated with a single example.
ℓ	In NNs, loss associated with a single example.
J	In NNs, average loss of the training set.
$\underline{\boldsymbol{\theta}}$	In NNs, vector that collects all the trainable weights.
$\sigma(x)$	Logistic sigmoid function.
$\mathcal{A}$	In NNs, activation function.
$\mathcal{A}'$	In NNs, Jacobian matrix of an activation function.
$\mathcal{D}$	In NNs, dense affine application between two layers.
$\mathcal{L}$	In FNNs, application between two layers.
K_i	In NNs, kernel of the i-th layer.
$\underline{b}_i$	In NNs, bias of the i-th layer.

Acronyms

ACF	AutoCorrelation Function.
AI	Artificial Intelligence.
AIC	Akaike Information Criterion.
ANN	Artificial Neural Network.
API	Application Programming Interface.
APL	Average Pinball Loss.
AR	AutoRegressive.
ARMA	AutoRegressive and Moving Average.
ARX	AutoRegressive eXogenous.
BIC	Bayes Information Criterion.
BPTT	Backpropagation Through Time.
CNN	Convolutional Neural Network.
FNN	Feedforward Neural Network.
GEFCom	Global Energy Competition.
GLM	General Linear Model.
GRU	Gated Recurrent Unit.
HPC	High Performance Computing.
LHS	Left-Hand Side.
LSTM	Long Short-Term Memory.
MAPE	Mean Average Percentage Error.
MLP	MultiLayer Perceptron.
NARX	Nonlinear AutoRegressive eXogenous.
NAX	Neural Network Autoregressive eXogenous (cf. Chapter 3).

NLP	Natural Language Processing.
NN	Neural Network.
OLS	Ordinary Least Squares.
PACF	Partial AutoCorrelation Function.
PDF	Probability Density Function.
PLF	Probabilistic Load Forecasting.
ReLU	Rectified Linear Unit.
RHS	Right-Hand Side.
RMSE	Root Mean Square Error.
RNN	Recurrent Neural Network.
RTRL	Real-Time Recurrent Learning.
SGD	Stochastic Gradient Descent.
TBPTT	Truncated Backpropagation Through Time.

List of Figures

List of Tables

Introduction

The economic and technological development of a country is strictly related to the availability of electricity infrastructures and networks. Modern society requires a huge amount of electrical energy in every moment, every day: our world is interconnected, fast, smart and enlightened thanks to the presence of a complex system of production and distribution of electricity that has gradually evolved in the last decades.

As a general matter, the more a country is developed, the more electrical power is needed for industrial, business and personal use. The topic of electricity demand is contemporary, now more than ever: all the recent studies highlight how this latter is projected to increase worldwide in the next years (cf. e.g. IEA 2020). On the one hand, this is a consequence of the improvements in the standard of living and of the expansion of economies; on the other, electricity is expected to become increasingly central as clean energy transitions accelerate (cf. Vanegas Cantarero 2020).

Electricity demand forecasting plays a fundamental role both in allocating the load for everyday use and in planning the future augmentation of facilities and transmission lines; moreover being able to generate accurate predictions is extremely important in order to accomplish other relevant tasks, such as scheduling the maintenance of the power systems, minimizing the risks for the utility company and achieving the maximum utilisation of the power plants (cf. e.g. Hong 2010). All these activities concern specific time horizons: in the present thesis, we will focus on *mid-term* forecasting, which means that we will consider one-year-ahead load forecasting; why this time-frame is relevant in the practical situations will be explained in the next chapter.

In general a multitude of different models exists and many techniques have been used over the years with the aim of producing precise forecasts. The traditional methodologies are designed to generate point-forecasts; however in the most recent decade, because of the increase of market competition and of the intrinsic uncertainties associated to electricity demand, Probabilistic Load Forecasting (PLF) has become ever more important (cf. Hong & Fan 2016). The advantage of PLF is that each forecast is not thought as a single point-prediction, but rather it takes the form of a predictive probability distribution over the future values of the load.

Clearly this approach offers a more valuable information, since it provides details about the uncertainty of each prediction.

Of course this requires more sophisticated models, which have to be carefully designed and tested. In the latest years, in accordance with the technological improvements of the sector and the availability of a greater amount of data, new techniques are emerging: the most interesting ones are based on Artificial Intelligence (AI) and in particular on Artificial Neural Network (ANN), soft computing models that are finding countless applications in all the fields. In detail, interesting results have been obtained adopting Recurrent Neural Network (RNN), peculiar models which are designed for analysing sequences, and therefore time-series (cf. e.g. Hong & Fan 2016).

Having outlined the context in which this thesis is conceived, we now mention that it is partly based on the previous work of Azzone & Baviera (2021). In this paper, the authors propose a model for mid-term PLF that addresses the problem of predicting the *daily* aggregate power consumption. More in detail, they consider a Gaussian-based approach for modelling the daily electrical load and make use of a Recurrent Neural Network to predict the probability densities of this latter quantity.

The present thesis is the natural prosecution of this work and is aimed at:

- introducing a novel model for predicting the mid-term PLF on an *hourly* basis. We would like to adopt a Gaussian parametric approach as well, and to suitably use a Recurrent Neural Network in order to forecast the mean and the variance of the future hourly distributions. In designing the model, we need to find an effective method to take into account the autocorrelation of the time-series: hourly electrical load is indeed characterized by a noticeable serial dependency, as remarked by Bianchi et al. (2017).

- testing the performances of the proposed model in terms of both point accuracy of the predictions and reliability of the forecasted probability distributions. Moreover, we would like to show that the proposed model can outperform some existing linear models that are used in this field.

- analyse the state-of-the-art of PLF, focusing in particular on the model introduced in Azzone & Baviera (2021).

Our work is organized on two complementary levels: firstly the modelling one, since the final aim is to build a convincing mathematical representation of the dynamics of electrical load; secondly, the computational one, because a special interest is reserved to the practical implementation of those models. It is clear that in principle the two things should be closely related; nevertheless, in the field of Neural Networks this is not always true. Because of the existence of many

high-level libraries that allow the creation of these versatile models, their use is increasingly oriented towards immediate practical application, often neglecting their actual functioning. In many cases this is perfectly reasonable, however, in our opinion, it is always fundamental not to lose touch with the mathematical structure of the models; this holds in particular when facing new research problems. In this sense, we heed the appeal of Tien (2003): 'Most people explain networks are just black boxes. However, this lack of understanding has done more harm than good to this research area.'

Furthermore, the attention for the implementation aspects is also aimed at considering the required computational resources: Artificial Intelligence is known for being an extremely computer-intensive field. Instead, we would like to design models that can be reasonably managed without the need for too advanced technologies.

The rest of the thesis is structured as follows:

Chapter 1 provides an overview of electrical load forecasting, in order to define some important notions, to present the overall framework and to review the existing literature on PLF.

Chapter 2 is devoted to describing Artificial Neural Network. It is thought to give an insight into the structure of Feedforward and Recurrent Neural Networks, to analyse their functioning, to summarize their properties and to explain the algorithms that are typically used for training them.

Chapter 3 introduces the problem of forecasting the daily aggregate load. It is focused on the analysis of the NAX model (Azzone & Baviera 2021), which has central importance for the following chapters.

Chapter 4 illustrates the dynamics of intra-daily electrical load: this is necessary to deduce its main features and the criticalities that may arise when designing a model; moreover, two benchmark models are presented.

Chapter 5 introduces two new models based on Recurrent Neural Networks, designed to produce intra-daily probabilistic forecasts of electrical load; their predictive accuracy is then compared to the one of the previously introduced benchmark models.

Chapter 6 summarizes the work and provides conclusive comments.

Chapter 1

Overview of Load Forecasting

Electrical load forecasting is a fundamental problem in the energy field, and it is arguably one of the most interesting and attractive for researchers and practitioners. Since the inception of the electric power industry, which happened more than a hundred years ago, the need for predicting future power consumption has led to the development of a large number of techniques for the purpose. The associated literature is thus extensive and covers a multitude of methodologies and time-horizons. In this regard, the central point that has to be considered is that electricity as a product has very different characteristics compared to a material product, since electrical energy cannot be stored and has to be generated when it is demanded. The ability to forecast electrical load is therefore one of the most important aspects for the management of the power systems, but also for planning and for decision making.

1.1 Classification of Load Forecasts

In the first place, load forecasting can be classified according to the considered time-frame: up to one day (or one week) for *short-term* forecasting, up to one year for *medium-term* and more than one year for *long-term* (cf. e.g. Guerini 2016).

Short-term forecasting is for instance necessary for estimating load flows, for making decisions that can prevent overloading and for many other day to day operations. Moreover, in recent years this area is becoming more and more important as a consequence of the deregulation of the power industry and the restructuring of the energy sector, which have brought new challenges in the field of load forecasting (cf. Hong 2010).

Mid-term forecasting is instead used mainly for schedule maintenance, fuel supply and some minor infrastructure adjustments. In addition, it enables a company to understand the expected load demand over a longer horizon, which is

useful for instance for the negotiation with other companies. Moreover, for the proper functioning of a company, it is important for short-term decision level to be incorporated into long-term decision level, so as to have coordination between the two (cf. Reneses et al. 2006); in this sense, this is relevant to guarantee that mid-term operations and goals are taken into account when deciding the short-term strategies.

The last category, long-term forecasting, is purely aimed at planning; for instance planning the construction of new power stations, the expansion of the grid and the future investments. Long-term forecasting focuses on the analysis of different factors with respect to short and mid-time horizons, that for instance can be the socioeconomic situation, industrial development or population growth. Furthermore, also the level of detail of the predictions is different, since typically the required output for this kind of forecasting is the average annual consumption for the next years (cf. e.g. Kandil et al. 2002).

It is clear that there is no forecast that can satisfy all these needs, and thus different models and approaches are employed for different purposes. In general, depending on the selected time horizon, the type of variables that are considered are obviously different; short-term forecasting - for instance, few hours ahead - is usually performed utilising just the historical data of the load. The more the time horizon is near in the future, the more the recent history becomes fundamental for the prediction. However, when larger time ranges are considered, like weeks, months or years, other variables come into play and have to be properly considered in order to obtain accurate and reliable predictions (cf. e.g. Fahad & Arbab 2014).

1.2 Weather Dependent Modelling

When describing the dynamics of load on longer time horizons, *weather conditions* represent the most relevant independent variables; they can for instance include temperature (both dry-bulb and wet-bulb), relative humidity, precipitations, but also wind speed and cloud coverage. The causal correlation between weather and energy demand has been underlined by hundreds of studies (see Hong 2010 and references therein) and it has thus become a common practice to consider weather conditions as a known datum when designing models. This approach, known as *weather dependent modelling*, has been used for many years and it has gradually become the standard methodology (cf. Hong 2010); moreover it 'is being commonly accepted by the industry for its simplicity and interpretability' (Xie & Hong 2018). In general, it is reasonable to assume that the incidence of weather on the electrical load is more notable for domestic consumers, but actually, it has a relevant impact also on the industry. Besides, it has to be considered that temperature can also alter the conductivity of the transmission lines, affecting the overall

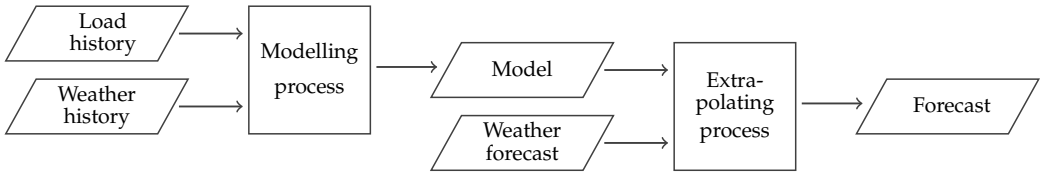


Figure 1.1. Flow diagram of weather-dependent modelling. In the initial phase, the modelling process, historical data of both load and weather are used for constructing and calibrating the model; then, during the prediction phase, weather forecasts are used to produce the final load forecast. (Figure adapted from Hong 2010)

carrying capability thereof (cf. Fahad & Arbab 2014).

The typical workflow of weather-dependent modelling is outlined in Figure 1.1. The scheme is formed by two main blocks; the first concerns the calibration of the model, in which weather and historical data are suitably combined so as to explain the realised power consumption. The second instead has to do with the forecasting process: once the model has been calibrated, it can be used to predict the energy demand over a specified period of time; however, in order to generate the forecasts, the future weather conditions must be provided as input. Depending on the time horizon, weather forecasts can be more or less accurate; in the case of few days ahead, it is possible to rely on precise predictions provided by the meteorological services. Instead, in the case of mid and long-term predictions, the model is typically fed with weather scenarios, that can be suitably generated using different techniques (cf. e.g. Xie & Hong 2018).

This is what happens in real cases; however, during the design phase of a model, it is typical to test its performances using data of the past years, so as to compare the forecasts produced by the model with the realized load. In this regard, Hyndman & Fan (2010) introduced the distinction between *ex-post* and *ex-ante* forecasting: the former expression refers to the case in which the *true* weather conditions are supplied to the model, the latter instead to the case in which the *true* weather conditions are not considered an available piece of information, and either predictions or scenarios are thus supplied to the model.

In detail, *ex-post* forecasting is very useful to evaluate the model in terms of its ability to generalise, i.e. to adapt properly to previously unseen data. In other words, it serves to understand whether the model has convincingly captured the relationship between climate and energy demand. *Ex-ante* forecasting instead represents the typical real-world application of a model, since future weather variables are unknown and they have to be somehow predicted.

In this sense, we notice that two sources of error exist: one is associated with the inaccuracy of the model (even when the *true* weather conditions are provided to the model, the prediction error is not equal to zero), while the second is due to

the inaccuracy of the provided weather forecasts. Therefore, *ex-post* forecasting is very helpful in evaluating the model performances because it separates the first kind of error from the second. For this reason, in the following chapters, we will focus on *ex-post* forecasting.

1.3 Literature Review

Among the different branches of load forecasting, mid-term PLF is characterized by rather limited literature. It has to be said that generally the largest part of the overall literature is focused on the techniques for producing point predictions: this is arguably due to the fact that decision making in the utility industry is mainly based on the evaluation of the expected value of the power consumption. Moreover, this fact holds not only in the case of load forecasting but also for the other important research area of *electricity price forecasting* (cf. Weron 2014).

However, it is not really clear why the literature of PLF is more limited - for instance - than the ones of probabilistic wind forecasting and of probabilistic forecasting in general. What instead is known is that, as a consequence of the previously mentioned deregulation of the power industry, during the early 2000s the efforts of researchers and practitioners were primarily devoted to short-term predictions, because of the increased competition in electricity markets; conversely, limitations in infrastructure investment reduced the need for mid and long-term forecasting (Hong & Fan 2016).

Just in the latest years, because of the fact that existing infrastructures have shown their age limits and because of the increased employment of new technologies, like *smart grids*, the research in these latter fields has been encouraged, and the interest for a higher level of granularity of the predictions has increased. In addition, the recent introduction of the GEFComs, the Global Energy Competitions, has ulteriorly stimulated the development of new techniques and methodologies; in particular, the last one, that took place in 2017, was expressly aimed at enhancing the knowledge about mid-term probabilistic forecasting (cf. Hyndman 2020).

Since PLF is an emerging branch of the more general field of load forecasting (and therefore is not totally independent of *point* load forecasting) we firstly present a concise review of the techniques used in this latter subject, focusing on the main methodologies that are important for the discussion. Subsequently, we analyse the literature of PLF, providing suitable insights about the state-of-the-art, and specifying the framework in which we will work in the following chapters.

1.3.1 Point Forecast

Load forecasting techniques are usually classified into two categories: *statistical techniques* and *Artificial Intelligence techniques* (cf. e.g. Hong & Fan 2016). The former group is mainly composed of standard methodologies, such as multivariate linear regression, exponential smoothing, AutoRegressive and Moving Average (ARMA) models and parametric additive models. The greatest part of the literature is based on multivariate linear regression, which allows to model the dependency of load from other exogenous factors, like calendar and weather variables. Many different linear regression models have been introduced and are currently in use both for short and long-term point predictions (see e.g. Hong 2010 and Guerini 2016). In this regard, it is important to recall that these models are *linear* in the sense that the dependency between the variables is expressed by a linear equation, but the variables themselves can be suitably transformed (for instance, the logarithm of the load is often used as dependent variable, as noted by Hong 2010, Chapter 3): this makes this kind of models extremely versatile and able to describe even some forms of non-linearity.

Artificial Intelligence techniques are instead more diverse and include for instance Artificial Neural Networks, fuzzy regression models, support vector machines and gradient boosting machines (cf. e.g. Hong & Fan 2016): as mentioned, in the present thesis we will be particularly interested in the first kind of models (which will be presented and described in detail in Chapter 2). Artificial Neural Networks are indeed reported to be extensively used in the field since the 1990s; the advantage of these models is the fact that in principle the relationship between variables does not have to be explicitly modelled, as happens for instance in linear regression: by learning patterns from the historical data the network can construct a proper mapping between the input and output variables. Moreover, their remarkable ability to capture nonlinear dependencies have led to a widespread diffusion of these models, which are currently used for forecasting on different time-horizons (cf. e.g. Feilat et al. 2017 and Rodrigues et al. 2014). In particular, noticeable results in mid and long-term modelling have been obtained by using a particular type of Neural Networks, called Long Short-Term Memory (LSTM), which is expressly designed for the analysis of sequences (cf. Agrawal et al. 2018).

1.3.2 Probabilistic Load Forecasting

Despite being quite scarce, the literature of PLF contains a variety of techniques. One of the first works that outlined the importance of density forecasting is the one of Hyndman & Fan (2010), in which the authors propose a two-staged semi-parametric additive model for predicting the long-term peak demand in South Australia. Their approach makes use of simulated scenarios for the temperature to

generate the required probability distributions, and they show that such a model can be used for predictions up to ten years. Incidentally, this methodology is reported to have been employed in practice by the Australian Energy Market Operator (Hong & Fan 2016).

The recent edition of GEFCom 2017 led instead to the introduction of other methodologies: for instance, significant results have been obtained using *quantile regression* for mid-term PLF (cf. Ziel 2019 and references therein). This technique - which is an extension of standard linear regression - is aimed at characterizing the entire conditional distribution of the dependent variable, providing a more comprehensive statistical modelling with respect to the usual (mean) regression.

Other relevant machine learning approaches are instead associated with tree-based techniques, namely *gradient boosting machines* and *quantile random forests* (cf. Smyl & Hua 2019 and Roach 2019). Both methods were exploited by some of the winning teams of the Global Energy Competition 2017, proving the remarkable accuracy that these predictive models can achieve in mid-term PLF. The main feature of all these methodologies (including quantile regression) is that they do not require assumptions on the probability distribution of the load. In these cases, the models produce as output an empirical density, which is characterized by means of a suitable number of forecasted quantiles.

This fact can of course bring some benefits in terms of flexibility; however, the approach that we want to adopt in this thesis is based on the use of a parametric distribution, namely a Gaussian density. As mentioned in the Introduction, we will consider as main reference the model proposed by Azzone & Baviera (2021), called NAX, which makes use of a Recurrent Neural Network (cf. Section 2.2) to generate mid-term forecasts; in particular, they focus on the daily aggregate consumption on a time horizon of one year. In this case, each output of the Neural Network consists of the mean and the standard deviation of the forecasted distribution. A similar approach can also be found in Vossen et al. (2018), but in this other case, the aim is to forecast just the one-hour-ahead electrical load.

Another interesting technique that makes use of Gaussian distribution is the Gaussian process regression, a Bayesian statistical methodology based on the multivariate normal distribution. The effectiveness of this approach in mid-term PLF is for instance shown in Baviera & Messuti (2020); however Gaussian process regression relies on a strong assumption of multivariate normality and it has been outperformed by the previously mentioned NAX in terms of accuracy of the predictions (cf. Azzone & Baviera 2021).

For this reason, we would like to extend the approach used in Azzone & Baviera (2021), so as to generate one-year-ahead probabilistic forecasts, but with a higher level of granularity: we aim indeed at making forecasts on an hourly basis. To the best of our knowledge, there are no previous studies that make use of a parametric

distribution for predicting the hourly dynamics of electrical load on the mid-term horizon.

1.4 Evaluation

We conclude this introductory section by illustrating the most typical procedures that are used to evaluate the goodness of a probabilistic forecast.

In general, PLF aims at providing information on the uncertainty of a future value. As mentioned before, the energy industry is mostly interested in the point predictions, which are more immediate to be interpreted and to be used to make decisions. However, expected values can be easily computed starting from the predicted probability densities, entailing that point predictions can always be deduced from probabilistic forecasts. To evaluate the accuracy of point predictions, the most commonly used statistical measures are RMSE (Root Mean Square Error) and MAPE (Mean Average Percentage Error), as mentioned by Hong & Fan (2016). Given a set of forecasted values $\{\hat{y}_t\}_{t=1}^n$ and the corresponding actual values $\{y_t\}_{t=1}^n$, we define the two quantities as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|$$

Instead, evaluating a probabilistic distribution is a more complex task. A forecasted density is usually measured in terms of *sharpness* and *reliability*. The former notion refers to 'how tightly the predicted distribution covers the actual one' (Hong & Fan 2016), the latter is associated with the frequency of *exceptions*, i.e. how often the actual values fall outside the predicted confidence intervals.

Common metrics for evaluating sharpness are the *pinball loss* and the *Winkler score*. Given a quantile $q \in [0, 1]$ and the pinball loss for the element t is given by

$$P_t(q) = \begin{cases} (1-q)(\hat{y}_t - y_t) & \text{if} \quad \hat{y}_t \geq y_t \\ q(y_t - \hat{y}_t) & \text{if} \quad \hat{y}_t < y_t \end{cases}$$

and therefore the pinball loss over the sample is computed as

$$\text{Pinball}(q) = \frac{1}{n}\sum_{t=1}^{n} P_t(q)$$

Incidentally, pinball loss is the quantity that has to be minimized in quantile regression, implying that the aim of this latter technique is to generate a density that fits

as well as possible the true quantiles of the distribution. Moreover a point metric, which is called Average Pinball Loss (APL), can be computed as the average of the pinball loss on a given set of quantiles (usually percentiles are considered).

Winkler score (Winkler 1972), instead, is computed as follows: let $\hat{L}_t$ and $\hat{U}_t$ be the lower and the upper bounds of the double-sided confidence interval for the element t, with a level q, and let

$$W_t(q) = \begin{cases} \hat{U}_t - \hat{L}_t + \frac{2}{1-q}(y_t - \hat{U}_t) & \text{if} \quad y_t > \hat{U}_t \\ \hat{U}_t - \hat{L}_t & \text{if} \quad \hat{L}_t \leq y_t \leq \hat{U}_t \\ \hat{U}_t - \hat{L}_t + \frac{2}{1-q}(\hat{L}_t - y_t) & \text{if} \quad y_t < \hat{L}_t \end{cases}$$

Then the overall score of a set of forecasts is defined as

$$\text{Winkler}(q) = \frac{1}{n} \sum_{t=1}^{n} W_t(q)$$

In practice, Winkler score is composed of a fixed quantity $\hat{U}_t - \hat{L}_t$ (which rewards narrow confidence intervals) plus a penalty that is applied in case the actual value falls outside the confidence intervals.

Instead, the reliability of the predicted distributions are commonly evaluated by means of *backtesting*, which basically consists in checking the compatibility between the fraction of the observed values that fall inside the predicted confidence intervals and their theoretical confidence level. Statistical tests can also be performed in order to verify some relevant properties of the exceptions: this a very standard procedure that is followed, for instance, in financial risk management (cf. e.g. Azzone & Baviera 2021 and references therein).

Chapter 2

Artificial Neural Networks

Nowadays, Artificial Intelligence is carving out an ever-growing and privileged space in the collective imaginary. It is extremely difficult to imagine a future in which modern devices and machine learning will not play a prominent role and, actually, it is even more difficult to imagine any task that AI will not be able to accomplish in the future. Year after year, while scientists and philosophers raise questions on these topics, modern technologies based on Artificial Intelligence become more and more popular and high-performing, thanks to the contribution of practitioners and enthusiasts all around the world. Behind the scenes of many developments of this kind, we find Artificial Neural Network (ANN), or simply Neural Network (NN), powerful computational models that will be briefly described throughout the present chapter. Contents are freely adapted from two of the main textbooks that cover this subject, namely Goodfellow et al. (2016) and Aggarwal (2018), and supplemented with additional references. For a more exhaustive treatment of the topics, we refer the interested reader to the original books.

2.1 Feedforward Neural Networks

2.1.1 Introduction

Since the brain is *par excellence* the place where learning and cognitive processes take place, why not imitate its functioning?

The idea behind **Deep Learning**, the branch of machine learning we are about to introduce, is both simple and brilliant. The human brain has always been one of the most mysterious and complex subjects for the scientific community, which still has not been able to comprehend and explain many of its dynamics. However, the cerebral structure, intended as the billions of neurons and the physical architecture of interconnections that allow the flow of information among them, is a well-known

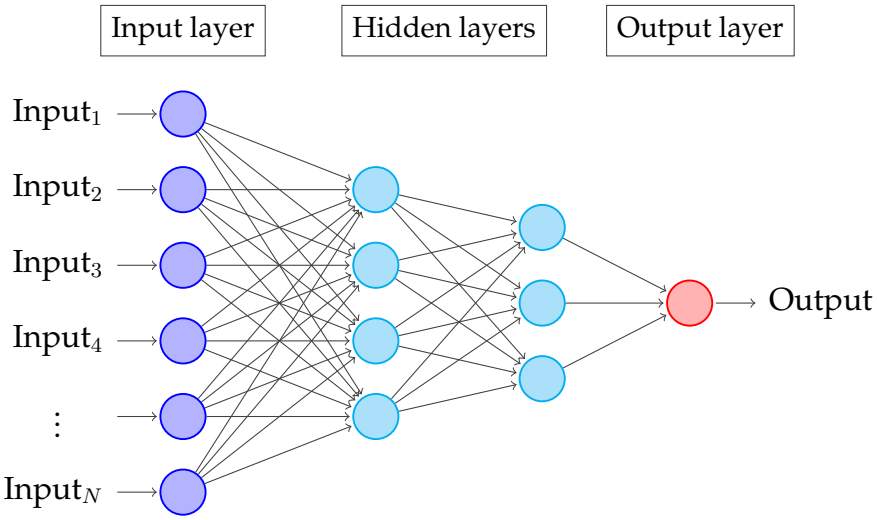


Figure 2.1. Scheme of MultiLayer Perceptron; in this case two hidden layers are considered. As denoted by the arrows, the flow of information goes from the input layer to the output layer.

fact. Artificial Neural Networks take inspiration from this structure since they represent an attempt to reproduce - in a more or less simplified way - the behaviour of the brain and of its cognitive dynamics.

In Figure 2.1 is reported the scheme of a paradigmatic type of network, which is known as MultiLayer Perceptron (MLP) (cf. Goodfellow et al. 2016, Chapter 6). This model exhibits the simplest architecture for a Neural Network and is formed by many *layers*; in turn, each layer is composed of a certain number of basic units, which are in fact called *neurons*. In particular, an MLP has an input layer, which contains all the exogenous variables that are provided to the model, an output layer, that can contain one or more neurons (depending on the problem that the model is expected to solve), and an adjustable number of *hidden layers* in between.

Neurons are tasked with elaborating and sharing information. As shown in the image, each neuron of a layer is connected to all the neurons of the following layer, generating a complex network of interactions: if we consider a neuron i in the n-th layer and a neuron j in the $(n+1)$-th layer, a weight w_{ij} is attributed to the link (i, j). Simply, this value indicates how much the information brought by the neuron i influences the neuron j. Altogether, these weights characterize the interactions between two consecutive layers and in short the calibration of a NN, called *training*, is aimed at determining the optimal value of all these parameters.

Networks designed to accomplish sophisticated tasks may require a large number of layers, leading to the creation of very deep structures: the name *Deep Learning* was originally thought to resemble this feature. On the contrary, networks that are formed by few hidden layers (typically just one) are also denoted as *shallow* Neural Networks.

Moreover, since the flow of signals is one-way, going from the input to the output, these networks are typically referred to as Feedforward Neural Networks (FNNs). The reader may imagine that other types of interconnections between layers or neurons are possible: in these cases, other architectures are obtained, like for instance the so-called Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs). We will be particularly interested in the former, which are discussed in Section 2.2; instead, the latter (which incidentally should be included in the category of FNN) will not be considered in this work (cf. Goodfellow et al. 2016, Chapter 9).

2.1.2 A quick dive into neuronal models

To understand the mechanisms of Neural Networks and the intuition behind them, it is certainly useful to go through a couple of historical moments that have been fundamental for their introduction. We have already said that the basic element of a NN is the neuron, which has been studied thoroughly by medicine and science.

In detail, as described in Wang & Raj (2017), the origins of such networks actually have to be sought in biological models. In 1943 two researchers, McCulloch and Pitts, introduced the first simplified biomathematical model for the functioning of a neuron (cf. McCulloch & Pitts 1943). They imagined a structure as the one in Figure 2.2: the dendrites, the tiny ramifications that encircle the central body of the neuron, receive the electrical stimuli from other neurons; conversely, the axon, the long protrusion, has the task of conducting the electrical impulse away from the neuron and transmitting it to the dendrites of other neurons through the synapse. What has been said so far concerns the input and output connections, but in addition the body of the neuron, the soma, and the nucleus in it are is responsible for the elaboration of the electrical information captured by the dendrites: indeed neurons belong to the class of the so-called excitable cells, which are able to modify their physicochemical behaviour when suitably stimulated. If the received stimulus is sufficiently intense, a neuron can electrically communicate with the surrounding neurons; otherwise, its axon may not propagate any impulse.

From a mathematical perspective, the McCulloch-Pitts model is defined by the following equation:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} x_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{N} x_i < \theta \end{cases}$$

where y and $\{x_i\}_i$ represents output and inputs respectively, while θ is a threshold parameter that governs the behaviour described above (see Wang & Raj 2017). It is however necessary to underline that this model is a binary model, and for this

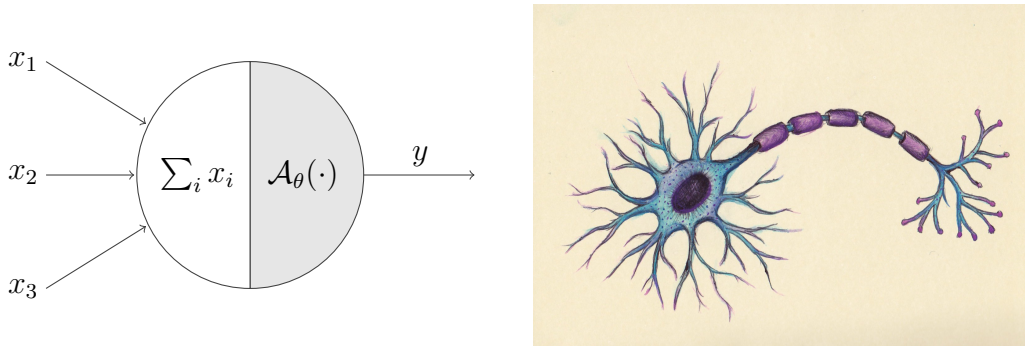


Figure 2.2. Scheme of the McCulloch-Pitts neuronal model. Dendrites gather information coming from the neighbouring neurons and transfer it to the soma: there, data are collected and summed, the threshold step-function $\mathcal{A}_\theta$ is applied and the resulting impulse is finally broadcast through the axon.

reason even the inputs $\{x_i\}_i$ shall be thought as variables that can hold either zero or one.[1]

A few years later, in 1958, the research psychologist Frank Rosenblatt introduced a modified version of the McCulloch-Pitts model (cf. Rosenblatt 1958): he abandons the purely binary framework, assuming that the nucleus can aggregate the information coming from the dendrites not just by summing them, but by means of a vector of weights $\underline{w}$:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} w_i x_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{N} w_i x_i < \theta \end{cases}$$

The Rosenblatt formulation offers greater flexibility to the model, because it leaves the opportunity of choosing (or determining empirically) the vector of weights in order to achieve a more precise and detailed mathematical description of the neuron.

2.1.3 From neurons to brains

Modern NNs are *de facto* composed of groups of neurons with a structure similar to the one prescribed by Rosenblatt model and are organized so as to form a hierarchical system of layers. Layers can easily be thought as vectors of neurons: the vector extension of the theory for the presented neuronal models is indeed straightforward. Anyway, a big difference is the fact that inputs and outputs are no longer forced to be binary variables, but are allowed to assume continuous values.

[1] In principle the model was designed to admit the presence of inhibitory variables capable of preventing the excitation of the neuron. Nevertheless they will not be considered since they are not of significant interest for the discussion.

Nevertheless, there are also two important features which, despite being presented in modern NNs in a slightly different formulation, are reminiscences of the neuronal models. Firstly, the presence of the threshold level θ is typically expressed by means of a parameter (one for each neuron) called *bias*, which is included in the weights $\underline{w}$ that have to be calibrated. It is indeed elementary to rewrite the excitability condition as

$$\sum_{i=1}^{N} w_i x_i \geq \theta \iff \sum_{i=0}^{N} w_i x_i \geq 0$$

once we define $w_0 = -\theta$ and $x_0 = 1$.

Secondly, a separate analysis of the two tasks performed by the soma, i.e. the passive reception of the incoming stimuli and the active propagation of the impulse, is very typical in contemporary architectures. The fact that electrical responses of the neurons are nonlinear functions of the (weighted) sum of the received inputs has a strong impact from a modelling perspective. Indeed we can look again at Figure 2.1: if we assume the state of each layer to be an affine function of the state of the previous one, the output layer would be an affine function of the inputs and the overall model would just become a standard linear model. Therefore the problem of determining the optimal weights for such a model would result in an underdetermined form of multivariate linear regression. Rather, the main strength of Artificial Neural Networks is the capability of capturing nonlinear relationships between variables.

In this regard, in contrast with the original neuronal models, the intensity of the output of a single neuron is typically expressed through specific functions known as *activation functions*. One of the most iconic activation function is the so-called *sigmoid*, defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

which takes values in $[0, 1]$ and thus represents a continuous extension of the step behaviour of neuronal models. Different types of activation functions are actually selected depending on the problem that the model is supposed to address: the individuation of the most suitable activation functions is indeed a part of the design process of the NN. An exhaustive list of the commonly used activation functions is provided in Appendix A.

In mathematical terms, we can therefore represent the action of every hidden layer of a MLP as a map between vector spaces of dimensions n_i and n_o

$$\mathcal{L} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_o}$$

obtained as a composition of an affine application $\mathcal{D}$ and an activation function $\mathcal{A}$

$$\mathcal{L} = \mathcal{A} \circ \mathcal{D} \tag{2.1}$$

such that

$$\mathcal{D} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_o} \qquad \mathcal{A} : \mathbb{R}^{n_o} \to \mathbb{R}^{n_o}$$
$$\mathcal{D} : \underline{x} \mapsto K\underline{x} + \underline{b}$$

The matrix K is called the *kernel* of the layer, while $\underline{b}$ is the *bias* vector. Since there is an all-to-all (directed) connection between neurons of consecutive layers, each layer of a MLP is said to be *dense* or *fully connected*.

Moreover in all the typical cases, with one significant exception, activation functions are intended as functions between n_o-dimensional spaces that actually operate *component-wise*; this is natural, since we expect the excitement of a neuron to depend only on the stimulus that it receives, and not on the stimuli received by the neighbouring neurons.

2.1.4 The universal approximation theorem

Starting from the late 1970s Neural Networks have increasingly become the subject of in-depth studies for the scientific community (in particular engineers, mathematicians, physicians and pioneers of computer science), interested in discovering potential uses within the field of Artificial Intelligence. For instance, it was exactly in those years that the principal algorithm for the training of NNs, the **backpropagation** (cf. Subsection 2.1.5), was developed and devised (cf. Werbos 1994). Anyway, the initial applications were mostly based on heuristics and merely justified by the analogy with the human cognitive system.

A crucial turning point for the theory of NN occurred when in 1989 George Cybenko proved that a Neural Network composed by a linear output layer and a hidden layer endowed with sigmoidal activation function and a suitable number of neurons can approximate with arbitrary precision any Borel-measurable function between two finite-dimensional spaces (cf. Cybenko 1989). Nevertheless, this remarkable theorem, known as the **universal approximation theorem**, has to be intended as a sort of existence result, since it cannot offer any indication of the number of neurons that are actually required for concrete applications. Some authors have tried to further investigate this topic (e.g. Barron 1994), but their findings are far from being enough general and are not applicable in practice.

Moreover, many extensions of Cybenko's theorem are present in literature, some of which - under suitable hypothesis - prove that an analogous result holds not only for sigmoids, but for any continuous activation function (cf. Pinkus 1999); however the discussion of these theorems goes beyond the scope of the thesis.

In summary, the key point of the universal approximation theorem is that NNs are in principle capable of approximating any function between Euclidean spaces. This fact is undoubtedly the main strength of this kind of models, and from a

conceptual point of view, this theorem represented the historical legitimisation for the extensive use of Neural Networks: thanks to this theoretical result, the research in this field was encouraged and intensified, in an effort to unveil their effective potential. It is nonetheless true that, as just mentioned, Cybenko's result did not bring any awareness of the way in which networks should be built in order to obtain good performances. This aspect remained - and still remains - a critical issue for Deep Learning and is usually addressed through heuristics and empirical approaches (cf. Goodfellow et al. 2016, Chapter 6.4.1).

In this regard, an important remark is that the Neural Networks are generally trained to detect and capture nonlinear dependencies between a set of provided inputs and the corresponding provided outputs, which form respectively the exogenous and endogenous variables of the so-called *training set*. In almost all the practical cases the exact way in which these variables are related, i.e. the function we are trying to approximate, is unknown: hence the critical difficulty of choosing *a priori* a suitable architecture.

2.1.5 Training a FNN: the backpropagation

So far we have described the formal structure of a FNN, the interaction between the constituent parts and we have highlighted the important approximation property; at this point we can start focusing on the effective use of these models.

There are two key phases in the creation of a Neural Network. The first has to do with the design thereof, namely the choice of the number of hidden layers, the number of neurons of each layer, the activation functions and some of the other so-called *hyperparameters* of the network. The second instead is the **training** of the network: this is the moment in which the parameters of the model - the weights - are calibrated and the *learning* actually takes place.

The training of a Neural Network somehow resembles human learning. In short, a set of pairs of input-output vectors $\{(\underline{x}_1, \underline{y}_1), \dots, (\underline{x}_N, \underline{y}_N)\}$, known as **examples** ot **training set**, is provided to the network; for each of the inputs $\underline{x}_i$ the network is asked to produce an inferred output $\underline{\hat{y}}_i$, which is then compared to the true output $\underline{y}_i$.[2] Every discrepancy between these two values is used by the network to adapt its own weights, refining its knowledge about the analysed data and, hopefully, improving its ability to generalise and produce good predictions whenever a new input is supplied.

Although more general formulations are possible, we can assume that the inputs

[2]Since the true outputs (also called *targets* or *labels*) are known and the network can access them during the training, this learning paradigm is termed *supervised learning*. In the thesis we will always consider problems that deal with this type of framework. Anyway Neural Networks are also used with remarkable results for performing *unsupervised* tasks, such as *clustering* (cf. e.g. Aljalbout et al. 2018).

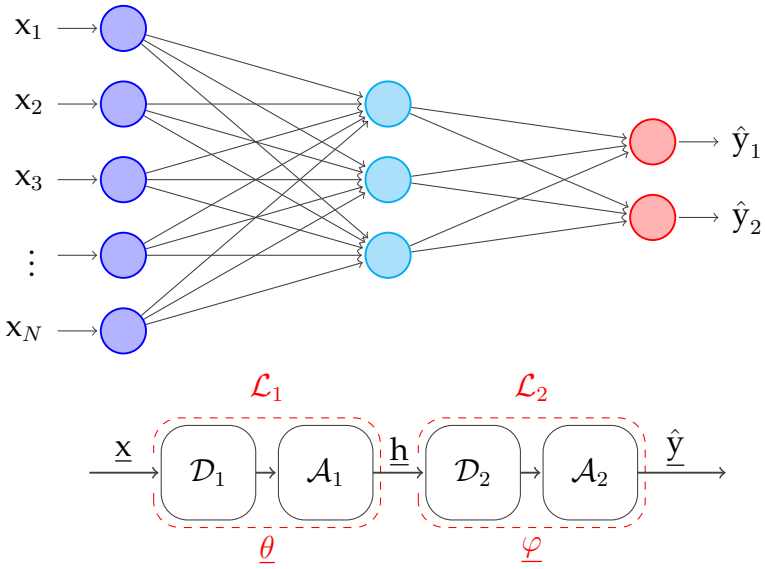


Figure 2.3. A FNN with one hidden layer and the corresponding block diagram: every map between two consecutive layers is a composition of an affine transformation (a *dense* application) and an activation function. $\underline{\theta}$ and $\underline{\varphi}$ refer respectively to the weights of the first and of the second layer.

belong to a certain Euclidean space X, while the outputs belong to a Euclidean space Y. In principle, the NN is meant to approximate the function

$$f : X \to Y$$

that generated the samples of the training set: clearly the more the produced outputs are similar to the targets, the more the model is high performing and well calibrated. In detail, the goodness of a produced output is evaluated by means of a **loss function** that suitably measures the magnitude of the error by attaching a cost to every pair $(\underline{y}_i, \hat{\underline{y}}_i)$; for many applications the loss function is a map of the form

$$\mathscr{L} : Y \times Y \to \mathbb{R} \tag{2.2}$$

which can simply be any L^p norm of the residual $\underline{\varepsilon}_i = \underline{y}_i - \hat{\underline{y}}_i$, but it may also assume different forms depending on the problem that the NN is intended to solve. For instance, we will see that the models presented in the following chapters do not use a loss function of the form (2.2).

The aim of the training procedures is to minimize the average loss over the sample, which is the average of the losses over the N pairs in the training set. To do so, gradient-based algorithms are typically used; this entails that the weights update procedure requires the computation of the gradient of the loss function with respect to the same weights.

This task is usually addressed by exploiting the characteristic structure of a FNN, which is a composition of many stacked layers. Indeed, said M the number

of hidden layers, the inferred output $\underline{\hat{y}}_i$, i.e. the one relative to i-th example, can be written as

$$\underline{\hat{y}}_i = \mathcal{L}_{M+1} \circ \mathcal{L}_M \circ \cdots \circ \mathcal{L}_1(\underline{x}_i) \tag{2.3}$$

where each $\mathcal{L}_\bullet$ is a map between two layers of the form (2.1). Instead, the corresponding loss value ℓ_i, i.e. the actual cost of the error, is given by

$$\ell_i = \mathscr{L}(\underline{y}_i, \underline{\hat{y}}_i)$$

In view of equation (2.3), differentiation with respect to weights can be performed by using the *chain rule*. For the sake of clarity, the procedure is here discussed taking as reference the shallow network in Figure 2.3; anyway the reader will realize that the following deductions apply to any more general case. For the depicted network, the loss is given by

$$\ell = \mathscr{L}\Big(\underline{y}, \mathcal{L}_2(\mathcal{L}_1(\underline{x}, \underline{\theta}), \underline{\varphi})\Big) = \mathscr{L}\Big(\underline{y}, \mathcal{L}_2(\underline{h}, \underline{\varphi})\Big)$$

where $\underline{\theta}$ and $\underline{\varphi}$ shall be thought as vectors that collect all the trainable weights in the corresponding layer. Then,

$$\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\theta}} = \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\underline{\hat{y}}} \frac{\mathrm{d}\mathcal{L}_2}{\mathrm{d}\underline{h}} \frac{\mathrm{d}\mathcal{L}_1}{\mathrm{d}\underline{\theta}} \tag{2.4}$$

$$\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\varphi}} = \frac{\mathrm{d}\mathscr{L}}{\mathrm{d}\underline{\hat{y}}} \frac{\mathrm{d}\mathcal{L}_2}{\mathrm{d}\underline{\varphi}} \tag{2.5}$$

In these computations, it is important to underline that $\underline{x}$ and $\underline{y}$ have to be treated as fixed variables, since they are externally provided and clearly the NN cannot have any influence on them.

A further investigation leads to more refined and usable formulae, once we consider the structure of the basic functions $\mathcal{L}$. Indeed each $\mathcal{L}$ can be written as

$$\mathcal{L}(\underline{x}, \underline{\theta}) = \mathcal{A}(\mathcal{D}(\underline{x}, \underline{\theta})) = \mathcal{A}\big(K\underline{x} + \underline{b}\big)$$

and the corresponding derivatives, which actually are Jacobian matrices, are given by

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\underline{x}} = \mathcal{A}' \, K$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\underline{\theta}} = \mathcal{A}' \, \frac{\mathrm{d}\mathcal{D}}{\mathrm{d}\underline{\theta}}$$

Here the symbol $\mathcal{A}'$ is to indicate the Jacobian matrix of the activation function with respect to its own inputs, which is (nearly) always a *diagonal* matrix.

To conclude, it is necessary to deduce the expression for the matrix $\mathrm{d}\mathcal{D}/\mathrm{d}\underline{\theta}$; in order to do so, we must first declare how the *vectorisation* of the weights is

performed. For our purposes, we will always assume that the kernel matrix K is flattened by considering a *row-major order* and that the bias term is subsequently appended to the obtained vector. In other words, the following type of enumeration for the θ-parameters is used

$$K\underline{x} + \underline{b} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \theta_7 \\ \theta_8 \end{bmatrix} \tag{2.6}$$

Under this assumption, the Jacobian matrix $\mathrm{d}\mathcal{D}/\mathrm{d}\underline{\theta}$, which has as many columns as the total number of weights that have to be calibrated, is

$$\frac{\mathrm{d}\mathcal{D}}{\mathrm{d}\underline{\theta}} = \begin{bmatrix} \underline{x}^T & \underline{0}^T & \cdots & \underline{0}^T & 1 & 0 & \cdots & 0 \\ \underline{0}^T & \underline{x}^T & \cdots & \underline{0}^T & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \underline{0}^T & \underline{0}^T & \cdots & \underline{x}^T & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.7}$$

with $\underline{0}$ null (column) vector of the same size as $\underline{x}$. For example, in the case of equation (2.6), the Jacobian matrix $\mathrm{d}\mathcal{D}/\mathrm{d}\underline{\theta} \in \mathbb{R}^{2\times 8}$. A relevant remark is that each column has only a non-zero element: from the implementation perspective, this fact should not be neglected.

Having analysed all the terms, we can combine them and rewrite equations (2.4) and (2.5) as

$$\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\theta}} = \nabla^T\mathscr{L}\ \mathcal{A}_2'\ K_2\ \mathcal{A}_1'\ \frac{\mathrm{d}\mathcal{D}_1}{\mathrm{d}\underline{\theta}} \tag{2.8}$$

$$\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\varphi}} = \nabla^T\mathscr{L}\ \mathcal{A}_2'\ \frac{\mathrm{d}\mathcal{D}_2}{\mathrm{d}\underline{\varphi}} \tag{2.9}$$

where $\nabla^T\mathscr{L}$ is used to indicate the gradient of the loss function with respect to the output of the network, but intended as a row vector. In summary, the problem of calculating the gradients reduces to the computation of products of Jacobian matrices. Although the entire procedure is here explained for a simple case, the presented methodology can be used for any kind of FNN with an arbitrary number of fully connected layers.

Moreover, it should be noted that in general the optimal way of performing the matrix multiplications is backward, from the loss to the input layer. Indeed the loss function is a scalar-valued function and its Jacobian matrix is actually a gradient, i.e. a vector (a *row vector*); the backward approach ensures that the long chain of matrix multiplications is treated as a series of matrix-vector products, and therefore - with very few exceptions - a low number of operations is required. In addition, this fact offers a very nice visual representation of the training procedure, since every neuron is initially implicated in a *forward* flow of the information during the processing of the inputs, and in a *backward* flow of the gradient for the optimization procedure (see Figure 2.4). This should explain why in literature the expression **backpropagation** is commonly used for referring to this differentiation algorithm.

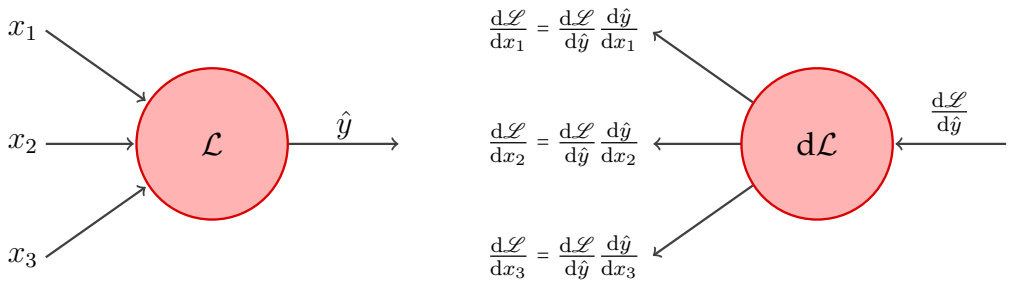


Figure 2.4. Diagram of forward and backward propagation from a neuron perspective.

2.1.6 Training algorithms

The aim of this subsection is to provide an overview of the algorithms and of the techniques used in practice to train Neural Networks. We have already mentioned that the calibration of a Neural Network involves the minimization of the function

$$J(\underline{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(\underline{\boldsymbol{\theta}})$$

where N is the number of the examples in the whole training set. Here $\underline{\boldsymbol{\theta}}$ is meant to collect **all** the trainable parameters of the model. As discussed in the previous subsection, such a problem is usually approached with gradient-based algorithms and ∇J is easily obtained thanks to the linearity of the gradient operator.

The simplest algorithm that can be used is the *gradient descent*, which recursively updates the parameters according to the equation

$$\underline{\boldsymbol{\theta}}_{t+1} = \underline{\boldsymbol{\theta}}_t - \eta \nabla J_t \tag{2.10}$$

In this context a specific terminology is adopted: the parameter η, representing the step size of the update, is called *learning rate*. Instead, the training set is said to form a *batch*, and so the term *batch gradient descent* is often used for referring to this algorithm. Lastly, an important notion is the one of *epoch*, which stands for a complete presentation of the whole training set, or better 'each repeated entry of the full set of training patterns' (Stegemann & Buenfeld 1999); in the case of batch gradient descent, the weights are updated once per epoch.

Despite its simplicity, in practice this formulation of the gradient descent is rarely used; this happens for several reasons (cf. Goodfellow et al. 2016, Chapter 8), in particular

- ▷ computing gradients for the entire training set is computationally expensive, entailing that a long time is required for each iteration;
- ▷ the number of the weights is usually very large and therefore the risk of being stuck in local minima of the function J is highly relevant.

A very common approach to address these issues is to split the training sets in *mini-batches* of size K and to approximate the exact gradient ∇J with the average gradient computed on each mini-batch:

$$\nabla J = \frac{1}{N}\sum_{i=1}^{N}\nabla\ell_i \simeq \frac{1}{K}\sum_{k=1}^{K}\nabla\ell_{i_k}$$

where $\{i_1, i_2, \ldots, i_K\}$ are the K indexes of the statistical units that form the mini-batch. The procedure, in this case, consists of analysing the training set one batch at a time: for each of them, the estimate of the exact gradient is computed and used to update the weights as in (2.10). This means that with this algorithm, the *mini-batch gradient descent*, several updates occur during each epoch. At the end of any epoch, the training set can even be shuffled; this implies that new mini-batches are formed and, in some cases, this can bring benefits in terms of convergence speed.

A very special case of the mini-batch gradient descent is called Stochastic Gradient Descent (SGD) and is obtained by selecting $K = 1$. It is clear that in principle the more K is large, the more the approximated gradient (i.e. the update direction) is similar to ∇J (i.e. the correct descent direction): therefore the SGD is characterized by a very large variability and, despite being still fairly used in practice, it is seldom the best solver that one can choose (cf. Choi et al. 2020).

In this regard, it is necessary to specify that for this kind of complex optimization problems an update in the wrong direction is not always a completely negative fact, since for instance it can help to escape from sub-optimal local minima. On the other hand, it should be said that the most effective algorithms are typically those that progressively adjust their update direction, rather than changing it completely as SGD does. Progressive adjustments are usually implemented by keeping a moving average of the previous gradients, so that the information brought by a new gradient modifies only partially the descent direction adopted for the previous updates; this approach is used in some algorithms such as *Momentum optimizer* (cf. Qian 1999) and *Nesterov Accelerated Gradient* (cf. Nesterov 1983).

Nowadays, one of the most popular and versatile optimizers is *Adam* (cf. Kingma & Ba 2014) that was originally conceived as an extension of another solver called *RMSProp* (Geoffrey Hinton, unpublished). Its name stands for *adaptive moment estimation*, and the idea behind it is indeed to keep track of both the mean and the uncentered variance of the previous gradients through moving averages. In detail, the core of the update strategy of Adam is specified by the following set of equations:

$$\begin{aligned}
&\underline{m}_t = \beta_1\underline{m}_{t-1} + (1-\beta_1)\nabla J \\
&\underline{v}_t = \beta_2\underline{v}_{t-1} + (1-\beta_2)\nabla J \odot \nabla J \\
&[\ldots] \\
&\underline{\boldsymbol{\theta}}_t = \underline{\boldsymbol{\theta}}_{t-1} - \eta\underline{m}_t \oslash \sqrt{\underline{v}_t + \varepsilon}
\end{aligned}$$

where $\odot$ and $\oslash$ are used to indicate the elementwise multiplication and division, respectively. Typical values of the hyperparameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (meaning that the impact of every innovation is very limited), while ε is just a small constant introduced for numerical stability. For the sake of completeness, RMSProp is instead obtained by setting $\beta_1 = 0$.

The reason why it is worth analysing Adam-like solvers (Adagrad, RMSProp, AdaMax, AdaDelta, Nadam, AMSGrad) is that they take advantage of what is called an *adaptive learning rate*, which means that the step size used in the optimization procedure is not kept fixed, but it is scaled by the inverse square root of the estimate of the uncentered variance $\underline{v}_t$ (cf. Goodfellow et al. 2016, Chapter 8.5). Moreover, this holds component-wise, entailing that every weight is actually updated accordingly to a *customized* learning rate, which is a function of the history of the gradients. In particular, the larger a partial derivative has been in recent history, the more cautious is the solver in proceeding along that direction; on the other hand for some solvers, this effect is counterbalanced by a Momentum-like running average of the gradients at the numerator.

A final remark is that of course all these algorithms are designed for *mini-batch learning*. The size of the mini-batches (usually called just *batch size*), as well as the learning rate η, is one of the fundamental hyperparameters that have to be accurately tuned in order to achieve good performances in optimization.

2.1.7 Underfitting, overfitting and regularization

The algorithms presented in the previous section are aimed at finding a reasonable local minimum of the loss function: when this happens, the training can be considered complete, since the model can reproduce as closely as possible the provided data. Unless few trivial cases, the minimisation of J is a complex problem and we cannot expect the provided solution to be the global minimum, because of the nonlinear structure of the loss function; however, these algorithms are designed so as to increase as much as possible the probability of reaching a *good* local minimum. Moreover, it should be underlined that all of them are iterative methods, which means that some termination criterion should be set: the most naive choice is that the algorithm is stopped after a certain number of *epochs*.

Once the network has been trained, it can be used for predicting new outputs given unseen inputs, which is the principal purpose of these kinds of models; if the network has been well designed and trained, it should have 'learnt' the crucial features from the training set and so it should be able to *generalize* and to produce reasonable predictions.

In some sense, the training procedure explained so far is not different from an optimization problem; what mainly separates machine learning from mere

optimization is that the *generalization error*, i.e. the discrepancy between the inferred value in presence of an unseen input and its real value, is desired to be low as well. In practice, the two factors that determine the performance of a machine learning algorithm are its ability to:

- ▷ make the training error (i.e. the loss) small
- ▷ make the gap between training error and generalization error small as well

The inability to fulfil these objectives is called respectively *underfitting* and *overfitting* (cf. Aggarwal 2018, Chapter 1.4 and Goodfellow et al. 2016, Chapter 5.2). Underfitting happens when the model is not complex enough to suitably capture the relevant features of the training set; overfitting instead occurs when the model learns too many details of the training set (including the existing noise) and is thus not able to generalise well. Figure 2.5 shows with a simple example the principle of these two issues.

In practice, the Neural Networks are very unlikely to suffer from underfitting, in the sense that - according to the universal approximation theorem - they can approximate any function, as long as the number of neurons or layer is suitably large; therefore, such a problem may arise either when the network is too small or the number of samples in the training set (or their quality) is not sufficient to allow a proper learning.

For what concerns overfitting, the issue is more delicate. In general, the strategies that are employed to avoid overfitting are called *regularization* techniques: for the sake of brevity, we mention two of them that are arguably the most used ones. The first one consists of adding an extra penalty to the loss function, that depends on the norm of the vector of parameters $\underline{\boldsymbol{\theta}}$ by means of a suitable positive

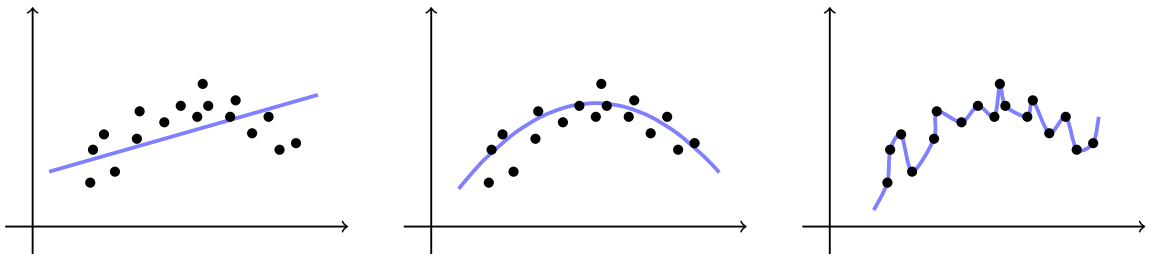


Figure 2.5. Example of underfitting and overfitting on polynomials. Underfitting (*left*) is characterised by the fact that the model is too simple to properly describe the data. The overfitting case instead (*right*) guarantees optimal flexibility and an incredible ability to minimize the training loss; but on the other hand, it does not extract the relevant features of the data. The best fit (*centre*) is obtained when the model has the right degree of complexity.

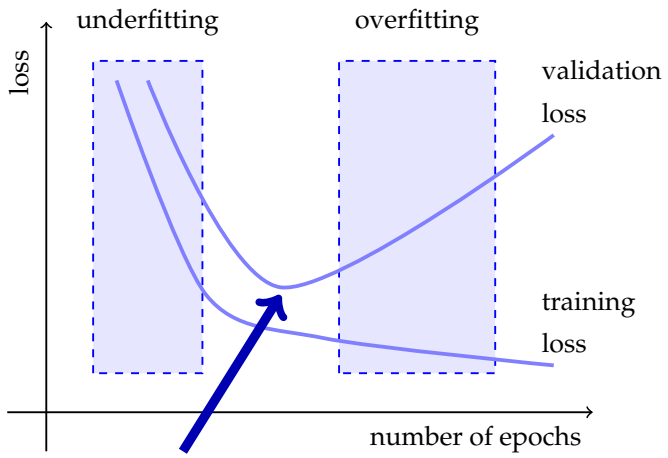


Figure 2.6. Example of use of Early stopping: when - after a certain number of epochs - overfitting starts to take place and the validation loss starts to increase, the training procedure is stopped.

function $\Omega(\cdot)$. Hence, the loss function is corrected as

$$\tilde{J}(\underline{\boldsymbol{\theta}}) = J(\underline{\boldsymbol{\theta}}) + \lambda\, \Omega(\underline{\boldsymbol{\theta}})$$

where $\lambda \geq 0$ is called *regularization coefficient*. The most important cases are the so-called L^1 and L^2 regularization, that respectively consider

$$\Omega(\underline{\boldsymbol{\theta}}) = \|\underline{\boldsymbol{\theta}}\|_1 \qquad\qquad \Omega(\underline{\boldsymbol{\theta}}) = \frac{1}{2}\|\underline{\boldsymbol{\theta}}\|_2^2$$

The second strategy is instead called *Early Stopping* and is thought to contrast situations like the one in Figure 2.6, which are fairly common in Deep Learning. *Early Stopping* requires the available data to be randomly split into a *training set* and a *validation set*. During the optimization procedure, the former is used to train the model in a regular way, while the latter (which represents a small portion of the total data) is used to evaluate in real-time the generalization error - and clearly, the network is never allowed to learn from the validation set. The training is then conducted as usual until the *validation error* starts to increase as an effect of overfitting.

However, although the original meaning of *Early Stopping* is the one just outlined, this expression is also used in a broad sense to indicate the interruption of the training when a monitored metric has stopped improving. In the case in which no validation set is created, the optimization procedure can be prematurely stopped - for instance - when the loss or other statistics of the training set are no longer significantly decreasing (cf. e.g. Mahsereci et al. 2017).

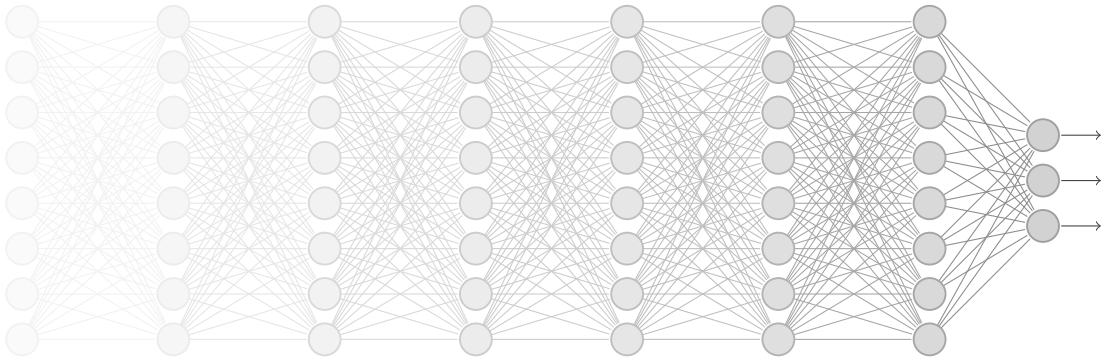


Figure 2.7. Graphical representation of the vanishing gradient in a deep feedforward network. Going backwards through the network, the magnitude of the backpropagated gradient diminishes progressively; therefore layers which are very distant from the output layer are likely to suffer from vanishing gradient problem.

2.1.8 The vanishing gradient problem

We conclude this section dedicated to the FNNs by describing a common problem that may affect Neural Networks and is called the *vanishing gradient problem*. This issue is typical of very deep networks and occurs when the partial derivatives of the loss with respect to the weights of the very first hidden layers are of several orders of magnitude smaller than the other partial derivatives. In other words, when the NN is very articulated and the flow of information must pass through many layers, a small perturbation of any weight in the initial layers may have no relevant impact on the loss (cf. Aggarwal 2018, Chapter 1.4).

The critical difficulty associated with this issue is that gradient-based optimization procedures are likely to fail since the weights under consideration are almost never modified during the training. This means that the learning is not carried out in an appropriate manner and this results in an underperforming network.

From the point of view of the backpropagation, the vanishing gradient is caused by the many matrix multiplications that are involved in the case of a deep network. We can, for instance, consider a NN with M hidden layers. The gradient of the loss for the first hidden layer (cf. equation (2.8)) is given by

$$\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\theta}} = \nabla^T \mathscr{L}\, \mathcal{A}'_{M+1}\, K_{M+1} \cdots \mathcal{A}'_2\, K_2\, \mathcal{A}'_1\, \frac{\mathrm{d}\mathcal{D}_1}{\mathrm{d}\underline{\theta}}$$

and thus for any submultiplicative norm we get

$$\left\| \left(\frac{\mathrm{d}\ell}{\mathrm{d}\underline{\theta}} \right)^T \right\| \leq \left\| \left(\frac{\mathrm{d}\mathcal{D}_1}{\mathrm{d}\underline{\theta}} \right)^T \right\| \left\| (\mathcal{A}'_1)^T \right\| \left\| K_2^T \right\| \left\| (\mathcal{A}'_2)^T \right\| \cdots \left\| K_{M+1}^T \right\| \left\| (\mathcal{A}'_{M+1})^T \right\| \|\nabla \mathscr{L}\| \tag{2.11}$$

For instance we can consider the infinity norm $\|\cdot\|_\infty$, i.e. the maximum absolute row sum of each matrix. For what concerns the most part of the activation functions - as shown in Appendix A - their infinity norm is ≤ 1 (and almost always strict

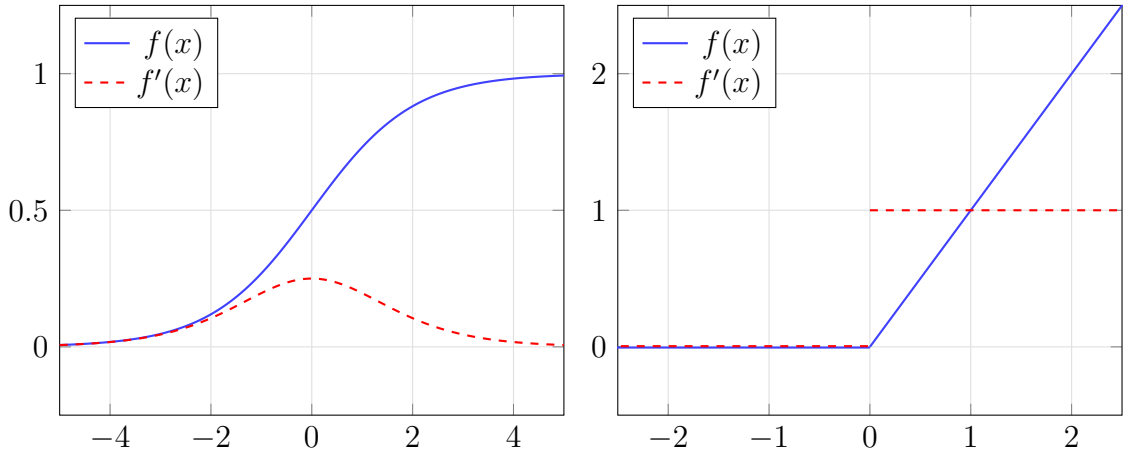


Figure 2.8. Plot of Sigmoid (*left*) and of ReLU (*right*) with their first derivatives. Unlike Sigmoid, ReLU activation function is designed to avoid saturation when the input argument becomes large and positive.

inequality holds); for $(\mathrm{d}\mathcal{D}_1/\mathrm{d}\underline{\theta})^T$, the same result is true whenever the training set is for instance **min-max** normalized[3]. Lastly, the weights that compose the kernel matrices are usually small in magnitude (in particular when regularization is applied) and for many applications kernels are rectangular matrices with more columns (i.e. inputs) than rows (i.e. outputs); thus the infinity norm of their transpose matrices is nothing but the maximum among the sums of just a few terms.

The latter is not a very formal argument, hence in practice an *a priori* upper bound cannot be found. Anyway, it should be enough to convince that in (2.11) many terms on the RHS are less than 1 and they can therefore force the LHS to be close to zero. In this regard, it is important to mention that vanishing gradient problem arises in particular with activation functions such as the sigmoid. Indeed, because of its shape, this function is very likely to get *saturated*, since its derivative is relevantly different from zero only in a relatively small neighbour of the origin, as shown in Figure 2.8. Even the presence of few saturated neurons is enough to prevent the proper functioning of the backpropagation (cf. Rakitianskaia & Engelbrecht 2015).

The vanishing gradient problem has been largely studied in literature, and usually the most effective solutions to tackle this issue are:

[3] *min-max* normalization is a feature scaling technique which is very popular in Deep Learning and consists in transforming each variable z into

$$z' = \frac{z - \min z}{\max z - \min z}$$

In this way, each transformed variable z' is forced to take values in the range [0,1] and this not only is useful in comparing the variability of the several features independently of their magnitude, but is also reported to improve the convergence of the optimization algorithms (Nawi et al. 2013).

- ▷ an accurate choice of the activation functions, because in deep networks the risk of saturating neurons becomes very relevant. Piecewise linear functions such as the *ReLU* and its variants have shown to be an effective remedy.
- ▷ a proper random initialization of weights, as suggested in Glorot & Bengio (2010) or Yam & Chow (2000).
- ▷ the selection of different training algorithms. In this regard, an interesting option is the algorithm *Rprop* (cf. Riedmiller & Braun 1993), which actually is still gradient-based, but the descent direction is chosen only according to the *sign* of the gradient; another suggestion is to training each layer separately and then to perform an eventual fine-tuning, as proposed for example in Schmidhuber (1992b).
- ▷ changing the topology of the network. For instance *Residual Networks*, also called *ResNets*, (He et al. 2016) and *Highway Networks* (Srivastava et al. 2015) are architectures that allow connections between non consecutive layers, introducing shortcuts which reveal to be useful during the backpropagation.

On the other hand, it is rarer (but not uncommon) to have a network that exhibits the opposite behaviour, i.e. an *exploding gradient*; such a problem may not only lead to suboptimal learning, but may also cause instability of the training algorithm. This issue is usually addressed with *gradient clipping* (cf. Pascanu et al. 2013): for a fixed threshold α, the gradient is adapted according to the relationship

$$\nabla J = \begin{cases} \dfrac{\alpha}{\|\nabla J\|}\nabla J & \text{if } \|\nabla J\| > \alpha \\ \nabla J & \text{if } \|\nabla J\| \leq \alpha \end{cases}$$

so that, if required, its direction is preserved and its magnitude is reduced.

Nevertheless, it is clear that both problems are actually pathological behaviours that require a careful analysis of the network and, if necessary, a redesign thereof, especially when a large number of layers is present.

2.2 Recurrent Neural Networks

2.2.1 Introduction

Feedforward Neural Networks have shown to be capable of achieving outstanding performances in a multitude of practical cases. Anyway there are many situations in which data are characterized by sequential relationships, e.g. in image generation, speech recognition, named-entity recognition, language modelling,

machine translation, and time-series analysis (cf. van den Oord et al. 2016 and Sutskever 2013, Chapter 1).

In such cases, each statistical unit $\underline{x}$ is typically formed by a sequence of basic subunits $\{\underline{x}^{(t)}\}_t$: these are for instance *pixels* for images, *words* for sentences and *observations* for time-series. Every subunit represents an essential part of the information, obviously; anyway we understand that, if we consider each subunit separately, we are neglecting the important knowledge coming from the *context*. Trivially, the meaning of a word might vary depending on the sentence to which it belongs, and a pixel simply defines a colour, which taken in isolation is not sufficient to recognize the subject of an image.

A Feedforward Neural Network is not able to capture this kind of dependency, since it treats any subunit as an independent datum; the need for a proper way of modelling sequences led to the introduction of other types of Neural Networks, which are characterized by specific topologies.

Recurrent Neural Networks (Rumelhart et al. 1986) are one of the solutions that were found to be effective for this purpose. They are networks that allow *feedback* connections, as shown in Figure 2.9; this basically means that - while elaborating the incoming information - the hidden layer can take advantage of the knowledge of its previous state $\underline{h}^{(t-1)}$. In this sense, a RNN treats the feedback as one of the many stimuli that usually travel among the neurons; thus it has to learn not only how to use properly every exogenous datum $\underline{x}^{(t)}$, but also how to fully exploit these new connections. As said, the huge advantage of such an architecture is that networks are able to understand sequential relationships among the subunits. Indeed at a given time, the state of the hidden layer is a function of the last hidden state, which is in turn a function of second-last, and so forth.

As networks become deeper, not all the hidden layers are required to have recurrent links; for instance, some of them can be just densely connected like the ones of FNNs. Of course the presence of many recurrent layers generates a strong web of interactions that may turn out to be useful in some situations. Furthermore, in other circumstances different types of recurrent connection may be required: for instance, instead of the hidden state $\underline{h}^{(t-1)}$, the output $\underline{\hat{y}}^{(t-1)}$ can be provided to the hidden layer. An example of this structure is the one of NARX neural networks, presented in Subsection 2.2.5. Although this section focuses on the general theory of RNNs, we will be particularly interested in this latter kind of models.

2.2.2 The structure of the RNN

As mentioned, RNNs are basically FNNs that, in addition, allow the presence of feedback links. With regard to the RNN in Figure 2.9, the usual convention is to consider a scheme of all-to-all *dense* connections between the returned hidden state

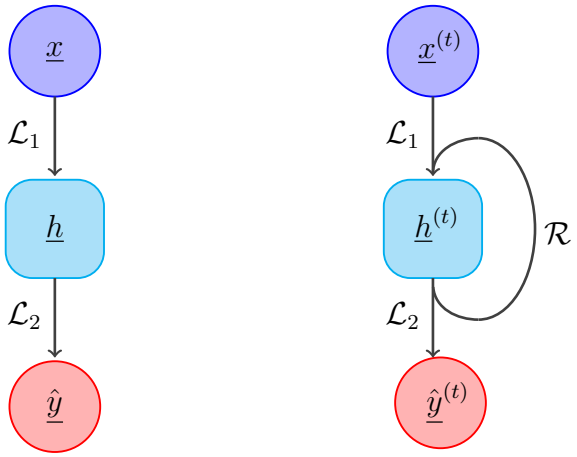


Figure 2.9. Scheme of a FNN (*left*) and of a RNN (*right*): these simple examples have only one hidden layer (the structure of the FNN is actually equivalent to the one of Figure 2.3). Every time the RNN processes a statistical unit, the output of its hidden layer $\underline{h}$ is stored and used as input for the next iteration.

$\underline{h}^{(t-1)}$ and the neurons of the hidden layer: in other words, the depicted Neural Network can be equivalently represented as (see Goodfellow et al. 2016, p. 374)

$$\hat{\underline{y}}^{(t)} = \mathcal{L}_2\Big(\underbrace{\mathcal{A}_1\big(K_1\,\underline{x}^{(t)} + R_1\,\underline{h}^{(t-1)} + \underline{b}_1\big)}_{\underline{h}^{(t)}} \Big) \tag{2.12}$$

where R_1 is the matrix that collects the trainable weights associated to the feedback links, while as before $\mathcal{L}_2 = \mathcal{A}_2 \circ \mathcal{D}_2$. Although the picture may suggest some kind of linear superposition of $\mathcal{L}_1$ and the recurrent function $\mathcal{R}$, with *abuse of notation* this is used to indicate a relationship of the form (2.12): an affine transformation of all the inputs (including the recurrent ones) followed by a passage through the activation function.

Also, it is possible to reformulate the equation for the hidden state $\underline{h}^{(t)}$ to obtain a FNN-like expression

$$K_1\,\underline{x}^{(t)} + R_1\,\underline{h}^{(t-1)} + \underline{b}_1 \;= \tilde{K}_1 \tilde{\underline{x}}^{(t)} \tag{2.13}$$

once we define

$$\tilde{K}_1 = [\; K_1 \mid R_1 \mid \; \underline{b}_1 \;]$$

and

$$\tilde{\underline{x}}^{(t)} = \begin{bmatrix} \underline{x}^{(t)} \\ \underline{h}^{(t-1)} \\ 1 \end{bmatrix}$$

This block matrix formulation highlights how all the weights can be grouped in a trainable matrix and all the stimuli in a (non-trainable) vector. It is moreover trivial

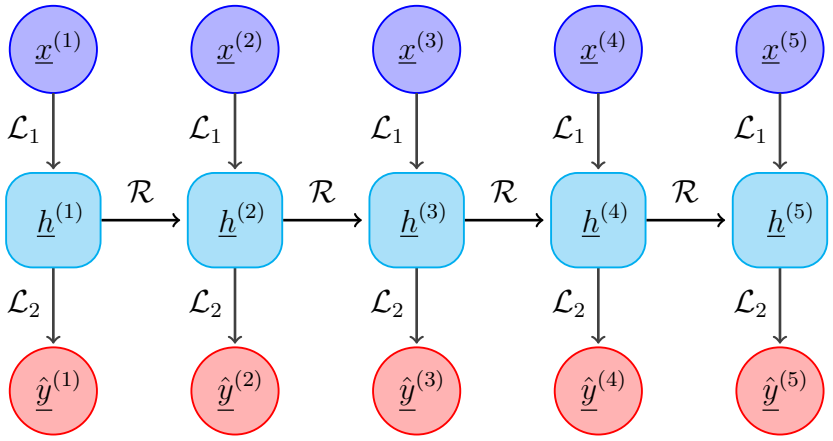


Figure 2.10. Unrolling a RNN. Because of their peculiar structure, RNNs can be thought as particular forms of FNNs.

to deduce that if no feedback connection is allowed, equation (2.13) reduces to the standard dense application that appears in FNNs.

In this regard, a crucial remark is that in most cases RNNs can actually be represented as feedforward networks: this is done through a procedure called *unrolling* of *unfolding* (Goodfellow et al. 2016, Chapter 10.1), which is illustrated in Figure 2.10. The obtained envelope is a very particular feedforward network: the trainable weights are somehow shared among the subunits, and indeed the same symbols have been used to denote the corresponding applications (again, the abuse of notation regarding $\mathcal{L}_1$ and $\mathcal{R}$ is adopted).

Moreover, the scheme highlights a matter that deserves special attention: in the beginning, we do not have an initial condition $\underline{h}^{(0)}$ for the hidden state. The most popular solution is to impose it equal to zero, a solution that anyway can introduce a bias if the hidden state is expected to be significantly different from the null vector. Other possibilities are to use the last available state, which formally belongs to a different sequence and therefore may assume nonsensical values as well, or to randomly generate an initial state; lastly, an interesting proposal that is worth mentioning is that the initial state can be learned as well as the other parameters, as suggested for instance in Forcada & Carrasco (1995).

2.2.3 Architectures for sequence modelling

Depending on the form that we require for the output, Figure 2.10 can offer a more or less appropriate representation of the network. Indeed in some situations one may be interested in getting as output not an entire sequence, but just a single element: a so-called *many-to-one* scenario. A field in which this happens is *sentiment analysis*, that for example aims at recognizing if the review of a product (i.e. a sentence) is positive or negative.

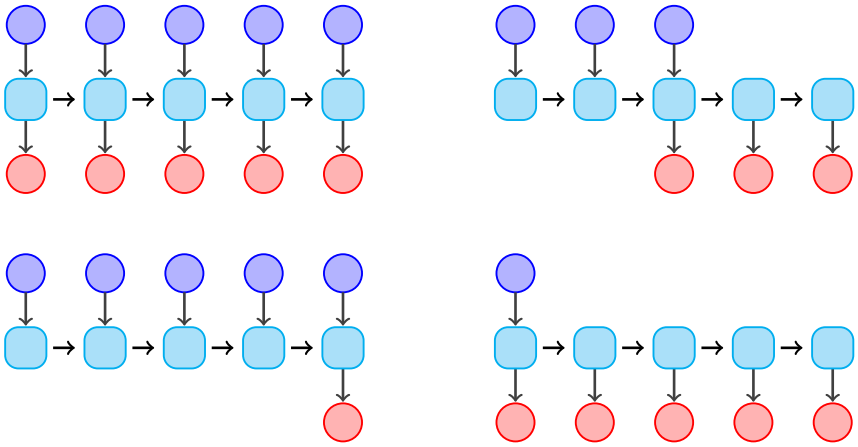


Figure 2.11. Different architectures for sequence modelling: many-to-many (*above left* and *above right*), many-to-one (*below left*), one-to-many (*below right*). The first two types of network transform sequences into sequences, while the others transform a sequence into a single vector and *vice-versa*.

Figure 2.11 shows the four typical architectures that are used in sequence processing (cf. e.g. Kapoor et al. 2019 and references therein). In music generation *one-to-many* architectures can be employed, in such a way that from an initial note a melody is produced. On the other hand, *many-to-many* architectures are necessary whenever a sequence needs to be transformed into another sequence. They come in the two depicted configurations: the one on the left is used for instance for classification tasks, when each input has to be properly labelled; for example in *part-of-speech tagging* each word has to be labelled as noun, verb, adverb... The one on the right is instead a structure used in language translation: in this case the whole sentence is formerly processed, and only then the output is produced.

An important remark is that the *loss function* has to be designed so as to match the type of output that the network generates. Therefore in principle slight differences in the training procedure can arise when different architectures are selected.

From now on the discussion will be focused on **time-series modelling**, because of the central importance they will have in the next chapters. In this case, data assume a very simple form: a time-series is nothing but a sequence of real-valued vectors.

2.2.4 The backpropagation through time

As FNNs, RNNs are commonly trained with gradient-based procedures; this entails that the problem of computing gradients is central also in this case. The technique of backpropagation that has been developed for feedforward networks cannot be used in principle, because the feedback links alter the topology of the network; anyway, we can take advantage of the feedforward representation that is

obtained by unrolling the RNN. In this initial stage, we analyse for simplicity the case a RNN with one hidden layer and a many-to-one architecture. In particular we consider the network in Figure 2.12, that processes a sequence with t elements and produces a final output $\underline{\hat{y}}^{(t)}$.

The idea is that the gradient can be backpropagated along with the unrolled graph. With reference to Figure 2.12, backpropagation occurs not only *vertically*, i.e. from $\underline{\hat{y}}^{(t)}$ to $\underline{x}^{(t)}$ as usual, but also *horizontally*, i.e. going backward *in time*: this means that we must take into account both the impact of the weights in the usual feedforward flow and the one associated with the recurrent connections.

In symbols, for the loss $\ell^{(t)}$, the one associated with $\underline{\hat{y}}^{(t)}$ and the corresponding true target $\underline{y}^{(t)}$, we can write

$$\frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\theta}} = \nabla^T \mathscr{L} \frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\theta}} = \nabla^T \mathscr{L} \frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{h}^{(t)}} \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{\theta}} \tag{2.14}$$

On the other hand, in view of the previous discussion it is clear that

$$\frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{\theta}} = \frac{\partial \underline{h}^{(t)}}{\partial \underline{\theta}} + \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(t-1)}} \frac{\mathrm{d}\underline{h}^{(t-1)}}{\mathrm{d}\underline{\theta}} \tag{2.15}$$

Assuming a structure of the form (2.12), we get

$$\frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(t-1)}} = \mathcal{A}_1' \Big|_{K_1 \underline{x}^{(t)} + R_1 \underline{h}^{(t-1)} + \underline{b}_1} R_1 \tag{2.16}$$

In this context, we are obliged to distinguish between *total* derivatives and *partial* derivatives of $\underline{h}^{(t)}$ with respect to $\underline{\theta}$. The meaning of the two is straightforward: the

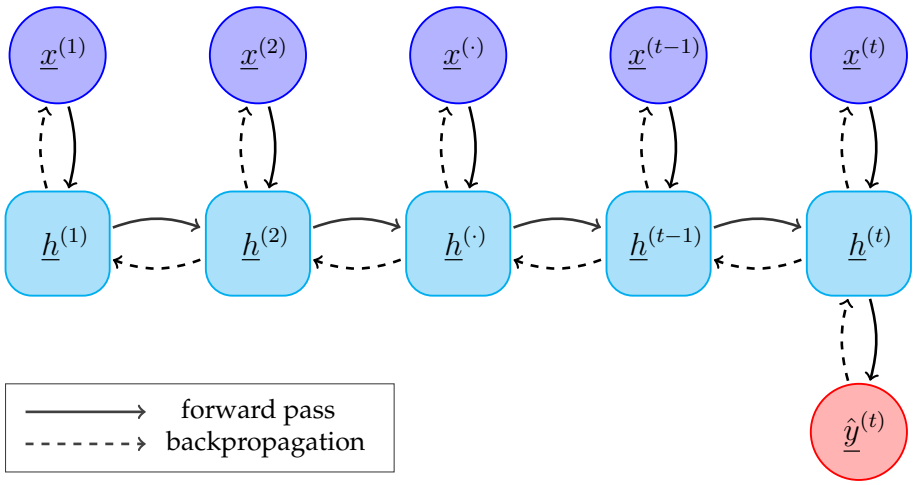


Figure 2.12. Backpropagation Through Time. The scheme highlights how the backpropagation procedure works in the case of a RNN. In detail not only the usual feedforward contribution is considered (i.e. the one associated with $\underline{x}^{(t)}$), but also the dependencies associated with the previous time-steps.

former indicates both the *horizontal* and *vertical* dependencies, as we defined them; the latter account only for the *vertical* dependency, i.e. the one associated with the processing of subunit t and not with the feedback connections.

Equation (2.15) anyway hides a recurrent relationship: indeed also $\underline{h}^{(t-1)}$ can be written in an analogous form. This in general holds until $\underline{h}^{(1)}$ is reached in the backpropagation, since that element has only a *vertical* dependency (unless the *initial condition* is parametric as well, but we do not treat this case in this thesis); thus, for this term, equation (2.15) becomes simply

$$\frac{\mathrm{d}\underline{h}^{(1)}}{\mathrm{d}\underline{\theta}} \equiv \frac{\partial \underline{h}^{(1)}}{\partial \underline{\theta}}$$

The final formula for this modified backpropagation is given by

$$\frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\theta}} = \sum_{k=1}^{t} \nabla^T \mathscr{L} \, \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{h}^{(t)}} \, \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(k)}} \, \frac{\partial \underline{h}^{(k)}}{\partial \underline{\theta}} \tag{2.17}$$

Since this algorithm involves also the dependency at previous time steps, it is known as **backpropagation through time**, or BPTT (Williams & Zipser 1995).

A couple of remarks should be made about equation (2.17). First, we emphasize that such formulation has the advantage of being compact and easily understandable, but for practical purposes it prescribes useless matrix multiplications. Indeed, when computing the k-th element of the sum, the following factorisation is exploited

$$\frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(k)}} = \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(t-1)}} \, \frac{\mathrm{d}\underline{h}^{(t-1)}}{\mathrm{d}\underline{h}^{(t-2)}} \, \cdots \, \frac{\mathrm{d}\underline{h}^{(k+2)}}{\mathrm{d}\underline{h}^{(k+1)}} \, \frac{\mathrm{d}\underline{h}^{(k+1)}}{\mathrm{d}\underline{h}^{(k)}} \tag{2.18}$$

Therefore, according to (2.17), the contribution of the k-th element is given by

$$\left(\nabla^T \mathscr{L} \, \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{h}^{(t)}} \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(t-1)}} \, \cdots \, \frac{\mathrm{d}\underline{h}^{(k+2)}}{\mathrm{d}\underline{h}^{(k+1)}} \right) \frac{\mathrm{d}\underline{h}^{(k+1)}}{\mathrm{d}\underline{h}^{(k)}} \, \frac{\partial \underline{h}^{(k)}}{\partial \underline{\theta}}$$

but the row vector in brackets is the same that appears when considering the contribution of the element $k+1$, that reads

$$\left(\nabla^T \mathscr{L} \, \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{h}^{(t)}} \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(t-1)}} \, \cdots \, \frac{\mathrm{d}\underline{h}^{(k+2)}}{\mathrm{d}\underline{h}^{(k+1)}} \right) \frac{\partial \underline{h}^{(k+1)}}{\partial \underline{\theta}}$$

In other words, again with regard to Figure 2.12, the procedure expressed in equation (2.17) works as follows: starting from $\hat{\underline{y}}^{(t)}$, the gradient is backpropagated until $\underline{h}^{(k)}$ is reached; once this is done, the procedure restarts again from $\hat{\underline{y}}^{(t)}$ and the gradient is now backpropagated until $\underline{h}^{(k+1)}$ (or $\underline{h}^{(k-1)}$, depending on the order) is reached, and so forth. Instead the advised procedure to avoid repeated multiplication is to follow the *natural path* of the backpropagation, i.e. recursively employing equations (2.14) and (2.15).

Second, given the large number of matrix multiplications in equation (2.18), typically of the form (2.16), Recurrent Neural Networks are likely to suffer from the vanishing or exploding gradient problem. In some sense, it is not a surprising fact because RNNs are not so different to deep FNNs, as we showed; hence in general they are not able to learn long-term dependencies, but they mainly rely on a *finite-context* learning. This fact clearly causes the training of RNNs to be a difficult task. More in detail, while exploding gradients are associated with some sort of instability of a network, or at least of the training procedure (cf. e.g. Pascanu et al. 2013), vanishing gradients are somehow related to NN stability. For instance, a recent work (Miller & Hardt 2019) proves that any *stable recurrent model*, i.e. a model that fulfils some Lipschitz-like stability conditions and is characterised by vanishing gradient, can be well-approximated by the corresponding *truncated* feed-forward model: in short, this is equivalent to say that such networks cannot have *long-term memory*.

As a consequence of the finite-context dependency, in many cases the **Truncated Backpropagation Through Time**, or TBPTT (Williams & Zipser 1995) is used in practice: this simply means that in equation (2.17), instead of backpropagating from time t to time 1, the algorithm considers just the last K_1 time steps. The equation that governs TBPTT(K_1) is thus:

$$\frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\theta}} \simeq \sum_{k=t+1-K_1}^{t} \nabla^T \mathscr{L} \, \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{h}^{(t)}} \, \frac{\mathrm{d}\underline{h}^{(t)}}{\mathrm{d}\underline{h}^{(k)}} \, \frac{\partial \underline{h}^{(k)}}{\partial \underline{\theta}} \tag{2.19}$$

If K_1 is suitably tuned so that the sum of the ignored terms is reasonably negligible, this algorithm is in practice equivalent to BPTT, but it is for sure advantageous from a computational perspective (in particular when K_1 is significantly less than the length of the sequence).

2.2.5 Other RNNs: LSTM and NARX

In this section we briefly mention two other types of RNNs that perform better than the standard RNNs (also called *vanilla* RNNs) on long-term dependencies. The fact that vanilla RNNs are not capable of learning long-term dependencies has been a central problem in literature (cf. e.g. Bengio et al. 1994 and Hochreiter 1998). Many applications require the knowledge of details which are distant in time; this has always been the main limitation of standard RNNs, which instead 'cannot bridge more than 5–10 time steps' (Gers et al. 2000).

Long Short-Term Memory (LSTM) networks are a class of RNNs with a particular structure that allows a greater persistence of the context-related information. Indeed, they are reported to be able to bridge time intervals of up to 1000 steps even in case of noisy input sequences, without exhibiting any loss of short time lag capabilities (cf. Hochreiter & Schmidhuber 1997). Their architecture is much more

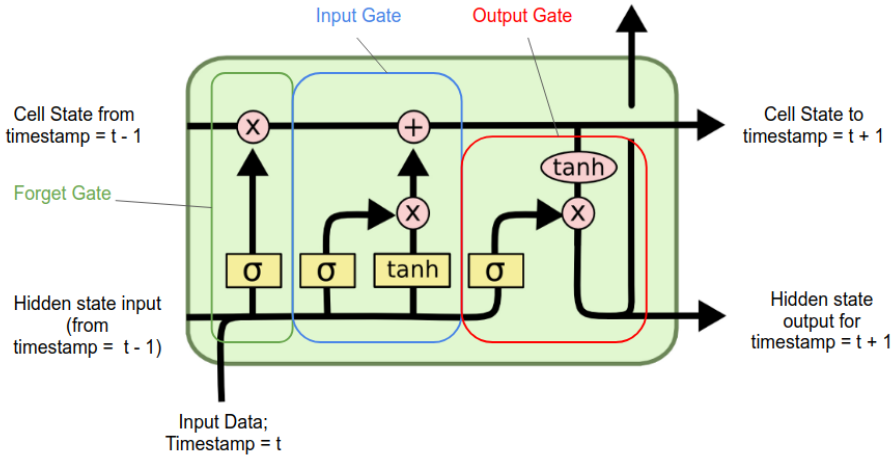


Figure 2.13. LSTM scheme. This complex cell is characterized by two different recurrent connections, the usual hidden state associated with the processing of the previous element of the sequence, plus an additional cell state that allows the persistence of the information.

complex than the one of *vanilla* RNNs, as shown in Figure 2.13. The interesting novelty introduced by LSTM is that two different recurrent connections exist: in addition to the hidden state $\underline{h}_t$, also $\underline{c}_t$ - called the *cell state* - is used as a feedback signal. This latter is indeed what allows the long-term permanence of the context details.

The overall functioning of a LSTM cell is governed by the following set of equations:

$$\begin{aligned} \underline{f}_t &= \sigma(K_f[\underline{h}_{t-1}, \underline{x}_t] + \underline{b}_f) \\ \underline{i}_t &= \sigma(K_i[\underline{h}_{t-1}, \underline{x}_t] + \underline{b}_i) \\ \tilde{\underline{c}}_t &= \tanh\big(K_c[\underline{h}_{t-1}, \underline{x}_t] + \underline{b}_c\big) \\ \underline{c}_t &= \underline{f}_t \odot \underline{c}_{t-1} + \underline{i}_t \odot \tilde{\underline{c}}_t \\ \underline{o}_t &= \sigma(K_o[\underline{h}_{t-1}, \underline{x}_t] + \underline{b}_o) \\ \underline{h}_t &= \underline{i}_t \odot \tanh(\underline{c}_t) \end{aligned} \tag{2.20}$$

In short, each cell is composed of three main parts: a *forget gate*, an *input gate* and an *output gate*. First of all, it is possible to notice that two different types of activation functions are present: the *sigmoid* and the *hyperbolic tangent*. The latter is actually used to transform the processed vector (as in FNNs or vanilla RNNs), whilst the former is always employed with the explicit purpose of transforming the processed vector into a vector of numbers between 0 and 1. Three vectors of this kind are produced: $\underline{i}_t$, $\underline{o}_t$ and $\underline{f}_t$.

Considering the equations in (2.20), in every case these vectors are involved in an element-wise product with another vector. For instance, the RHS of the (iv)

equation is formed by two blocks: the first one, i.e.

$$\underline{f}_t \odot \underline{c}_{t-1}$$

represents the fraction of the previous cell state $\underline{c}_{t-1}$ that has to be remembered. On the other hand

$$\underline{i}_t \odot \tilde{\underline{c}}_t$$

is the new part that will compose the cell state: it is composed of the product of the scale vector $\underline{i}_t$ and the *informative* vector $\underline{i}_t$, both obtained by processing the input $[\underline{h}_{t-1}, \underline{x}_t]$. The sum of the two contributions forms the state $\tilde{\underline{c}}_t$.

We will not go into deeper detail, since LSTMs will not be utilised in the following; anyway it is worth mentioning that some models in the energy forecasting sector make use of these structures, as well as of a similar type of recurrent networks called Gated Recurrent Unit (GRU) (cf. e.g. Bianchi et al. 2017 and references therein).

Nonlinear AutoRegressive eXogenous (NARX) networks (cf. Billings 2013) are another family of recurrent models that has gained significant importance in many fields of data analysis. Basically they constitute the nonlinear analogous of the well-known ARX models and are thus described by the equation

$$\underline{y}_t = F\big(\underline{y}_{t-1}, \underline{y}_{t-2}, \ldots, \underline{u}_t, \underline{u}_{t-1}, \underline{u}_{t-2}, \ldots, \big) \tag{2.21}$$

where F is a nonlinear function and each $\underline{u}_{t-k}$ is a (time-lagged) exogenous vector. In the general formulation of the NARX model, F can assume different forms, but in the case of NARX networks it is supposed to be a function described actually by a Neural Network, as shown in Figure 2.14.

In other words, NARX networks are built in such a way that there is a recurrent connection between the output and the first hidden layer. Nevertheless their power is that they allow a better description of temporal dependencies, in the sense that

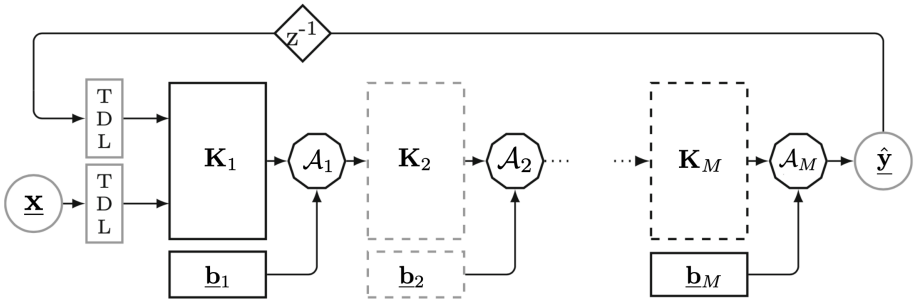


Figure 2.14. NARX network scheme. These recurrent networks are characterised by the existence of a feedback connection from the output layer to the input layer: in this way the model can take advantage of its previous predicted values. (Figure adapted from Bianchi et al. 2017, p.33)

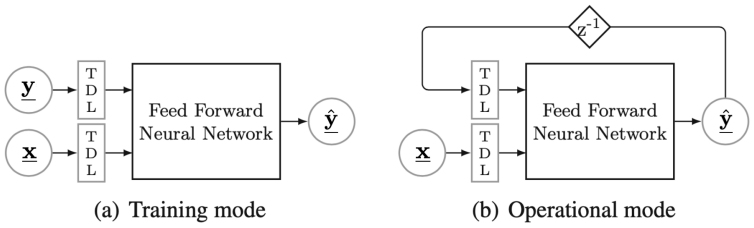


Figure 2.15. NARX network: training mode and operational mode. During training (*left*), the network is considered as a FNN and the true lagged values y_{t-k} are used as inputs. When the network is instead utilised for forecasting (*right*), the loop is closed and the model is fed with the previous inferred outputs. (Figure adapted from Bianchi et al. 2017, p.33).

not only the 1-lagged output $\underline{\hat{y}}_{t-1}$ is provided as input when computing $\underline{\hat{y}}_t$, but in principle an arbitrary number of the previous outputs can be used; in this regard, there are several studies (cf. e.g. Menezes & Barreto 2008) that show that NARX networks perform better than vanilla RNNs on predictions involving long-term dependencies.

Because of their peculiar architecture, NARX networks often are not trained with the backpropagation through time, but it is possible to exploit a particular strategy to learn the parameters $\underline{\boldsymbol{\theta}}$ of the NN. The procedure is shown in Figure 2.15: during the training phase, the output feedback is disconnected and the entire model is treated like a FNN. In this sense, the required ideal targets $\underline{y}_{t-k}$ are fed into the network instead of the corresponding inferred values $\underline{\hat{y}}_{t-k}$: this technique, which allows a great reduction of the training time, is known in literature as *teacher forcing* (cf. Goodfellow et al. 2016, Chapter 10.2.1). Notice that this is not possible for instance with vanilla RNNs, since the ideal values of the previous hidden states $\underline{h}_{t-k}$ are not retrievable by the training set. Once the training is over, the feedback link is reconnected and the network is used in *closed loop* to make predictions (cf. Bianchi et al. 2017).

In this chapter we have described the features of the Artificial Neural Networks, defining the notion of FNN and of backpropagation, explaining the typical algorithms used for their calibration and highlighting the issues that may arise during the training. Then we have focused on RNNs, which are more powerful, but also more complex kinds of networks that can be used for modelling sequences, and we have shown how the backpropagation algorithm works in for them. In the following chapter, we will show how RNN can be used for probabilistic load forecasting.

Chapter 3

Forecasting the Daily Demand

The most natural way to approach the problem of forecasting intra-daily energy demand is to understand how to model daily consumption effectively. What makes this problem simple yet challenging is the fact that daily energy consumption is characterized by a peculiar seasonal behaviour, that is strongly correlated with the climatic conditions (heating and cooling systems play a critical role in this); however, the standard time-series techniques, such as regressive and autoregressive models, are not capable of producing accurate medium-term predictions.

In recent years, the knowledge in the field of Probabilistic Load Forecasting has grown enormously, also thanks to the competitions that encouraged brilliant researchers to work on this subject; as a consequence, many interesting techniques and models were developed. In the present thesis, we consider as the main reference the NAX model, a NN-based architecture proposed in Azzone & Baviera (2021), which achieves remarkable results in forecasting the probabilistic distribution of daily energy consumption. In this chapter, we introduce the NAX model - which will be a fundamental building block for the following chapters - and describe it in detail, with a particular focus on its training phase. In this regard, we discuss and compare some different procedures that can be used to train the model and we show their impact on the accuracy of the predictions.

3.1 Preliminary analysis

We are used to having a daily routine in our lives: in spite of pretending to be unpredictable, we have a multitude of habits, especially when it comes to energetic consumption. It is not complicated to imagine that for instance the weekly energetic consumption of a person, whose working and social life lead to specific situations and actions, is incredibly similar to the last week's ones and to the next week's ones. This is in general true, of course, when we do not consider public holidays, which instead represent particular exceptions in the usual daily routine.

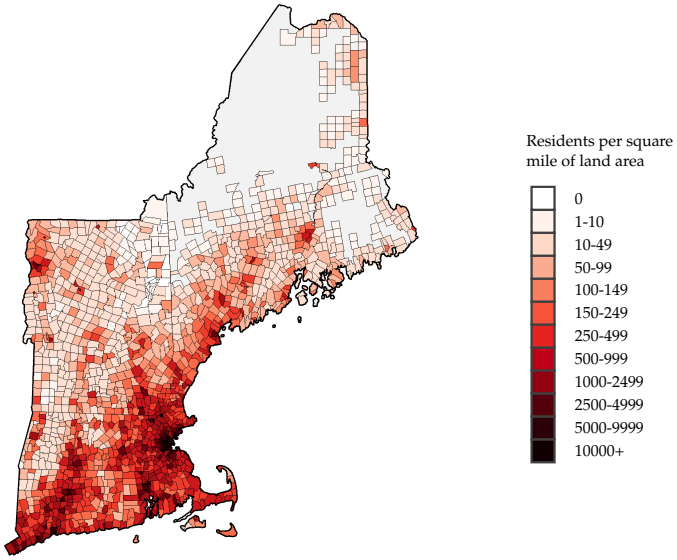


Figure 3.1. Population density of New England divided by municipality, based on the 2010 US Census data. It can be noticed that the region is characterized by an inhomogeneous density: the coastal zones are the most densely settled, whilst the internal areas are far less populated.

However, there is another big periodicity that characterizes our lives, and it is represented by the annual cycle. Different seasons mean different climatic conditions, and different climatic conditions lead to different energetic demands. Temperature and weather produce a huge impact on electricity consumption. In winter and summer, heat pumps and air conditioning are the biggest cause of elevated power consumption in the most developed regions of the world, while this phenomenon is less noticeable in autumn and spring.

3.1.1 Dataset introduction

The selected dataset is the one used for the GEFCom 2017 and is provided by *ISO New England Inc.*, the Regional Transmission Organization that coordinates the electric grid for the New England region, in the US; it contains the energy consumption profiles for the whole region, subdivided into eight bottom-level zones (cf. Hong, Xie et al. 2019).

New England is one of the most renowned zones of the United States, it is situated in the northeast of the country and it is composed of six states: Connecticut, Maine, Massachusetts, New Hampshire, Rhode Island and Vermont. In detail the dataset defines eight zones, each of them corresponding to a single State with the exception of Massachusetts, which is subdivided into three sub-regions; this is mainly due to the fact that it is the most populated State and it features the biggest town in the New England zone, Boston.

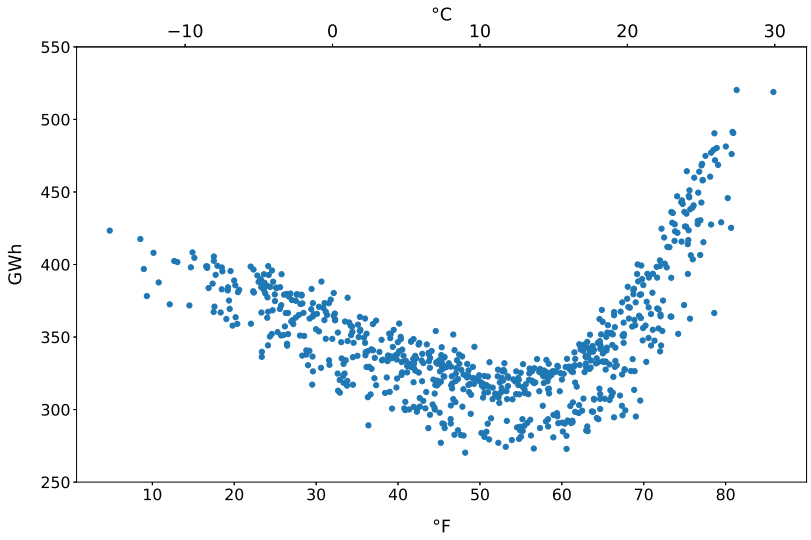


Figure 3.2. Scatter plot of dry-bulb temperature and consumption in 2009-2010 for the whole New England area. The distribution highlights how extreme temperatures cause an increase in energy consumption; moreover, the impact of high temperatures seems to be more pronounced.

As proposed in the work of Azzone & Baviera (2021), the electrical load data of the entire region are aggregated, despite the relevant territorial extension of the area. This approach ensures to have an average point of view of the power consumption of New England, which in principle is a region characterized by a large inhomogeneity, as Figure 3.1 shows. On the other hand, this should limit the possibility of having noisy data – associated for instance to zonal events or criticalities – and provide a more reliable picture of the overall consumption.

Climate effects are similarly local, they change from State to State and it is not so hard to imagine that the temperature could be so different in the eight zones on the same day. New England is indeed a region characterized by an unpredictable climate, there is a big difference between Atlantic zones (especially in the North) and inland zones. However, in order to be coherent with the consumption data, the average temperature for the entire New England is considered.

In detail, the dataset contains the following variables

- ▷ energy demand
- ▷ dry-bulb temperature
- ▷ dew point temperature
- ▷ hour and date

and, in addition, a list of the federal holidays is provided. Since the proposed variables are actually sampled on an hourly basis, energy demand is summed to

obtain the total consumption on each day, whilst temperatures are averaged on daily intervals. The resulting variables are therefore the sum over all zones and the 24 hours for what concerns the consumption, and the average over all the zones and over the 24 hours for what concerns the temperatures.

Figure 3.3 captures the combined effect of the two previously presented periodicities - one anthropic and one natural - that govern the daily energy use. The weekly routine is marked by an evident collapse of energetic consumption during the weekend, especially on Sunday, when many production and business activities are closed. People have different habits during these days, they may move from their usual houses and spend their time out, they wake up late and do not have to go to their workplace. These weekend habits have a big impact on the electrical demand, which will be even more significant in the intra-daily analysis of the following chapters.

The long-term behaviour is instead a consequence of the fact that climate changes during the year. The green dashed line in the plot is suitably able to describe the dynamics of demand; it is obtained by performing a linear regression of power consumption against two pairs of sines and cosines with a period equal to one year and six months, plus a linear trend term. Incidentally, this approach of modelling seasonality by means of sinusoidal functions is fairly common in this field and it will be discussed in the following.

3.1.2 Holiday impact

Holidays are another significant point to consider; despite being somehow periodical, they are not usually modelled as working days. The literature often define holidays and extraordinary events (like blackouts or strikes) as *intervention events* (cf. e.g. Guerini & De Nicolao 2015): this emphasises their uncommon nature, even under the perspective of energy demand. In particular, only federal holidays will be considered in this study. The energy demand is obviously lower during these days since that most of the business activities are closed. Interestingly, in several States, for instance, in Massachusetts, *blue laws* are in force; these old laws prescribe that the main part of business activities must be closed during federal holidays. Figure 3.4 shows the power consumption of every Monday of the year 2009; traditionally, in U.S. many *fixed-day* holidays, i.e. the ones that occur every year on a specific day of the week, fall on Monday, creating a three-day holiday called *long weekend*.

Anyway, energetic demand on public holidays - and public holidays as well - could be different from country to country (for instance, in Italy *fixed-day* holidays are rare since they are usually *fixed-date*, i.e. they occur on the same day of the year, like Christmas). They are somehow based on local customs, state laws and population's habits and thus there is not a unique common modelling strategy; as a

general rule, it is always advisable to model these days separately from the usual dynamics of power consumption (cf. e.g. Ziel 2018).

3.2 The NAX model

3.2.1 Overview

As mentioned in Chapter 1, the critical point of mid-term and long-term forecasting is that they mainly rely on the detection of trend and seasonality of a time series, just like the ones that were discussed in the previous section. Contrarily to the short-term forecasting, which is strongly based on what we can call the *recency effect* (i.e. the fact that the knowledge of today's demand is an important piece of information for predicting tomorrow's demand, as the two are clearly very close in time), mid and long-term forecasting cannot in principle do much more than extracting just the *ordinary* features of a time series. In this sense, it is difficult to imagine that, say, the exact knowledge of the energy load of April is useful for predicting the one of the next December; the longer time horizon clearly introduces a large variability and the seasonal behaviour is simply what survives. Essentially, this lack of long-term certainty is the main motivation behind the use of *weather dependent forecasting* (cf. Subsection 1.2).

This is the context in which the NAX model (Azzone & Baviera 2021) was conceived. It is a weather-dependent model that 'joints the advantages of classical univariate time-series analysis and a shallow NN': indeed NAX stands for *Neural Network with Autoregression and eXogenous inputs*. Despite some common features, this architecture should not be confused with NARX networks (cf. Subsection 2.2.5); the reason why NAX is a peculiar nonlinear autoregressive model will become clearer in the following. The main strength of NAX is that it is designed density forecasting, and in particular for the mid-term one.

As reported in Guerini (2016) - and references therein - a well-established approach to load forecasting is the *sequential* one, which consists in

- ▷ data transformation,
- ▷ detrending and seasonality removal,
- ▷ intervention analysis,
- ▷ modelling of the residual variability.

We will see in the next pages that NAX methodology follows exactly this scheme.

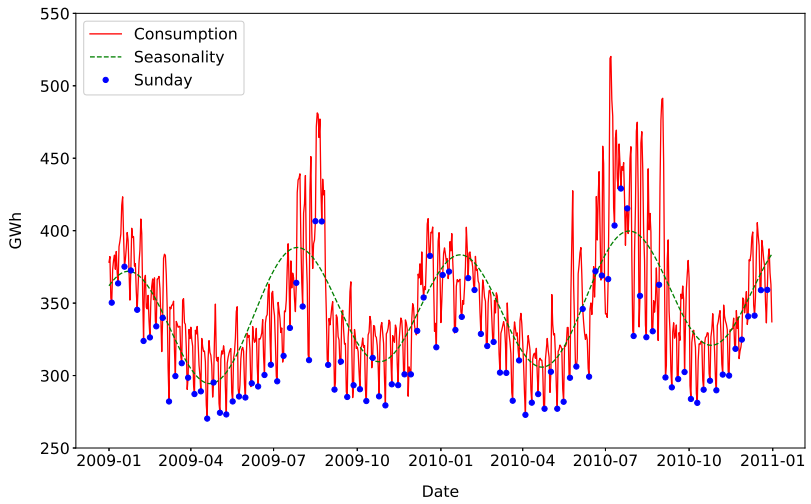


Figure 3.3. Daily aggregate energy consumption in 2009-2010. The profile exhibits a relevant seasonal pattern, with peaks in summer and winter. Furthermore, on Sundays, the consumption is considerably lower than the other days. (Figure adapted from Azzone & Baviera 2021).

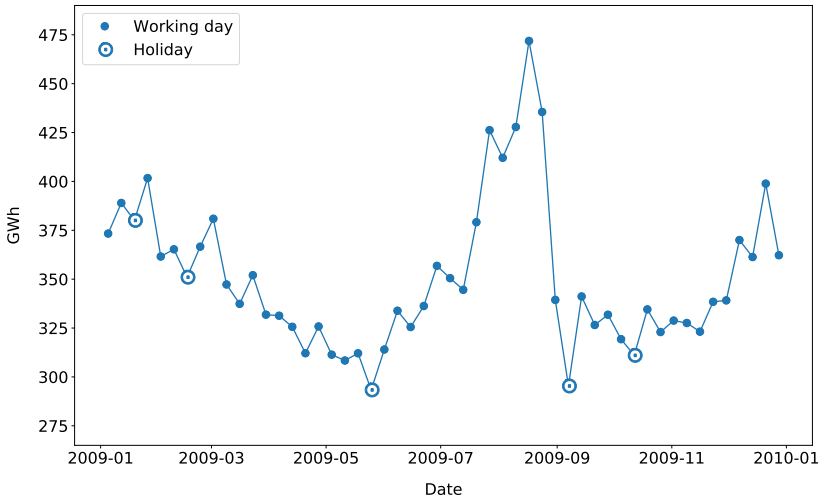


Figure 3.4. Holiday impact on Mondays of 2009. The plot highlights the relevant decrease in energy consumption during the holidays. During long weekends, indeed, the consumption on Monday is always far lower than the other Mondays.

3.2.2 Data transformation

It is a common standard in the literature to model the natural logarithm of the consumption (cf. e.g. Benth et al. 2008), also denoted as *log-consumption* in the following. As pointed out in the original paper, this is useful to tackle the seasonality in the observed *volatility* of the time-series; indeed energy load in summer is characterized by turbulent fluctuations and by the presence of huge spikes. Although it is not immediate to notice it in Figure 3.5, thanks to the logarithmic transformation the behaviour of log-consumption appears to be more harmonious; similarly, the resulting distribution is more balanced because of the compression effect that logarithm has on the right tail (Figure 3.6).

Furthermore, it is worth mentioning that in general the advantage of working with logarithms is that yearly and weekly seasonality - and also holiday effects - can be modelled additively, whereas they are multiplicative in the original time-series (Soares & Souza 2006).

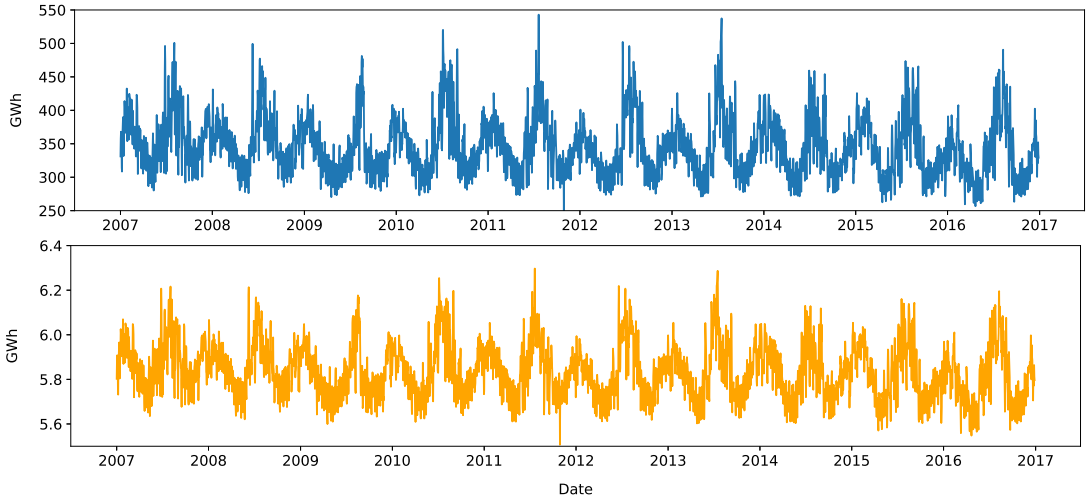


Figure 3.5. The original time-series of daily energy consumption for 2007-2016 (*above*) and the log-transformed one (*below*). The two plots do not show significant differences, but the series of log-consumption appears to be more regular and compact; in particular, the summer peaks are less pronounced in the second plot.

3.2.3 Trend, seasonality and intervention

According to Guerini (2016), additive models are usually specified as

$$Y_t = \text{Trend}_t + S_t + r_t \tag{3.1}$$

where Y_t is the logarithm of energy demand, Trend_t is a term that accounts for the linear trend, S_t is the seasonal term, that incorporates all the periodicities and the

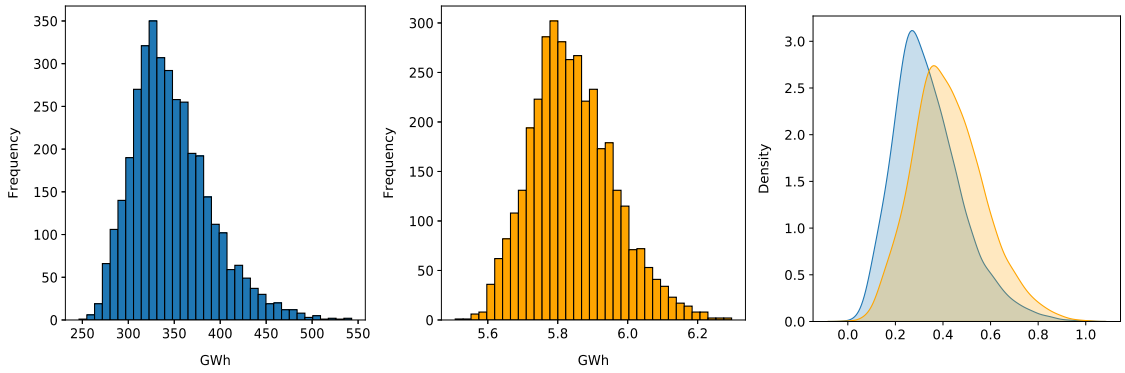


Figure 3.6. Histograms of consumption (*left*) and of log-consumption (*centre*). It is possible to notice that the original distribution is very skewed (actually resembles the PDF of a lognormal distribution) due to the presence of a heavy right tail, whilst the log-consumption exhibits a more centred and balanced distribution. The plot on the *right* instead compares the two densities after the two samples have been *min-max* scaled.

special days, and r_t is a stochastic process. Trend_t is here indicated in this way to avoid any risk of confusion with the (dry-bulb) temperature, indicated in the following as T_t.

Actually, the structure of NAX is completely coherent with this definition. The first two terms are expressed as follows:

$$\begin{cases} \text{Trend}_t = \beta_0 + \beta_1 t \\ S_t = \sum_{k=1}^{2} [\beta_{2k} \sin(k\omega t) + \beta_{2k+1} \cos(k\omega t)] + \beta_6 D_{Sat}(t) + \beta_7 D_{Sun}(t) + \beta_8 D_{Hol}(t) \end{cases} \tag{3.2}$$

The three $D_\circ(t)$ terms are dummy variables that hold one if day t is a Saturday, a Sunday or a holiday, respectively, and zero otherwise. The sinusoidal terms instead account for the yearly and half-yearly seasonality ($\omega = 2\pi/365$); in this regard, the authors decided to remove 29 February from leap years to preserve the seasonality structure.

Altogether, the presented terms form the deterministic part of of the model, which is meant to describe the seasonal behaviour of the electric load. In this regard, it is important to underline that the log-consumption profile is assumed to depend linearly on these factors, defining therefore a General Linear Model (GLM), which is then calibrated on the training data by means of Ordinary Least Squares (OLS). Despite its simple formulation, the model is able to describe the most evident behaviour of the energy consumption, i.e. the seasonal dynamics, and could in principle be used to make some initial rough forecasts. Figure 3.7 shows for instance the mid-term predictions for the year 2012, obtained by training the model on the data of years 2009-2011. However, the residuals of this regression, indicated in the following as r_t, are still relevant, in particular during the summer

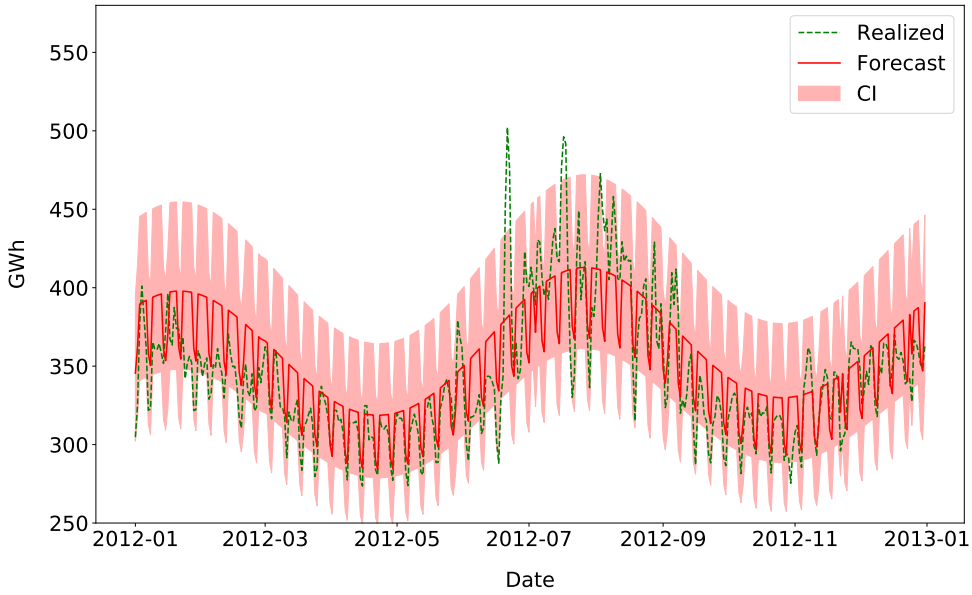


Figure 3.7. GLM predictions for the year 2012. The model is trained by using the data 2009-2011 as training set and the predicted profile is plotted together with the 95% confidence intervals. The GLM model is only capable of capturing the most evident part of the trend, i.e. the half-yearly seasonality and the reduction of consumption during the weekend.

months. As a consequence of this considerable variability that is not explained by this basic model, very large confidence intervals are generated.

3.2.4 Modelling the residual variability

At this point, the macroscopic features of the time series have been identified and removed. We should expect the residuals of the GLM model to be fairly autocorrelated as a result of the previously discussed *recency effect*. Indeed, the removed seasonality describes a long-term behaviour and thus it assumes almost the same value on two consecutive days; this entails that if on a day the true consumption falls relevantly above (or below) the predicted value, it is likely that this will happen even on the following day.

Formally, this aspect is investigated by means of the complete and partial autocorrelation functions reported in Figure 3.8: they show that the previous claim is actually correct. Moreover the plot of PACF allows in principle to deduce which lags are relevant to explain properly the observed phenomenon; in the case of a standard AR(p) model, we would have probably selected the first two lags. Since the autocorrelation at first lag is very pronounced, usual tests for the presence of a unit root are performed, that however lead to a negative response.

The central idea proposed in NAX concerns the form assigned to the stochastic

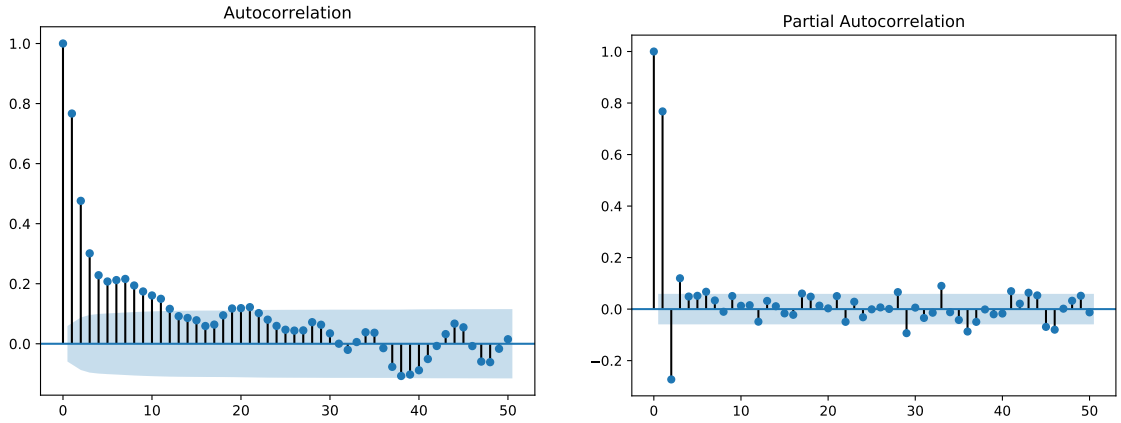


Figure 3.8. Total and partial autocorrelation functions for GLM residuals. The plots show an evident 1-lag autocorrelation, that cannot be captured by the GLM. According to PACF also 2-lag autocorrelation is considered statistically relevant, which is however far less pronounced than the 1-lag one.

process r_t and the way its parameters are estimated. Residuals are indeed assumed to be a collection of independent Gaussian random variables with mean μ_t and variance σ_t, both deterministic functions of time. This structure allows in particular great flexibility in describing the previously mentioned erratic behaviour of variance, which clearly requires appropriate heteroskedastic modelling.

The two time-series of mean and variance, that completely characterize the process r_t, are estimated through a Neural Network. Anyway, since serial dependency is a crucial feature of the residuals, a Recurrent Neural Network is employed for the occasion; its scheme is illustrated in Figure 3.9.

It has only one hidden layer and - as usually happens for regression problems - the *linear* activation function, i.e. the identity, is chosen for the output layer. The network produces as output a pair (μ_t, σ_t), respectively the estimated mean and standard deviation of the distribution of r_t; instead, every exogenous input vector is formed by the two temperatures contained in the dataset, namely the dry-bulb and the dew point, and eight calendar variables, which are the ones that appear in equation (3.2), i.e. the linear trend, the four sinusoidal terms and the three dummy variables. In addition, in order to replicate the evident one-lag serial dependency, a feedback connection from the output to the input is introduced in the model. This latter element constitutes the *autoregressive* part of the model.

As mentioned in Chapter 2, the training strategy is specified by choosing the loss function, because it establishes how errors are measured. In this case, a *maximum likelihood estimation* approach is used; in detail, for each time t, the adopted loss function is expressed by

$$\mathscr{L}(\mu_t, \sigma_t | r_t) = \frac{1}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2}\frac{(r_t - \mu_t)^2}{\sigma_t^2} = -\log\varphi(r_t, \mu_t, \sigma_t) \tag{3.3}$$

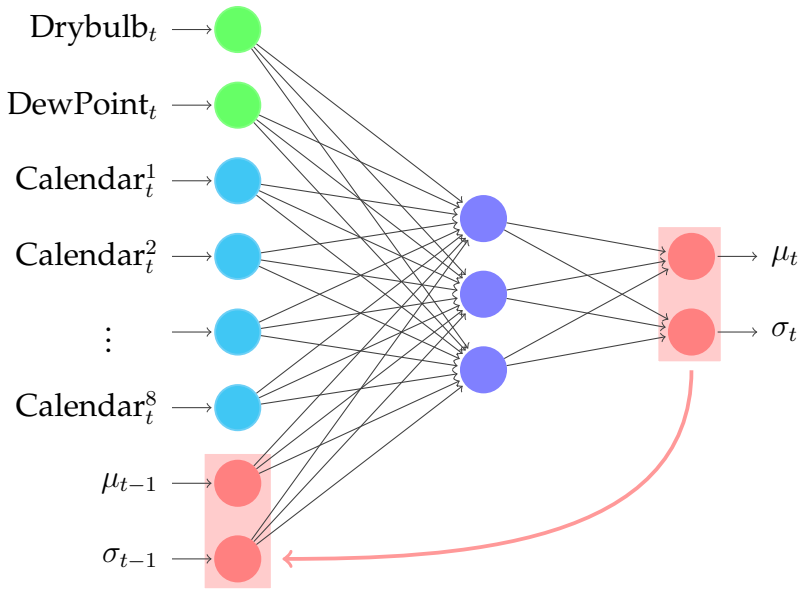


Figure 3.9. Scheme of the Recurrent Neural Network that is employed in NAX model. The output of the previous day, composed by the mean and the standard deviation of the predicted distribution, is used as autoregressive input for the following day.

where $\varphi(x, \mu, \sigma)$ is the value at x of the normal PDF with mean μ and standard deviation σ. Thus, it should be clear that, because of stochastic independence, minimizing the loss function is equivalent to maximizing the corresponding Gaussian log-likelihood.

As a final note, we would like to highlight that this is one of the cases in which the output of the network (μ_t, σ_t) and the true target r_t are elements of different Euclidean spaces (cf. Subsection 2.1.5), being the first a vector in the 2D space and the second simply a scalar. In this sense, we do not want the output to be as close as possible to the target in a usual L^p norm sense, but a probabilistic framework is adopted. Incidentally, in addition to the entire methodology that has been described above, this also represents the main differences between this model and usual NARX models: μ_t and σ_t are not available data, but are obtained by processing the incoming information. Trivially, a training in open-loop form (cf. Subsection 2.2.5) could not be performed, since - with regard to Figure 3.9 - the pair $(\mu_{t-1}, \sigma_{t-1})$ is not known unless we process the inputs for day $t-1$ (but this, in turn, would require the output of the previous day, and so forth).

3.3 Training the RNN

3.3.1 Preparation of the dataset

As described in Chapter 2, RNN models are trained to make predictions based on a sequence of consecutive samples from the data. However, when dealing with time-series, it often happens that the available data come in the form of a single long sequence of past observations. In order to obtain a training set that can be handled by a RNN, a typical procedure is to use a *sliding window* approach, i.e. to split the long sequences into a family of sub-sequences of fixed length, each one shifted in time with respect to the previous one, as in Figure 3.10. Each of these sub-sequences, equipped with the corresponding target, represents a single training case. For example, this approach is adopted by Gasparin et al. (2019) and Bianchi et al. (2017).

When this methodology is considered, two further hyperparameters have to be taken into account: the *length* of each sub-sequence and the *time-shift* between two consecutive sub-sequences. Concerning the latter, unless exceptional cases, it is a common practice to set it equal to one; this is mainly due to the fact that, clearly, setting a larger time-shift parameter implies that fewer windows are obtained and so fewer training cases are made available to the network. In principle, the larger is the time-shift, the larger is the risk of losing some important pieces of information.

The length of the sub-sequences, instead, has no default value and there are no specific rules-of-thumb to select it. Its choice is driven by two aspects:

- ▷ if the sub-sequence is too short, some important information about the context may be lost;
- ▷ if the sub-sequence is too long, the training procedure is can be very expensive under the computational perspective, because of BPTT.

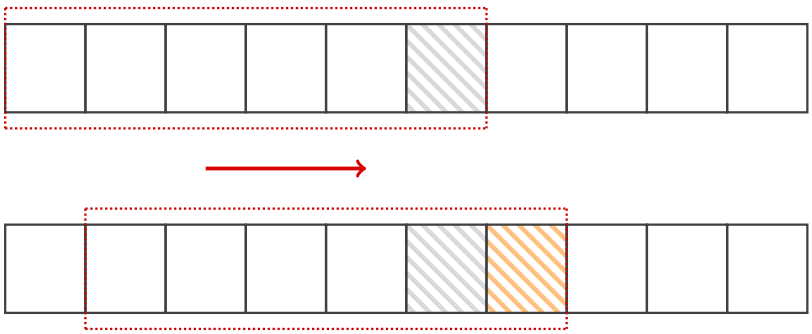


Figure 3.10. Generation of sub-sequences: in this case, a unitary time-shift is selected. Clearly, the total number of windows that can be created depends both on the length and the time-shift thereof.

This trade-off has always to be considered when designing an RNN, or better when preparing the data. Anyway, the intuition behind the choice of sequence length is that it does not alter substantially the kind of information contained in a sub-sequence. Instead, it is mainly concerned with how frequently the hidden state is reset; in this regard, we recall the discussion on the initialization of hidden states in RNNs (cf. Subsection 2.2.2). In accordance with the default choice of many Deep Learning libraries like `Keras` (cf. Kapoor et al. 2019, Chapter 8), in the following, we always assume that the hidden state is initialized to zero whenever a new sub-sequence is analysed.

In the present case, windows are composed by a sequence of length L of vectors of size 10 (each formed by the two temperatures and the eight calendar variables): they are thus matrices of data. As a consequence, the training set can be thought a 3D tensor, since it is formally an ordered set of windows. On the other hand, every target is just a scalar, i.e. the true residual corresponding to the last element of each sequence. In this sense, it is implied that a many-to-one architecture is selected for the occasion.

The initial choice is to consider windows of length two, in an attempt to focus just on the 1-lag autoregressive dependency and to fully exploit this very-short-term piece of information; in Section 3.4 we discuss instead the impact of longer dependencies by considering wider windows.

Moreover, to conclude the data pre-processing, the data are min-max normalized, so that all the features are in a comparable range and improve the convergence of the training algorithm; in detail, this applies for the two temperatures, which assume values on a different scale with respect to the other variables.

3.3.2 Implementation

As explained in the paper, the original implementation of NAX is done in `Keras` (cf. Chollet et al. 2015) with `Tensorflow` as backend. The simplicity of the interface is for sure a point in favour of this API, which is indeed extremely popular in the world of Deep Learning; on the other hand, it has the disadvantage that networks like the one depicted in Figure 3.9 are not built-in structures and must be defined *ad-hoc*.

`Keras` allows by default the construction of sequential networks by means of a specific *Sequential* API, which for many practical cases can be enough; in particular, this API offers the possibility to create *vanilla* RNNs, in which the recurrent connections are always intended to be *layer-wise* (see Section 2.2.2) and not - for instance - from the output layer to the hidden layer, as NAX requires. In order to implement more complex networks, one can either use the *Functional* API, which is used to create flexible models characterised for instance by non-

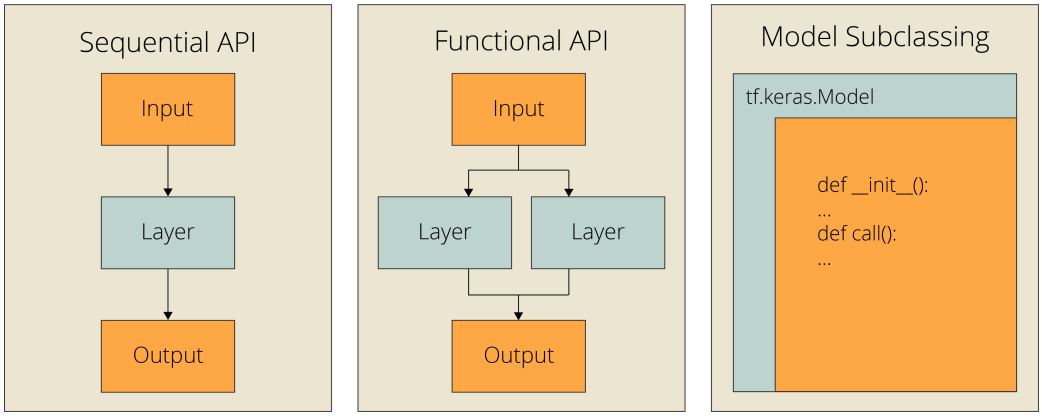


Figure 3.11. Keras APIs for models implementation. The *Sequential* API allows the creation of the most general networks; anyway, non-conventional models can be implemented using *Functional* API and *Model Subclassing*.

sequential topology, or resort to a most low-level alternative: Model Subclassing, i.e. implementing tailor-made derived classes and overriding the built-in methods. The main problem with these last two possibilities is that often training times become very long since the sophistication of the model has a direct impact on the backpropagation procedure (the same principle for which BPTT in RNNs is more computer-intensive than backpropagation in FNNs).

The structure of the network used in the NAX model has anyway a relevant peculiarity. It is represented by the equation

$$\underline{\hat{y}}^{(t)} = K_2 \underbrace{\mathcal{A}_1\left(K_1 \underline{x}^{(t)} + R_1 \underline{\hat{y}}^{(t-1)} + \underline{b}_1\right)}_{\underline{h}^{(t)}} + \underline{b}_2 \tag{3.4}$$

and thus

$$\underline{\hat{y}}^{(t)} = K_2 \, \mathcal{A}_1\left(K_1 \underline{x}^{(t)} + R_1 K_2 \underline{h}^{(t-1)} + R_1 \underline{b}_2 + \underline{b}_1\right) + \underline{b}_2 \tag{3.5}$$

In other words, since the output is an affine transformation of the hidden layer, it is possible to write the hidden state $\underline{h}^{(t)}$ as a function of $\underline{h}^{(t-1)}$:

$$\begin{aligned} \underline{h}^{(t)} &= K_1 \underline{x}^{(t)} + R_1 K_2 \underline{h}^{(t-1)} + R_1 \underline{b}_2 + \underline{b}_1 = \\ &= K_1 \underline{x}^{(t)} + R_1^* \underline{h}^{(t-1)} + \underline{b}_1^* \end{aligned}$$

This shows that the network is actually equivalent to a *vanilla* RNN with a fully recurrent hidden layer. As a consequence, the *Sequential* API that was mentioned before can be used for implementing such a structure.

3.3.3 Calibration, validation and testing

Once the network has been created and implemented, the actual training takes place: this is the moment in which the model learns the features of the residuals. In

`Keras`, the training is done through a built-in method, which actually implements the steps that are described in Chapter 2. In this regard, Adam is selected as optimizer.

As usually happens in these cases a procedure called *grid search* is performed: many combinations of the hyperparameters are considered and the corresponding results are evaluated and compared in order to identify empirically the best one. In Table 3.1, are reported the ones that are selected according to the paper. This phase actually represents the validation of the model.

As the last step, the obtained model is then trained on subsequent years and used to forecast the mid-term load and, thus, to test its effectiveness; as an illustration of the overall result, in Figure 3.12 are plotted the predicted values for the year 2012. For more precise details of the testing phase, we refer the interested reader to the original paper.

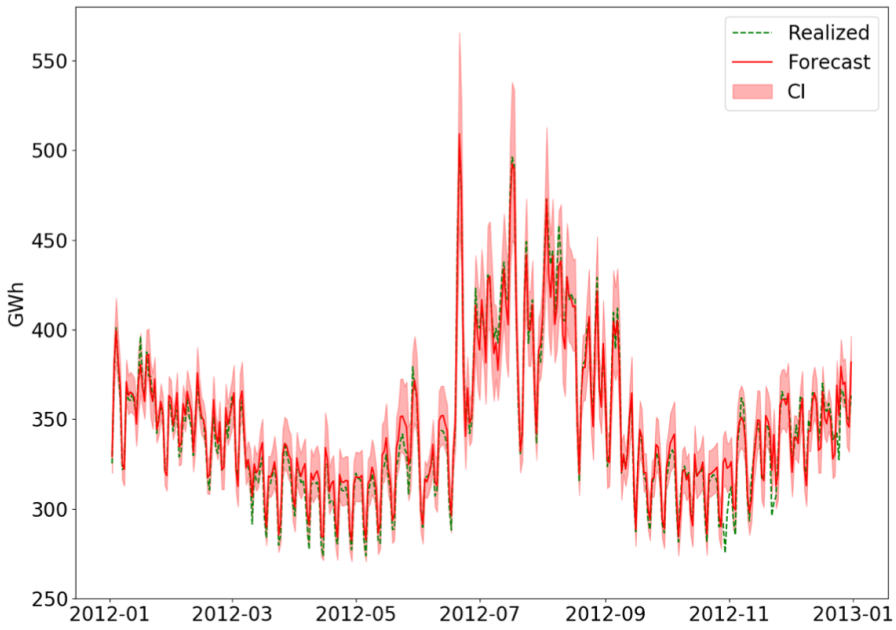


Figure 3.12. NAX *ex-post* predictions for the year 2012. The model is trained by using the data 2009-2011 as training set and the hyperparameters identified in Table 3.1. The predicted profile is plotted together with the associated 95% confidence intervals.

3.4 Improving NAX performances

Among people that work ordinarily with Neural Networks, there is a popular saying that goes: deep learning is an art rather than a science. This quote is meant to resemble how deep learning models are difficult to be treated: they are

Hyperparameter	Value
Hidden neurons	3
Activation function	Softmax
Learning rate	0.003
Batch size	50
Regularization parameter	0.0001
Time range of training set	3 years

Table 3.1. Best-performing configuration of hyperparameters for NAX model.

too complex to be studied analytically and usually we do not have particularly formal ways of explaining why an architecture is better than another. In this section, we investigate some aspects of the training of the RNN employed in the NAX model and we propose modifications that are useful to improve the predictive performances: in particular, in Subsection 3.4.4 the results obtained by best-performing configurations are compared to the results provided in the original paper that introduces NAX.

3.4.1 The impact of Window Length

In the first place, the incidence of the *window length* is discussed. As previously mentioned, this hyperparameter is related to the context that the network can make use of to produce each prediction: the larger is the size of each window, the more detailed is the information contained in it. This is obvious in principle, but in practice, it is not always convenient to use windows that are too large. In the present case the reason is twofold: first of all, the vanishing gradient problem prevents the network from learning long-term dependencies; secondly, the autocorrelation plot in Figure 3.8 shows that actually long-term dependencies are not relevant for explaining the residuals.

In Figure 3.13 is reported an evaluation of the forecast performance of NAX when the window length is set equal to 2, 5 and 10 respectively. Two accuracy metrics, MAPE and RMSE (cf. Subsection 1.4), are considered for the purpose: in both cases selecting a window length equal to 5 seems to be the most appropriate choice, which is expected to produce - on average - the best predictions. This is somehow consistent with what has been said before, i.e. that truncating the sequence too early causes the network to be blind to long-term temporal dependencies and that on the contrary, the learning becomes more difficult if the window is excessively long. Nevertheless, these results are in general to be taken as a tendency, since the training procedure depends considerably on the initial values of the weights of the

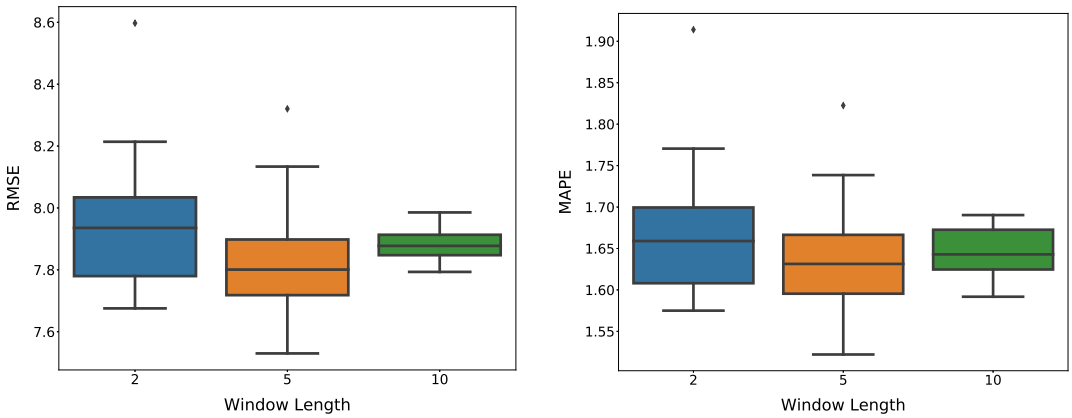


Figure 3.13. Boxplot of RMSE (in GWh) and MAPE (in %) against three window lengths. The values have been obtained by training the NAX model 10 times for each window length, with different initial conditions. The training set is composed of the data 2009-2011 and the accuracy of the corresponding forecasts for 2012 is evaluated; the selected hyperparameters are the ones in Table 3.1 with the exception of the learning rate, which is set equal to 0.001 to obtain more stable performances. Moreover, the training set is shuffled at the end of every epoch (cf. Subsection 3.4.2).

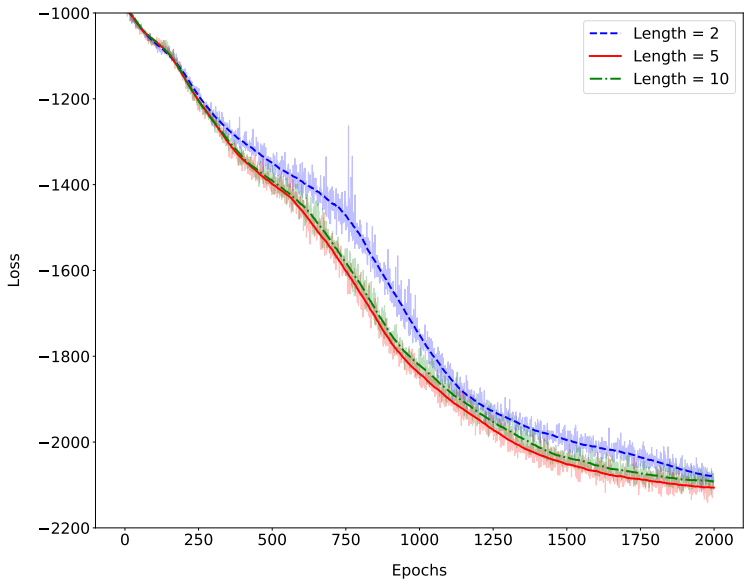


Figure 3.14. Evolution of *testing* loss for different choices of window length. Data and settings are the same as in Figure 3.13 and, similarly, 10 replications with a different random seed are considered: the shaded profile represents the average loss for each epoch. In order to obtain smoother curves, a running average is then computed and plotted.

NN, which are randomly selected.

3.4.2 The impact of Random Shuffling

Another aspect that is fundamental for practical applications is the *random shuffling* of the training data, which was briefly mentioned in Subsection 2.1.6. In many cases, shuffling the training set and creating new mini-batches at the end of each epoch can sensibly improve both the convergence of the optimization algorithm and the ability to generalize of the obtained model; the underlying reason is that shuffling data serves the purpose of reducing variance. In particular, this becomes a very relevant point when data are somehow sorted, since the gradient that is computed on a mini-batch may be a very biased estimate of the real gradient ∇J that represents the ideal descent direction: therefore, mixing the training set during the pre-processing of the data is always a recommended practice (cf. Bengio 2012).

Anyway it is common to do it also after each epoch: on the one hand to limit the risk of having many mini-batches are not representative of the overall dataset - and thus helping the optimization algorithm to escape from suboptimal local minima - on the other, because it should ensure that the model is less prone to overfitting since a new amount of randomness is introduced at every epoch.

In the case of NAX, both the convergence speed and the performances of the training algorithm are definitely increased with this procedure. In particular, a comparison between the average convergences of the same network, with even the same initialization, is shown in Figure 3.15. When data are not shuffled, many more epochs are required for convergence and the testing loss (but the same thing holds for the training loss) is greater than the one obtained by mixing the training set; this is mainly caused by the fact that in these cases the optimization algorithm gets stuck in sub-optimal local minima. On the other hand, the randomness associated with shuffling generates more volatile results, as highlighted by the wide shaded profile in the plot; the convergence is much smoother and more regular in the other case, and usually continuous slow improvements occur for thousands of epochs.

3.4.3 Continual operation and RTRL

The last modification we analyse is related to the algorithm used for training the network. The underlying idea of the following discussion is that instead of splitting the training set into a multitude of windows, we would like to consider the network as a *continually running* system.

In this regard, the distinction between the *epochwise*[1] and the *continual* operation

[1] In this context, *epochwise* should not be confused with the notion of *epoch* given in Subsection

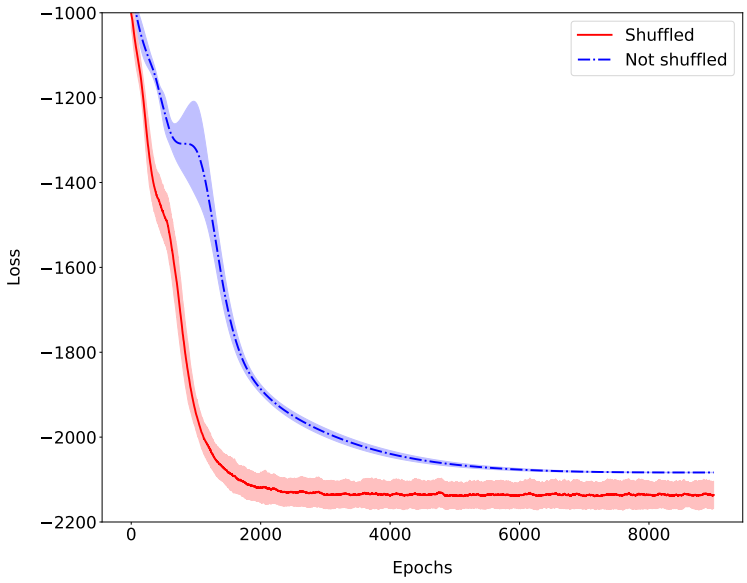


Figure 3.15. Evolution of *testing* loss with and without random shuffling. Four replications of the training are performed with the same hyperparameters used in Figures 3.13 and 3.14; the shaded area spans one standard deviation. Furthermore, in accordance with the previous results, the window length is set equal to 5.

of a recurrent network is an important concept in Deep Learning literature (cf. e.g. Williams & Zipser 1995, Schmidhuber 1992a and Williams & Peng 1990). *Epochwise* operation means that 'the network is run from some particular starting start until some stopping time is reached, after which the network is reset to its starting state for the next epoch' (cf. Williams & Zipser 1995): this is exactly the case that has been considered up to now, with the sliding windows approach. When dealing with epochwise operation, it is important to ensure that the new state at the start of the new epoch is unrelated to the state at the end of the previous epoch.

Conversely, a network is considered to operate continually if the hidden state is never reset, but it persists in time. In our case, what we mean by continually running network is that the RNN is supposed to process the available weather (and calendar) data as a single long sequence, without splitting it into smaller windows of few days. In this way, the model is expected to produce each forecast using the entire history available up to that time.

While in the case of *epochwise* operation BPTT is arguably the most used algorithm for differentiation, *continual* operation requires in principle a different approach: since its hidden state at time t is a function of *all* the previous history, BPTT becomes a very computer-intensive algorithm as soon as the number of time-steps gets large. Even for a *vanilla* RNN with a single hidden layer, after 1000 time-steps (i.e. more or less at the end of the three years of the training set of NAX) backpropagating the gradient is a very extreme task: basically it is not so different

2.1.6.

from training a FNN with 1000 hidden layers.

The most interesting algorithm designed to overcome this problem is called **Real-Time Recurrent Learning** (or RTRL, cf. Williams & Zipser 1995). It is based on the following principle: let us consider a generic recurrent network of the form

$$\underline{\hat{y}}^{(t)}(\underline{\theta}) = f\big(\underline{\theta},\ \underline{\hat{y}}^{(t-1)}(\underline{\theta})\big) \tag{3.6}$$

Then the Jacobian matrix of $\underline{\hat{y}}^{(t)}$ with respect to $\underline{\theta}$ can be written as

$$\frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\theta}} = \frac{\partial f}{\partial \underline{\theta}} + \frac{\partial f}{\partial \underline{\hat{y}}^{(t-1)}} \frac{\mathrm{d}\underline{\hat{y}}^{(t-1)}}{\mathrm{d}\underline{\theta}} \tag{3.7}$$

which, in other words, means that the Jacobian matrix at time t is a function of quantities that can be calculated at time t (i.e. $\partial f/\partial \underline{\theta}$ and $\partial f \Big/ \partial \underline{\hat{y}}^{(t-1)}$) and of the Jacobian matrix that has been computed at the previous time-step.

As known, the loss at time t is defined as

$$\ell^{(t)} = \mathscr{L}\big(\underline{y}^{(t)},\ \underline{\hat{y}}^{(t)}\big)$$

entailing that the corrisponding gradient is given by

$$\frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\theta}} = \nabla^T \mathscr{L}\ \frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\theta}} \tag{3.8}$$

Similarly, also the term $\nabla^T \mathscr{L}$ is a quantity that depends only on the values of $\underline{y}^{(t)}$ (the realized ones) and $\underline{\hat{y}}^{(t)}(\underline{\theta})$ (the forecasted ones).

Therefore, by suitably storing in memory the last computed Jacobian matrix, the differentiation algorithm can operate in *real-time*, in the sense that all the terms that are required for deducing the gradient of the loss can be computed at time t: there is no longer the need for unrolling the network.

One may notice that the previous equations are basically equivalent to (2.14) and (2.15), which were found when developing the BPTT. The real novelty introduced by RTRL is associated with the order in which the matrix multiplications are performed: indeed firstly $\mathrm{d}\underline{\hat{y}}^{(t)} \Big/ \mathrm{d}\underline{\theta}$ is computed, and only then the gradient $\nabla^T \mathscr{L}$ is calculated and the two terms are multiplied together.

In BPTT the gradient is always backpropagated from the loss to the inputs. As underlined in Chapter 2, the reason why this algorithm is executed in that way is for computational purposes, since it allows to transform the series of matrix multiplications into a series of matrix-vector products. However, this fact becomes a big disadvantage when we need to compute the gradients for many consecutive time-steps: in this case, several different iterations of BPTT have to be run, each one starting from the loss at the considered time-step and going backwards.

RTRL is instead designed also to solve this issue because exploiting equation (3.7) makes it possible to calculate the gradients in real-time. The drawback is a greater complexity in time because, actually, RTRL works with Jacobian matrices and not with vectors: this makes the algorithm tractable only for the smallest networks (as the ones considered in this thesis.) A simple estimate for *vanilla* RNNs is that the computational cost of each iteration of RTRL is $\mathcal{O}(k^2|\underline{\theta}|)$ where k is the size of the recurrent state and $|\underline{\theta}|$ the number of weights; thus when a sequence of length N is processed, the associated cost is $\mathcal{O}(Nk^2|\underline{\theta}|)$. On the other hand, BPTT has a time complexity that can be quantified as $\mathcal{O}(N|\underline{\theta}|)$ (cf. Menick et al. 2020). In the case of deep and wide networks, this difference is extremely relevant and therefore RTRL in its original formulation cannot be applied; anyway, its advantages have led in recent years to a search for more approximations that retain its desirable properties (cf. e.g. Tallec & Ollivier 2017 and Menick et al. 2020).

Nevertheless, for the RNN used in NAX, the complexity of the two algorithms is actually comparable, since the recurrent state has size 2. For the occasion, the network has been implemented in C++ , as `Keras` and `Tensorflow` - as well as the main Deep Learning API - do not offer RTRL as a built-in algorithm because of its lack of versatility.[2] Further technical details about the derivation of formulae are provided in Appendix B.

The obtained network is then tested against its epochwise operating version: both are trained on the data of 2009-2011 and the corresponding forecasts for 2012 are evaluated. In order to have a fair comparison, the epochwise operating network is never allowed to shuffle the training set; indeed by construction RTRL processes the entire time series sequentially. The results are reported in Table 3.2: it is possible to notice that no relevant differences between the two methods are detected. This shows firstly that the truncated sub-sequences used in the epochwise approach, obtained by means of the sliding window, contain a suitable amount of information; the knowledge of long-term context is not considered useful in explaining the phenomenon, or simply that the RNN is not capable of exploiting it because of its structure. Secondly, this shows that feeding the network with a null initial state does not introduce a significant bias.

According to these results, in the following we will consider the continually running version of the network. This is mainly due to the fact that, despite theoretically RTRL is expected to be a sub-optimal algorithm, the C++ implementation has proved to be much faster than the one of `Keras`.[3]

[2] Actually `Keras` allows the creation of continually operating recurrent networks by means of an optional parameter `stateful`; anyway they are trained using TBPTT (cf. e.g. Kapoor et al. 2019, Chapter 8).

[3] The measurements are made on a Intel CPU i5-5257U: using the IDE `PyCharm 2020.2.3`, `Python 3.8`, `Keras 2.3.1` and `Tensorflow 2.3.1`, the average time required for processing 1000 epochs is 32 seconds; instead 6 seconds are required by of C++ 14 compiled with `clang 12.0.0` and the option `-Ofast` enabled. Moreover the number of epochs that are required to con-

	Continual			Epochwise w/o shuffling		
Learning rate	MAPE	RMSE	APL	MAPE	RMSE	APL
0.001	1.73	8.06	2.14	1.71	8.13	2.13
0.0008	1.70	7.97	2.11	1.70	8.04	2.10
0.0005	1.68	7.93	2.08	1.66	7.89	2.06

Table 3.2. Results of epochwise and continual operation (without shuffling) in forecasting 2012 load. MAPE is expressed as a percentage, RMSE and APL in GWh. Both networks are trained with the parameters indicated in Table 3.1, with the exception of the learning rate which is suitably specified; lastly, for epochwise operation a window length of 5 is selected.

This is probably due to overhead and to the reduced functionalities of the former, which is instead optimized for the specific network of NAX. However it is important to remark that epochwise operation networks should in general be preferred if there is the possibility to run the code in parallel: in this case, each sub-sequence can be processed independently, whilst continually running networks rely on a sequential approach that prevents the parallelization.

To conclude, a relevant remark. A last important thing should be mentioned about RTRL: if the weights are updated while the network is running - which is the most natural use of the algorithm - the gradient of the loss is no longer exact, but becomes an approximation; indeed at each time-step, the Jacobian matrix is computed with the real-time weights. This entails that after an update of the weights, the previously computed Jacobian matrices are an estimate of the exact ones; anyway, the practical differences are often slight and become even slighter as the learning rate is made smaller (cf. Williams & Zipser 1989). In this regard, one should remember that, when working with mini-batches, also the gradient computed on each mini-batch is an approximation of the true gradient ∇J.

3.4.4 Results

In the light of the findings of the previous subsections, we select two final configurations for the Recurrent Neural Network adopted in NAX. The first is the original epochwise operating network, which is allowed to shuffle the training set, and is considered with the usual choice of hyperparameters (cf. Table 3.1), learning rate equal to 0.001 and window length of 5 time-steps. The second is instead the continually operating network trained with RTRL, using the same hyperparameters

verge are approximately the same for the two approaches, making the RTRL algorithm implemented in C++ the most convenient choice for the case.

	Continual			Epochwise w/ Shuffling			Original		
Year	MAPE	RMSE	APL	MAPE	RMSE	APL	MAPE	RMSE	APL
2012	1.68 (0.004)	7.95 (0.02)	2.09 (0.005)	1.63 (0.09)	7.84 (0.23)	2.04 (0.08)	1.74	8.10	2.15
2013	1.96 (0.028)	9.21 (0.13)	2.52 (0.04)	1.78 (0.03)	8.57 (0.10)	2.36 (0.03)	2.13	10.10	2.76
2014	1.76 (0.008)	7.73 (0.03)	2.19 (0.008)	1.78 (0.06)	7.82 (0.19)	2.22 (0.06)	2.00	8.70	2.49
2015	2.06 (0.006)	9.09 (0.03)	2.51 (0.007)	2.06 (0.09)	9.07 (0.30)	2.51 (0.11)	2.52	10.74	3.05
2016	1.66 (0.002)	6.97 (0.01)	1.95 (0.002)	1.62 (0.01)	6.94 (0.24)	1.95 (0.09)	1.70	7.58	2.07

Table 3.3. Results of epochwise and continual operation NAX. On the *left*, the results of the training in the case of the continually operating network trained with RTRL, at the *centre* the standard epochwise network with a sliding window of length 5, on the *right* the original results; ten repetitions of the training have been done, in brackets the standard deviations are indicated. It is possible to see that with a fine tuning of the hyperparameters NAX can perform even better.

as the first model and learning rate equal to 0.0005.

Following the usual NAX methodology, the two networks are trained on 5 different time windows and the resulting forecasts are analysed. In order to have an estimate of the robustness of results, 10 repetitions of each training are performed. Table 3.3 shows the forecast accuracy of the two configurations in terms of MAPE, RMSE and APL, which are compared to the original results provided in the original paper (cf. Azzone & Baviera 2021).

It can be noticed that in both case the original accuracy of the predictions is increased with respect to the original network and that the epochwise operating network with shuffling is the configuration that obtains the best performances; anyway also the continually operating network - which is trained using RTRL algorithm - can obtain relevant results, which are even more stable in terms of standard deviation. As a final result, we have proved how shuffling the sample and enlarging the short-term context can bring benefits for training the network.

This final comparison is the main result of the present chapter, since the performances of NAX are improved and stabilized. On the one hand, this fact is important to have more precise daily forecasts, on the other because this model will be used also in the *hourly* case in Chapter 5. Besides, this analysis has given us relevant insights about the sliding window methodology, the fine-tuning of some

hyperparameters (like the window length) and the impact of shuffling. In this sense, we have experimented how RNNs can capture pieces of information from the previous context and, instead, which are their limitations. Moreover, we have shown that, in this specific case, RTRL is a feasible algorithm that can be used to train the network. We have adopted it for training the continually running version of the NAX network, and the C++ implementation of this algorithm has proven to be faster than the epochwise sliding window approach implemented in `Keras` on a standard PC.

In summary, in this chapter we have introduced and discussed the main features that characterize the dynamics of daily consumption over the year, we have introduced NAX and shown its main strengths and we have discussed how to improve its performances. Moreover, we have introduced the RTRL algorithm, which will be used again in Chapter 5. In the following chapter, the main features of intra-daily load dynamics will be investigated.

Chapter 4

Analysis of Intra-daily Load Dynamics

Despite the fact that daily modelling can achieve a noticeable accuracy, for many applications it is necessary to have a more precise characterization of the intra-daily dynamics of power consumption. A good part of the recent literature is focused on the analysis of electrical load on an hourly basis, and some authors also consider quarter-hour data (cf. Guerini 2016 and references therein). On the one hand, the interest for this order of magnitude is due to the needs of the power industry: generators and utility companies require indeed detailed profiles in order to plan and set up a more correct production and distribution of the energy. On the other hand, the hour is also the fundamental unit of measure for many products which are traded in the energy market.

Although the problem of modelling intra-daily dynamics may seem very similar to a continuous-time framework, the techniques that are employed are based on statistical analysis, and linear models are still frequently used. This chapter introduces hourly modelling: the selected dataset is presented and used in order to explain the main features of intra-daily power consumption. Afterwards, two linear models are proposed and outlined, so as to have a better understanding of the typical methodologies that can be used for the purpose.

4.1 Data analysis

The analysis in Chapter 3 highlighted that daily demand is characterized by the superposition of two distinct effects, due to the (half-)yearly and weekly seasonality. It is easy to deduce that the main complexity associated with *hourly* forecasting is the occurrence of a third seasonal effect, the daily one. This latter has the straightforward implication that during the night energy demand is far less than the one registered during daytime because of the lack of human activity. But there

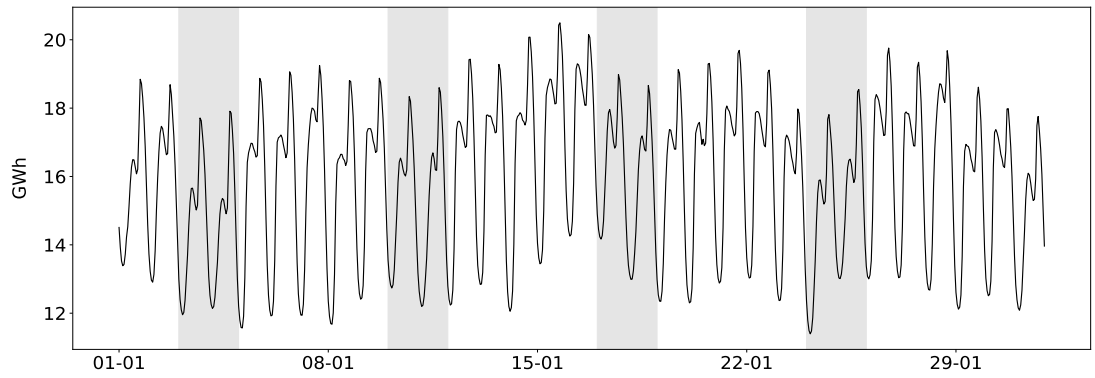


Figure 4.1. Hourly energy consumption in January 2009 for the whole New England. The plot shows the profiles of intra-daily load: during the cold months, it is characterised by this "M"-shaped behaviour with two peaks, one around midday and the other in the late afternoon. In the grey sections, Saturdays and Sundays: during weekends consumption is lower with respect to working days, as already underlined in the daily analysis.

is also to consider that the variation of temperature and humidity during the day has an impact on electrical consumption. For instance, during the hottest days of summer load peaks are expected to occur in the afternoon and in the early evening, due to the use of air-conditioners.

For all the following analysis, the considered data set is the same one used for the daily case, with the clear difference that now it is considered on an hourly basis, as it is originally provided by ISO-NE. Thus now the data are aggregated by considering the sum of the hourly demand on the 8 zones of New England and by computing the hourly average temperature of the entire region (cf. Section 3.1).

From the modelling perspective, the most critical issue is understanding how to manage the coexistence of the three hierarchical seasonal components. Figure 4.1 reports the dynamics of electricity demand during the month of January; as one can notice, it is characterised by a peculiar daily behaviour, with two peaks during daytime - the first around 11 in the morning and a second around 18 - which are followed by a sudden drop at night-time. Incidentally, in the following we will consider that hour 11 corresponds to the hour starting at 11:00 and ending at 12:00, hour 12 corresponds to 12:00-13:00 and so on. Moreover, it is evident that the impact of the weekend is still relevant and in detail, a decrease in the electrical demand is observed, mainly due to the fact that a large part of the businesses is closed.

However, the crucial aspect is that the daily profiles are not constant during the year, but - in particular during the summer months - they exhibit different shapes. Figure 4.2 shows four different profiles, one for each season, that reveal how different the intra-daily dynamics can be during the year: this is a very important point that has to be carefully considered when building an effective

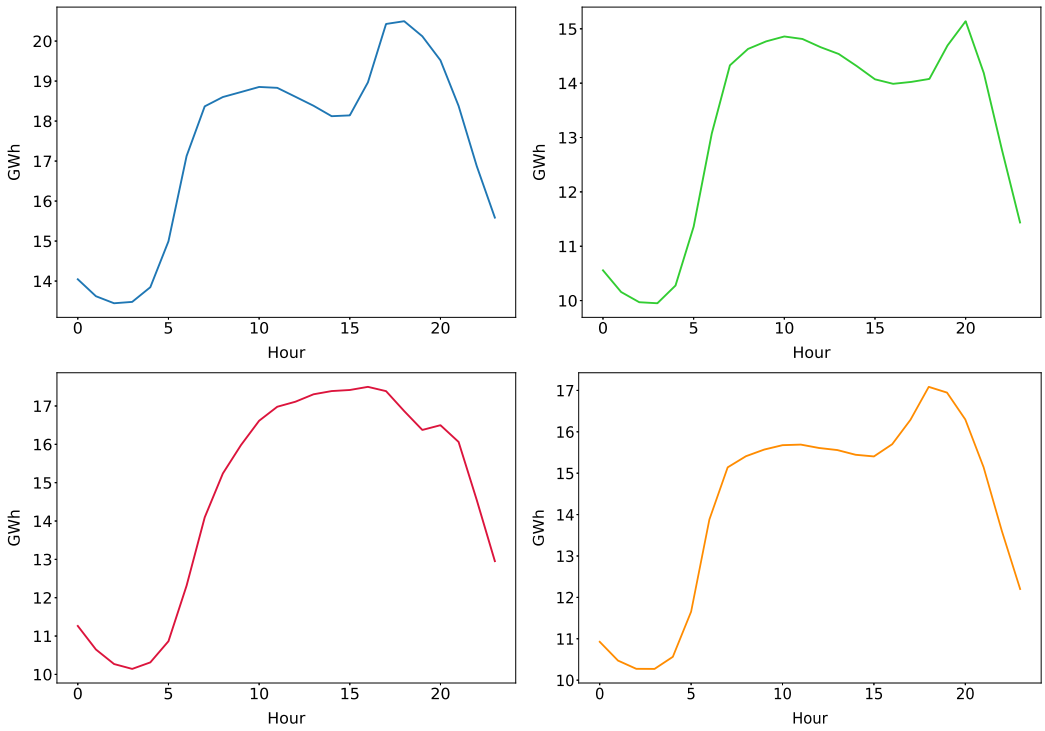


Figure 4.2. Intra-daily demand profiles in the four seasons in 2009: 15th of January (*above left*), 15th of April (*above right*), 15th of July (*below left*) and 15th of October (*below right*).

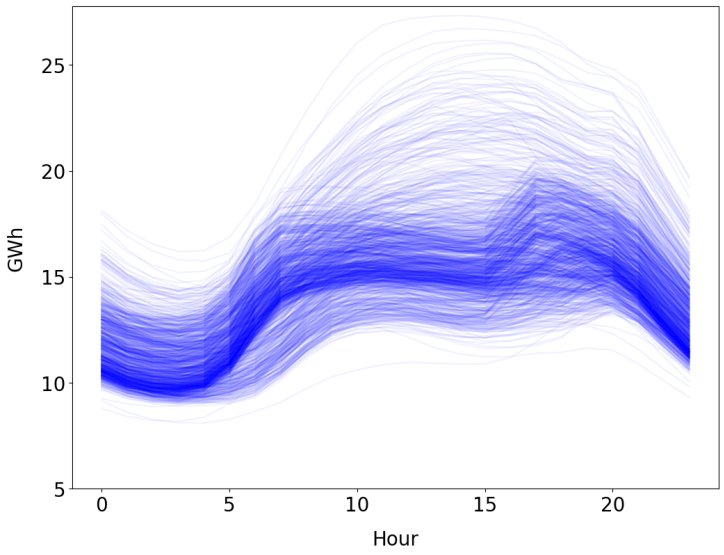


Figure 4.3. Spaghetti plot of the intra-daily patterns for years 2009-2011. There is strong evidence that the "M"-shaped profiles are the most typical over the year, but also the arc-shaped ones are quite common. Moreover, it is possible to see that these latter ones are usually associated with relevant peaks.

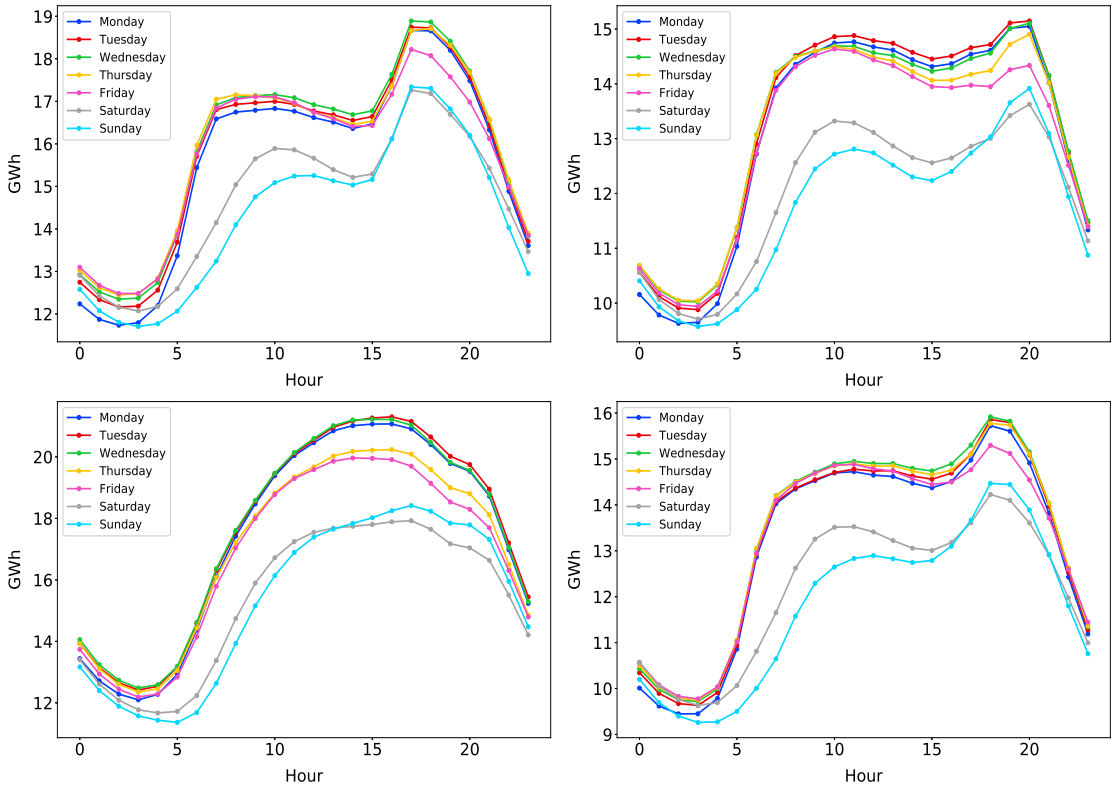


Figure 4.4. Intra-daily demand profiles for each day of the week: the plots refer to January (*above left*), April (*above right*), July (*below left*) and October (*below right*). For each day of the week, the average over the selected month is computed; moreover in order to limit the potential impact associated with the different weather conditions, the data for the time-window 2009-2015 are used. It can be noticed that the overall shapes of the demand are very similar to the ones in Figure 4.2.

model. Indeed it is clear that such behaviour is caused by the *interaction*, i.e. the combined effect, of the yearly and the daily periodicity. As highlighted in Figure 4.3, the most common pattern is the "M"-shape, which typically is characteristic of the cold months, but it is present also during the mid-seasons; instead, the summer months show a remarkable increase in the demand during the central hours of the day, in all probability determined by the massive use of air conditioners.

Anyway, not only the intra-daily load varies depending on the period of the year but also depending on the day of the week. In Figure 4.4 are depicted the average profiles for the seven days of the week - again - in four different months of the year. At first sight, it is evident that the electrical consumption is not constant during the week, and two main categories can be identified, weekdays and weekends; this is completely analogous to what has been done for the daily modelling with NAX. Nevertheless, even within these categories, there are some slight differences depending on the considered day. For instance, Fridays are usually characterized by a lower demand during the afternoon, as a consequence of the end of the working week; a similar behaviour is found also on the Thursdays of July. The first hours of

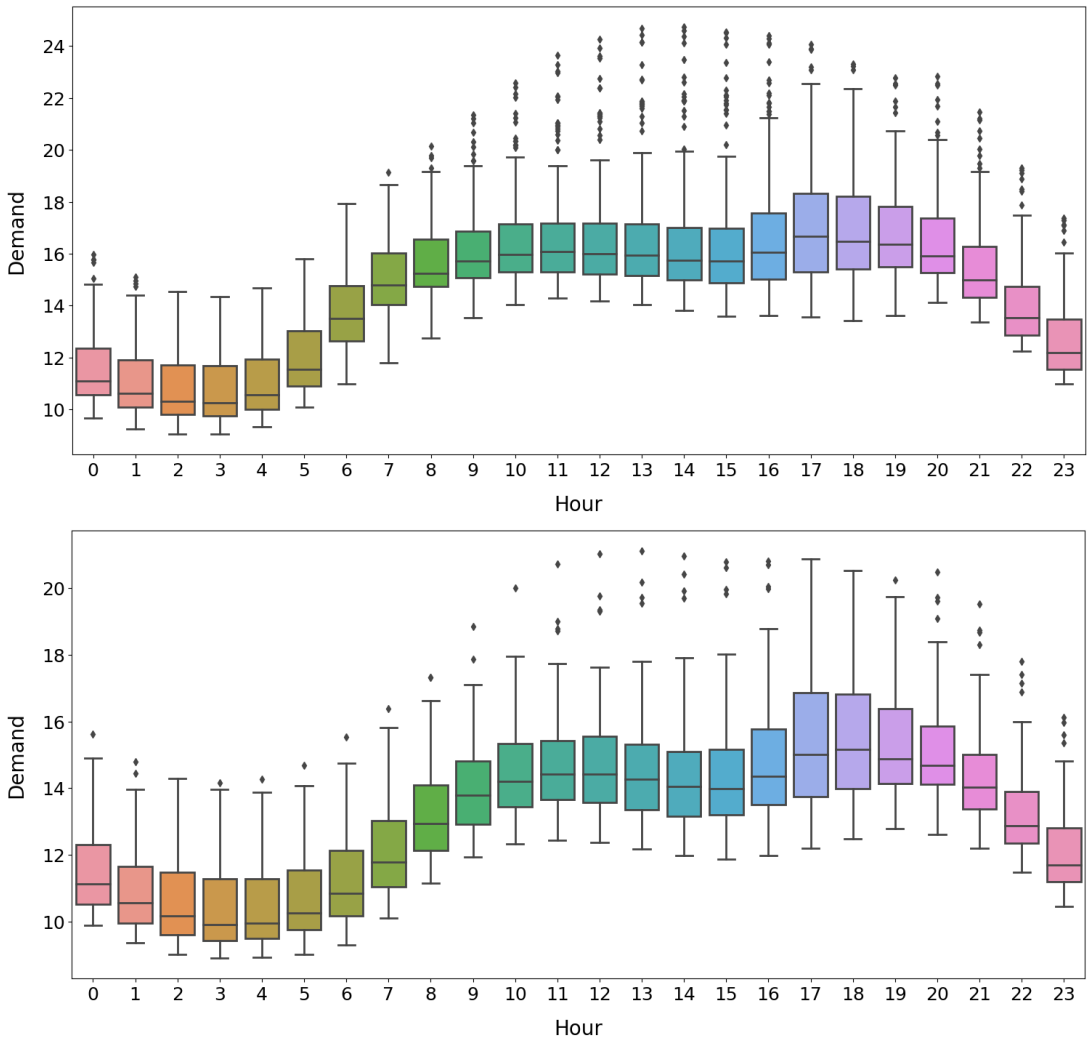


Figure 4.5. Boxplots of hourly demand in 2009 for weekdays (*above*) and weekends (*below*). The two graphical tools show that there are slight differences in the dynamics of intra-daily demand, that are associated in particular to the central hours of the day.

Monday, instead, exhibit a behaviour which is more similar to the ones of Sunday; and on the contrary, the first hours of Saturday are more similar to the ones of the working day.

In addition, it should be noted that the load during Saturdays and Sundays has not exactly the same shape as the weekdays. Indeed for example during the morning the growth of demand is much slower since people usually wake up later. In this regard, further details are provided Figure 4.5: the boxplots show that, although the night patterns of the two categories - weekdays and weekends - are similar, there are some differences during the working hours, i.e. 8:00-17:00. In general, they are characterised by greater variability and fluctuations during the weekend, while they are more regular during the weekdays; moreover, it is possible to notice also a greater increase of load around 18:00 and the already mentioned

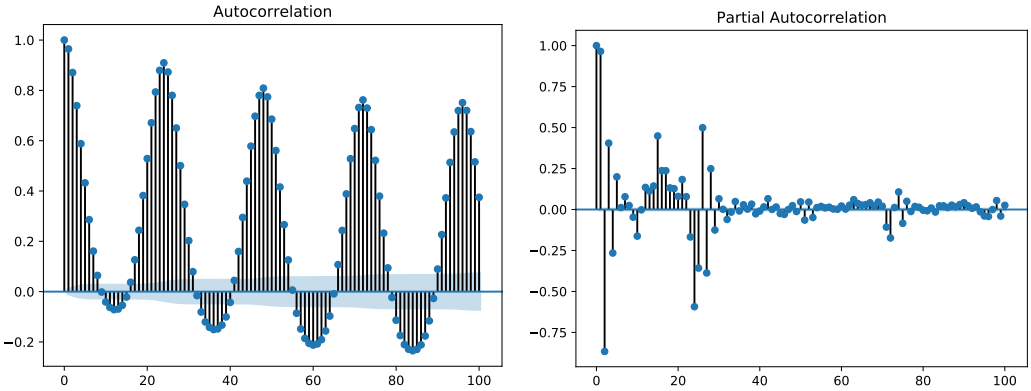


Figure 4.6. Autocorrelation structure of raw hourly data for the year 2009.

delayed growth in the morning.

Of course, as a consequence of all the previous discussions about the relevant seasonal effects, we expect the time series of hourly load to exhibit a strong serial correlation; the corresponding autocorrelation functions are plotted in Figure 4.6. In detail they show that in order to obtain efficient modelling, data must be processed in order to remove as much as possible the short-term dependencies; PACF plot reveals that there is a complex structure of serial correlation in the raw data that involves the previous day and that should somehow be exploited. In this regard, whilst in the daily case the sinusoidal seasonal trend was extremely evident, in this case it is not straightforward to understand how to remove the daily periodicity, mainly because it varies over time. This is a very important point for the construction of appropriate models and, in the following pages, different techniques will be considered for this purpose.

As a final note, when dealing with hourly data one has always to check the problem of *daylight saving time*. In general, the different dataset might treat it differently; in the present case, data are provided in such a way that at 2:00 of the transition day in November the demand is aggregated - being thus the equivalent of the consumption of two hours - while in March a null value is set in correspondence of the transition hour. Neglecting this fact does not produce serious issues when working with daily data (in principle the error can be quantified as 1/24, but it is even lower since demand at night-time is very limited), but has certainly a greater impact in the case of hourly modelling since relevant outliers are introduced.

4.2 Frequency-based approach

4.2.1 Fourier analysis of seasonality

A first approach that has to be considered for a better understanding of consumption dynamics is based on *Fourier analysis*. The idea is to use the typical tools of signal processing to identify the relevant periodic components that are present in the time series; the *Fourier Transform* is indeed known for converting a discrete signal from its original domain into the frequency domain. In detail, given a sequence of complex (or just real, in the case of time-series) numbers $x_0, x_1, \ldots, x_{n-1}$, the *Discrete Fourier Transform* (DFT) is defined as

$$d(\omega_j) = \sum_{t=0}^{n-1} x_t \, e^{-2\pi i \omega_j t} \qquad j = 0, \ldots, n-1$$

with $\omega_j = j/n$. Since each $d(\omega_j)$ is a complex number, it can be represented as

$$d(\omega_j) = R(\omega_j) e^{i\Phi(\omega_j)}$$

where $R(\omega_j)$ is the magnitude, a value associated to the intensity of the oscillation with frequency ω_j, and $\Phi(\omega_j)$ is the phase.

The values

$$I(\omega_j) = \frac{1}{n} R(\omega_j)^2 = \frac{1}{n} |d(\omega_j)|^2$$

form the so-called *periodogram*, which is an estimate of power spectral density of the signal (cf. Bloomfield 2013); in short, the well-known periodogram allows to identify which are the main frequencies that compose a time-series.

The periodogram of log-demand is reported in Figure 4.7 and shows the presence of periodic behaviours on three levels, as expected. In particular, we may choose to consider 3 harmonic frequencies for the Fourier expansion of the yearly seasonality, 3 for the weekly seasonality and 2 for the daily one. Thus the core dynamics of the time-series can be described as

$$\begin{aligned} Y(t) = \beta_0 \; + \; \beta_1 t \; + \; \beta_2 D_{Hol}(t) \; + \\ + \sum_{k=1}^{3} \left[\beta_{2k+1} \cos(k\Omega t) + \beta_{2k+2} \sin(k\Omega t) \right] \; + \\ + \sum_{k=1}^{3} \left[\beta_{2k+7} \cos(k\Psi t) + \beta_{2k+8} \sin(k\Psi t) \right] \; + \\ + \sum_{k=1}^{2} \left[\beta_{2k+13} \cos(k\Theta t) + \beta_{2k+14} \sin(k\Theta t) \right] \end{aligned} \tag{4.1}$$

where, assuming that t is measured in hours

$$\Omega = \frac{1}{365.25 \cdot 24} \qquad \Psi = \frac{1}{7 \cdot 24} \qquad \Theta = \frac{1}{24}$$

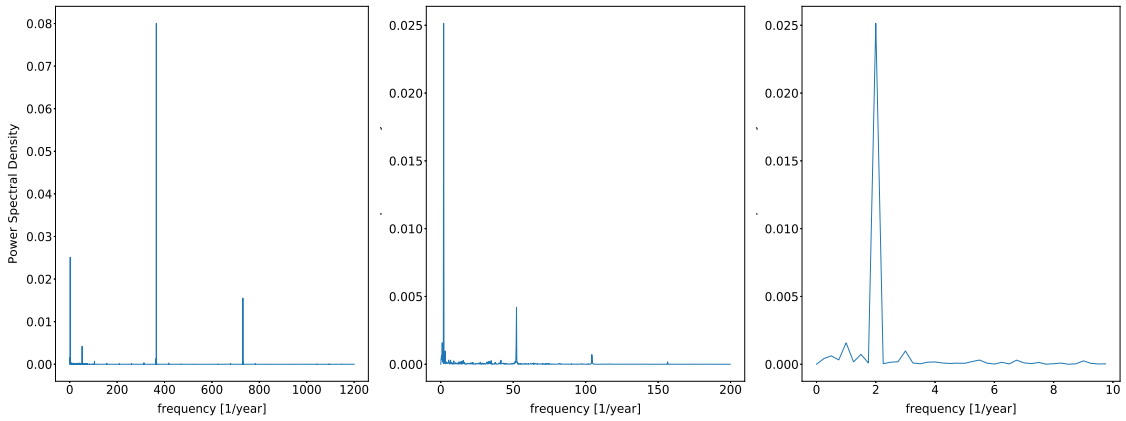


Figure 4.7. Periodogram of the time-series (2009-2012) on different scales. The reported frequencies are to be thought on a yearly basis, so that the most relevant component is the daily one, having frequency 365 (*left*). The three plots are meant to be representative of the different periodicities: the daily dynamics (*left*), where two harmonics are found to be significant, the weekly one (*centre*) where two or three harmonics can be selected and the yearly one, characterized by three relevant frequencies (*right*). In particular, this latter plot highlights that the half-yearly component is more important than the yearly one.

This Fourier-based approach can be a valid choice for reproducing the so-called *potential* of the time-series, i.e. the ensemble of its seasonal patterns (cf. Guerini 2016); in addition, trend and special events should be modelled appropriately - for instance in equation (4.1) holidays are considered by means of a specific dummy variable. The overall result is a linear model that can be calibrated through OLS.

Figure 4.8 reports some results of the fit on the 2009-2012 data: it is evident that such a model is not able to reproduce properly the intra-daily profile. In particular in the case of January, the "M"-shape of the profile is quite correct; instead, there are evident issues in replicating the consumption of July, or in general of the warm months. Intuitively, the model is not flexible enough to describe properly these two different kinds of profile, and so it chooses to fit the "M"-shaped one because it corresponds to the predominant behaviour over the year.

Nevertheless, the periodogram proves to be an effective tool for capturing the most relevant periodic features of the time series, since in practice all the selected harmonics are found to be statistically significant in the regression.

4.2.2 The interaction effect

What is for sure missing in the just presented model is the *interaction* between variables: the model is just additive and this is not enough to explain a daily profile that varies according to the season. A way to take into account this aspect is to

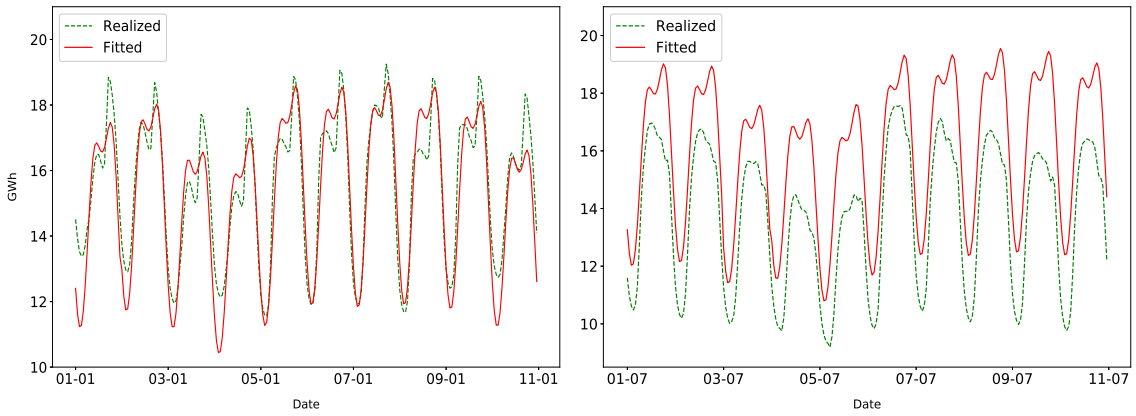


Figure 4.8. Fit of Fourier model on 2009-2012 data. In detail the first ten days of January 2009 (*left*) and of July 2009 (*right*) are plotted. The critical difficulty for the model is to reproduce the shape of the intra-daily consumption, as it can be noticed in particular for the July case.

extend the number of regressors that describe the potential as follows: let

$$\begin{aligned} \mathcal{Q} &= \{\cos(k\Omega t),\ k \in [0, \mathrm{N}_q]\} \ \cup\ \{\sin(k\Omega t),\ k \in [1, \mathrm{N}_q]\} \\ \mathcal{W} &= \{\cos(k\Psi t),\ k \in [0, \mathrm{N}_w]\} \cup\ \{\sin(k\Psi t),\ k \in [1, \mathrm{N}_w]\} \\ \mathcal{D} &= \{\cos(k\Theta t),\ k \in [0, \mathrm{N}_d]\} \ \cup\ \{\sin(k\Theta t),\ k \in [1, \mathrm{N}_d]\} \end{aligned} \tag{4.2}$$

Then it is possible to select the set of the covariates as the tensor product of these sets, $\mathcal{Q} \otimes \mathcal{W} \otimes \mathcal{D}$, i.e. to consider all the regressors of the form

$$h_{ijm}(t) = \sin(i\Omega t)\cos(j\Psi t)\cos(m\Theta t)$$

or

$$h_{ijm}(t) = \sin(i\Omega t)\sin(j\Psi t)\sin(m\Theta t)$$

In other words, each regressor is a combination of three sinusoidal functions with suitable frequencies. In addition, it should be noticed that the defined space contains not only all the triplets of multiplied sinusoids, but also all the singletons and the pairs, since in equations (4.2) the index k is allowed to hold 0 for cosines.

This approach, described e.g. in Guerini (2016), leads to a better description of the phenomenon (see Figure 4.9); intra-daily profiles are in this case far more appropriate and seem to follow convincingly the observed dynamics. The drawback is that a huge number of covariates is involved and very few of the ones that are formed by the multiplication of three terms are found to be significant in the regression.

For instance, the depicted model is the result of the interaction of 3 yearly, 3 weekly and 2 daily harmonics - it is thus analogous to the model developed in Subsection 4.2.1 - and in principle it includes 244 regressors (16 singletons, 84 pairs of sinusoidal functions multiplied together and 144 triplets) plus the intercept, the

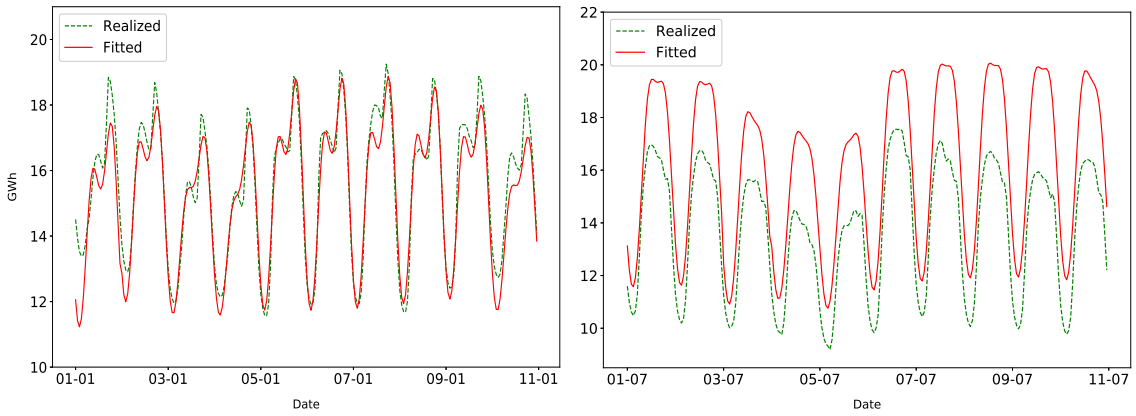


Figure 4.9. Fit of Fourier model with interactions (N_q=N_w=3 and N_d=2) on 2009-2012 data. The first ten days of January 2009 (*left*) and of July 2009 (*right*) are plotted. The presence of the interaction terms allows the daily profile to assume different shapes throughout the year.

linear trend and the holiday dummy variables. Among this multitude of terms, just 8 of the 144 triplets are found to be *individually* significant at level 5%, i.e. the p-value of the associated t-tests is almost always greater than this threshold. Of course, this does not mean that they are completely useless and can be removed all at once, but that actually the model could be somehow refined by reducing the number of covariates with an iterative procedure (for instance the *backward elimination* technique).

In this framework, the best model can be chosen by means of information criteria, the Akaike Information Criterion (AIC) and the Bayes Information Criterion (BIC), or according to the Adjusted R-squared. In particular BIC might be the most appropriate one, since it is prone to privilege more parsimonious model and thus it can help in limiting the number of regressors.

For the present case, the results of the model selection are provided in Table 4.1: under the constraint that no more than three harmonics can be considered for each seasonal pattern, AIC and BIC are used to define the best configuration. AIC leads to the selection of the model with the maximum available complexity, i.e. N_q=N_w=N_y=3; this however may not be the best practical choice because of the large number of mixed interaction terms that are considered in the linear regression.

BIC as expected selects instead a simpler model, which actually is characterised by N_q=N_w=3 and N_d=2. According to this criterion, the increase of the model size caused by considering the third daily harmonic is not followed by a relevant improvement of the performances; in this sense, the benefits are not enough to allow the introduction of all those regressors. An interesting thing is that the triplet (N_q,N_w,N_y)=(3,3,2) is the same one that was identified by the periodogram; this entails that the relevance of these frequencies is not only associated with their interaction, but they are also individually very important to understand the

N_q	N_w	N_d	N_{tot}	R^2	AIC
3	3	3	345	0.872	-85228.2
3	3	2	247	0.870	-84877.7
3	2	3	247	0.869	-84660.2

N_q	N_w	N_d	N_{tot}	R^2	BIC
3	3	2	247	0.870	-82786.9
3	2	2	177	0.867	-82729.6
3	2	3	247	0.869	-82569.4

Table 4.1. Results of model selection on 2009-2012 data by means of information criteria. The *left* table displays the best three models according to AIC, the *right* one according to BIC. In addition to the value of the statistics and number of harmonics N_q, N_w and N_d, also the total number of regressors N_{tot} and the R^2 of the regression are reported.

periodical structure of the time-series.

4.2.3 Incorporating the temperature

The Fourier model with interactions is anyway a weather-independent model and, as a consequence, it is not capable of fitting perfectly the provided data: there are indeed some features of energy load that cannot be explained with the seasonal analysis only. This fact can be observed in Figure 4.9, as the July panel shows a significant discrepancy between the fitted values and the observed consumption: the shape of the former is reasonably correct, but it is shifted upwards. During the warm months, energy demand is extremely sensitive to weather conditions and uncommon dynamics can occur when humidity or temperature is higher or lower than usual.

In order to obtain a better fit, it is possible to enrich the previous model by including the available climate variables among the regressors: in this case we may choose to consider the first three powers of the (dry-bulb) temperature. This choice is motivated by the behaviour of log-consumption as a function of temperature, which can be reasonably described by a cubic function:

$$Y = \alpha_0 + \alpha_1 T + \alpha_2 T^2 + \alpha_3 T^3$$

as shown in Figure 4.10.

We might expect that not all the informative content of these weather variables is completely new; after all it is characterised by a yearly and daily seasonality which should somehow resemble the one of energy load. However the statistical impact of the three powers of temperature is extremely noticeable and helps substantially in explaining the residual variability. As usual, the two profiles of the first ten days of January and July are presented in Figure 4.11: the same interaction structure of Subsection 4.2.2 (3 yearly, 3 weekly and 2 daily harmonics) is considered, and the

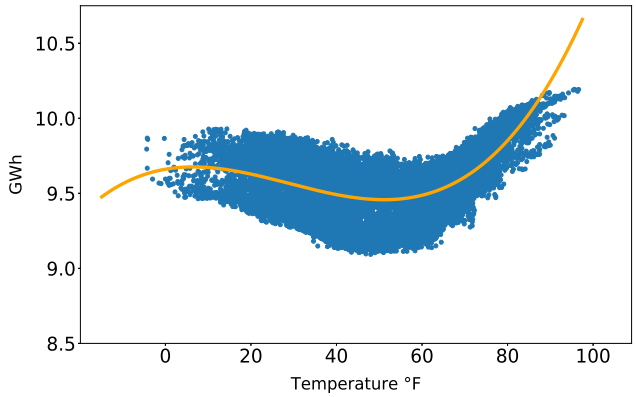


Figure 4.10. Log-consumption as a cubic function of temperature. A linear regression on normalised variables is performed (because of the different orders of magnitude involved), which are then converted back to their original scale. The obtained cubic is $Y = 9.66 + 4.94 \cdot 10^{-3}\, T + 4.26 \cdot 10^{-4}\, T^2 + 4.93 \cdot 10^{-6}\, T^3$. As usual, the time period 2009-2012 is considered for the analysis.

first three powers of dry-bulb temperature are added as predictors. Moreover, to have a quantitative measure of the different performances in terms of *fitting* the data, in Table 4.2 are reported R^2, RMSE and MAPE for the three models presented in this section.

In conclusion, Fourier analysis proves to be an effective technique to capture the core dynamics of this time series. The final model is shown to be a significantly improved version of the previous ones and is able to fit the data with reasonable accuracy, with a MAPE that is slightly less than 3.5%; instead, its forecasting performances will be discussed later on. For simplicity, in the following, we will refer to this benchmark model as the *extended Fourier* model.

interaction	weather	R^2	RMSE	MAPE
✓	✓	0.948	697.13	3.38%
✓		0.870	1137.04	5.38%
		0.831	1280.01	6.28%

Table 4.2. Summary of the results for the three frequency-based models on 2009-2012 data. Three measures for the goodness of fit, i.e. R^2, RMSE (in MWh) and MAPE, are considered in order to evaluate the differences among them.

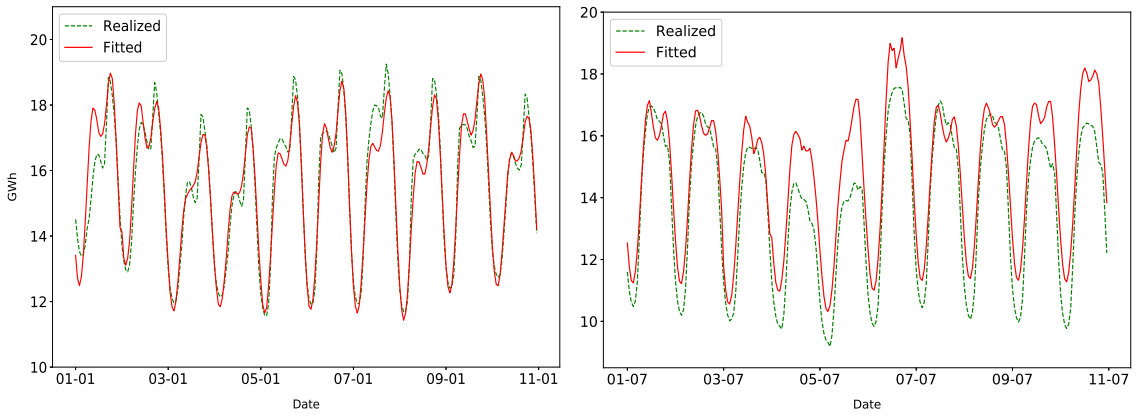


Figure 4.11. Fit of Fourier model with interactions and temperature effect on 2009-2012 data. As before, the first ten days of January 2009 (*left*) and of July 2009 (*right*) are plotted. The accuracy of the fit is noticeable increased with respect to the analogous plots presented in the previous pages.

4.3 Tao Vanilla Model

In recent times, the so-called *Tao* model (also known as the *Tao Vanilla* model) has become fairly popular in literature. It is a linear model that was introduced by Tao Hong in his dissertation (cf. Hong 2010) and mainly owes its fame to the fact that it has been used as a benchmark in the GEFComs.

Contrarily to the extended Fourier model, which is based on a sinusoidal encoding of the seasonality, the Tao model adopts a *one-hot* encoding scheme, i.e. every categorical variable is considered in the model by means of a specific dummy variable. For instance, 24 different binary variables are generated to encode the information about the hour of the day: they thus form a set of variables $\{H_i\}_{i=0}^{23}$ such that for a given hour h

$$H_i(h) = \begin{cases} 1 & \text{if } h = i \\ 0 & \text{if } h \neq i \end{cases}$$

In the Tao model the hour of the day, the day of the week and the month of the year are encoded in this way, generating three sets of dummy variables which are respectively denoted as $\{H_i\}_{i=0}^{23}$, $\{W_i\}_{i=1}^{7}$ and $\{M_i\}_{i=1}^{12}$. In this way each seasonal block is decomposed into the unit of the highest resolution.

According to its definition, the Tao model considers the demand and not the log-demand as the dependent variable; for consistency with the notation used so far, we indicate it as $\exp Y(t)$. In a synthetic manner the model is expressed as (cf. Hong, Xie et al. 2019):

$$\exp Y(t) = \beta_0 + \beta_1 t + \beta_2 M(t) + \beta_3 W(t) + \beta_4 H(t) + \beta_5 W(t)H(t) + f\big(T(t)\big)$$

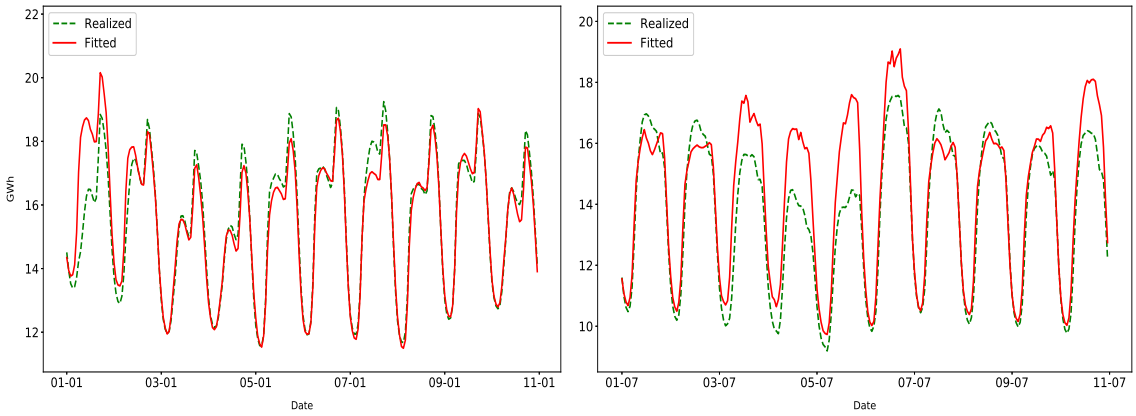


Figure 4.12. Fit of Tao model on 2009-2012 data. The overall results are similar to the ones obtained with the extended Fourier model.

Instead each element that involves one of the dummy variables should be thought as follows: for instance

$$\beta_2 M(t) = \sum_{i=1}^{12} \beta_2^{(i)} M_i(t)$$

In this sense there are 12 different values of β_2, that can be denoted as the set $\{\beta_2^{(i)}\}$, each one accounting for the contribution of each month; while $M(t)$ represents the only dummy variable that holds 1 at time t. This entails that the term $W(t)H(t)$ represents the notion of *hour of the week*, expressed by means of 168 dummy variables, and the associated coefficient β_5 is thought to assume 168 different values $\{\beta_5^{(i)}\}$. Lastly, the function f accounts for the weather-dependent term of the model and is defined as

$$\begin{aligned} f\big(T(t)\big) = \beta_6 T(t) + \beta_7 T^2(t) + \beta_8 T^3(t) + \beta_9 T(t)M(t) + \beta_{10} T^2(t)M(t) + \\ + \beta_{11} T^3(t)M(t) + \beta_{12} T(t)H(t) + \beta_{13} T^2(t)H(t) + \beta_{14} T^3(t)H(t) \end{aligned}$$

Essentially, the Tao model creates a network of connections between the variables. For instance, when defining the consumption on a Monday of January at 8:00, one has to consider that the parameter:

- ▷ $\beta_2^{(1)}$ is the same for all the other days of January
- ▷ $\beta_3^{(1)}$ is the same for all the other Mondays of the year
- ▷ $\beta_4^{(8)}$ is the same for all the other days of the year at 8:00
- ▷ $\beta_5^{(8)}$ is the same for all the other Mondays of the year at 8:00

and so forth. As a final note, holidays are not considered explicitly in this model.

R^2	RMSE	MAPE
0.953	616.00	2.77%

Table 4.3. Results of fit for Tao model on 2009-2012 data. As before, three measures for the goodness of fit are considered, i.e. R^2, RMSE (in MWh) and MAPE.

With reference to and Table 4.3, the Tao model seems to perform better than extended Fourier in terms of fitting the data, also considering that they have more or less the same complexity.

However, both models have been shown to be effective in capturing the main features that characterize the energy demand. Up to now, we have focused on the aspect of *fitting* the model, i.e. we have considered how these models are appropriate to describe the data. In the next chapter, we will instead consider intra-daily forecasting, and we will compare these two models with two other models which are based on the use of RNNs.

Chapter 5

Hourly Forecasting with RNNs

Linear regression models have always played a central role in load forecasting since the inception of the electric power industry. They are extremely used because of their versatility, their robustness and the fact that they can be easily calibrated. However, as already discussed in this thesis, the Artificial Intelligence techniques - and above all Neural Networks - are emerging, showing promises in much better predictions as compared to traditional methods. Among the main features that characterize hourly load there are the presence of nonlinear behaviours and a significant serial correlation: therefore obtaining accurate predictions is a very challenging goal and more complex models have been introduced to properly capture and exploit at best this kind of information. This chapter is focused on the development of two new models for electric load forecasting on an hourly basis, which make use of Recurrent Neural Networks and are partially inspired by the previously discussed NAX model. After the description of the two architectures, their forecast accuracy is analysed and compared to the one of the two benchmark models of Chapter 4.

5.1 Introduction

The main reason why extended Fourier and Tao models have been presented and described is to show two possible standard methodologies for modelling the hourly demand using linear regression. In particular, this has allowed us to understand which are the main features of the hourly load dynamics and how the two models are designed in order to tackle the problem. In this regard, they are both meant to exploit as well as possible the multi-seasonality of the time series, but they do it in different ways.

The extended Fourier model has a relevant and nice theoretical justification, but, in real life, it is very difficult to think of energy consumption as a superposition of harmonic frequencies. Moreover, the introduction of the interaction effects not

only leads to the creation of an almost uncontrollable number of terms, but it makes it even harder to understand the practical meaning of each regressor; in this sense, the model seems to be even more abstract. Indeed the great thing about linear regression models is that they allow us to comprehend the role of every covariate in the observed phenomenon, something that in this case is instead impossible.

On the other hand, Tao's approach seems to be extremely naive; in practice, many different models are introduced, each one in charge of describing a small part of the overall phenomenon. However, the fact that all these smaller models are not independent, but they somehow share some parameters, allows in part the reproduction of the complex seasonal behaviour of the energy demand. Also thanks to a better modelling of weather effect, the Tao model is able to achieve better results, and in addition, it has also a clearer interpretation of the meaning of each regressor, being almost all of them dummy variables.

Nevertheless, none of the two is originally conceived with the specific aim of providing probabilistic forecast, but rather point predictions. Our goal is instead constructing a solid model that is capable of forecasting in an appropriate manner the distribution of the predicted load, taking into account the associated uncertainty. Moreover, we are convinced that employing for the purpose a nonlinear architecture is fundamental in order to suitably capture all the details of the observed dynamics. The main focus is indeed on the application of Recurrent Neural Networks, in the light of the convincing results that the NAX model achieved in the daily case.

However, it is important to mention that the transition from daily modelling to hourly modelling has a critical implication, namely the huge increase in the size of the sample that has to be studied. This is often a central point that has to be carefully considered when creating and designing sophisticated models. Calibrating a model - especially a RNN - on a training set that is too large may represent a problem from the computational perspective. In this regard, one of the strengths of the NAX model is the fact that it can be calibrated on few years (three) of daily data, which is a great compromise between the complexity of the model and the size of the data-set used for training; we would like our model to be able to work on a similar time-range, without sacrificing its accuracy.

Hence, for the following discussion, we consider just a part of the original dataset made available for the GEFCom 2017, that covers the time window 2009-

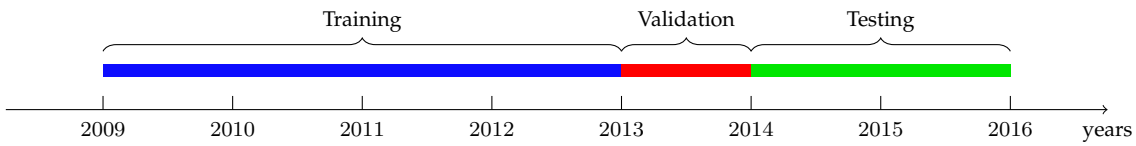


Figure 5.1. The timeline defines how the selected dataset is subdivided and utilised for performing training, validation and testing of the model.

2015. In detail, it will be divided as follows: years 2009-2012 for training the model, year 2013 for the validation thereof, years 2014 and 2015 for testing (in particular 2015 is used to test the robustness of the proposed models).

5.2 H-NAX

The first model we introduce conjugates in some sense the sinusoidal approach of the Fourier expansion and the hourly decomposition of Tao. It is called H-NAX, which stands for *Hour-by-hour-NAX*. The idea behind its functioning is very simple: 24 different models are used to describe separately the consumption for each hour of the day. In detail, the NAX model is utilised for every hour, so that H-NAX is actually made of 24 different NAX models. Incidentally, this explains in part why in Chapter 3 this latter model is analysed in detail and special attention is paid to improve and stabilize its performances.

Figure 5.2 is very important to understand why the modelling structure proposed in H-NAX is reasonable, as it shows that the yearly behaviour of the consumption is almost the same for all the 24 hours of the day. Indeed the 24 coloured 'ellipses', each one representing the energy load at a certain hour, do have more or less the same shape and the same kind of deformation during summer. In this sense, the consumption at 4:00 will obviously be lower than the one at 16:00 - and

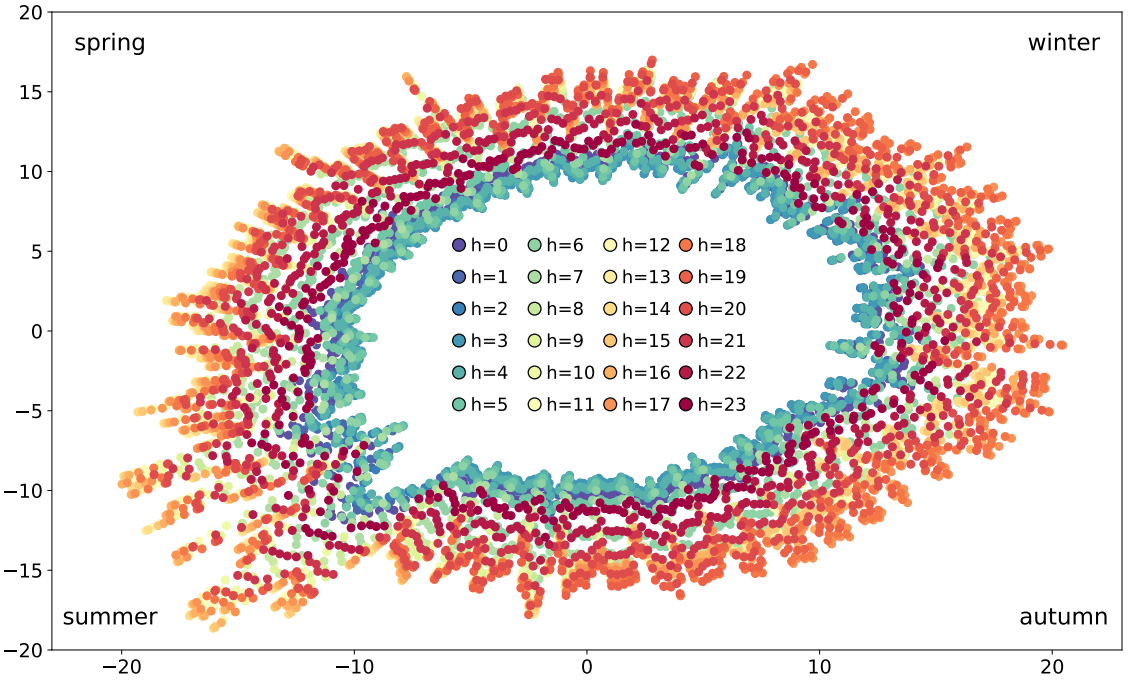


Figure 5.2. Polar representation of energy consumption in GWh during 2009. Instead of using the log-transformed data, the original ones are here considered so as to obtain a greater dispersion (Figure adapted from Ziel 2019).

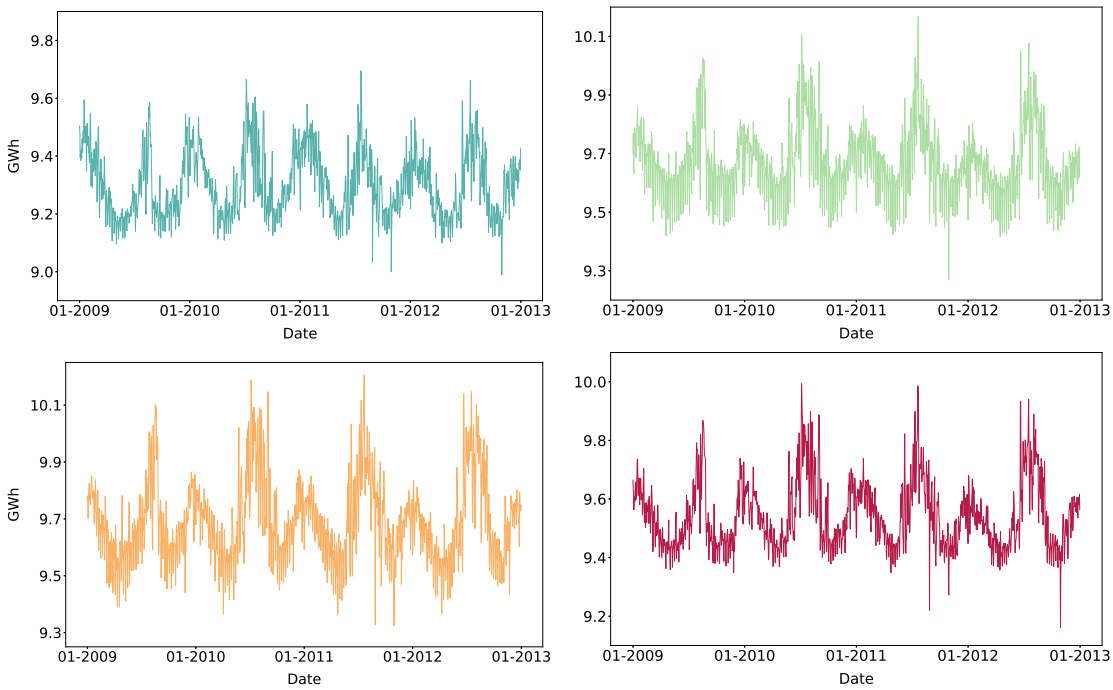


Figure 5.3. Yearly behaviour of energy consumption at a fixed hour for the period 2009-2012. The four profiles correspond respectively to 4:00 (*above left*), 10:00 (*above right*), 16:00 (*below left*) and 22:00 (*below right*); colours are thought to resemble the ones of Figure 5.2. It is possible to notice that the impact of heat pumps and air conditioners - the main causes of the increase of demand during the cold and the warm months - is substantial not only during the day but also at night-time; this leads to the creation of the half-yearly seasonality. In addition, it can be noticed that during night hours the yearly dynamics seems to be more balanced and the distinction between summer and winter is less evident.

indeed the ellipse associated with the former is smaller than the one associated with the latter - but they are both expected to exhibit the same seasonal patterns. In confirmation of this, in Figure 5.3 are depicted four consumption profiles that correspond to different hours of the day and they all reveal the peculiar half-yearly seasonality that was discussed for the daily case in Chapter 3.

From the theoretical perspective, the main defect of H-NAX is the inability to take into account intra-daily dependencies. The short-term context, i.e. what happened in the hours immediately preceding the considered one, is a piece of information that cannot be captured by the model. On the other hand, H-NAX is capable of creating long-term connections (the ones between the same hour in consecutive days) like NAX does for the consecutive days: this is of course a positive aspect that should in principle enhance the strength of the proposed model since there is a great correlation between the energy demand on a certain day at a certain hour and the one observed 24 hours before.

In addition, it is important to say that modelling separately the different hours

of the day is a fairly common approach in the literature; for instance it is reported that one of the first electricity forecasting competition was won by a team that included Engle and Granger (cf. Ramanathan et al. 1997) which proposed to use separate regression models for each of the 24 hours of the day. It is somehow reasonable to imagine that modelling separately the different hours is convenient because it allows the creation of tailor-made models - that for example can take into account the different impact that weather has on electricity demand depending on the hour of the day - and moreover, it reduces the computational complexity since the dataset is split into 24 independent parts. Of course, the price to pay when using this approach is the loss of one of the three seasonality, the daily one, as in practice every hour is modelled ignoring the existence of the other 23.

5.3 D-NAX

In contrast with the simplicity of H-NAX, the second model that we would like to present is characterized by a more challenging approach. Many applications of Deep Learning, above all the ones related to image recognition, make use of very deep networks in which every layer is in charge of studying a particular feature of the input (as a part of a process that is called *feature extraction*). Inspired by this idea, we consider a modified version of the RNN used in the NAX model which has no longer a single hidden layer, but two: the resulting model is thus called D-NAX, which is a contraction for *Double-NAX*. D-NAX is built so as to study the entire time-series and exploit the multi-seasonality that characterizes the hourly electrical load.

5.3.1 The model

In general, the procedure utilised by D-NAX is not so different from the original one of NAX. First of all, the time-series of the demand for the selected time interval is transformed into log-demand, in order to obtain a more compact and regular distribution. In this regard, Figure 5.3 shows that the during the daytime summer peaks are relevant and the impact of the logarithm can help to obtain a more uniform sample.

Modelling the *potential* (cf. Section 4.2.1) and the *interaction effects* is not a straightforward task, as mentioned in Section 4.1, because of the fact that it is not clear what the 'typical' daily seasonality is supposed to be throughout the year. One may choose to describe separately the daily profile of every month, but this may result inappropriate for instance in the case of a late spring or an early autumn. For this reason, we select two possible strategies: a frequency-based modelling with interaction, like the one of the extended Fourier model, and an approach similar to

the one of H-NAX, i.e. associated with the creation of hourly slices of the dataset.

In this latter case, the behaviour of the log-demand at each hour is modelled according to equation (3.2) and thus 24 different linear models are obtained. Instead, for what concerns the former option, we consider the configuration defined by the triplet of parameters (N_q, N_w, N_d)=(3,3,2), which is the one that is found to be optimal in Subsection 4.2.2.

The hour-by-hour approach is shown to be more effective in terms of the residual sum of squares, as indicated in Table 5.1, being is also the most parsimonious model of the two; it is therefore selected for the removal of the seasonality. Because of the similarity with the NAX model, we adopt the same notation used in Chapter 3 and write the reference model in the form of equation (3.1), here reported for convenience:

$$Y_t = \text{Trend}_t + S_t + r_t$$

It is important to remark that now the terms Trend_t and S_t depend on the hour of the day, because each one of the 24 linear model is characterized by a different vector of coefficients $\{\underline{\beta}^{(h)}\}_{h=0}^{23}$.

Of course, since we are going to model the residuals r_t of this regression, it is also important to ensure that the removal of this component does not alter too much or destroy their autocorrelation. The plots in Figure 5.4 show the structure of the serial dependency of the residuals after the detrending and the deseasonalization and can be compared to the initial plots in Figure 4.6. The PACF diagram indicates the presence of a short-term autocorrelation that involves mainly the first 2 lags; but more in general the residual at a given hour is correlated with the ones of the entire day before and, especially, with the ones of the days before during the same period of the day. In this regard, by construction each GLM creates a link between hours on a regular daily basis; this explains why the autocorrelation at 24-lag is lower in magnitude than the one at 25 and 26. Furthermore, because of the relevant 1-lag dependency, Augmented Dickey-Fuller test is performed: the corresponding p-value is found to be negligible and the null hypothesis of the presence of a unit root is rejected.

	Hour-by-Hour	Frequency-based
RSS	116.84	179.88
N_{tot}	216	247

Table 5.1. Comparison of the methods for identifying the potential in D-NAX. The two alternatives are evaluated in terms of Residual Sum of Squares (in MWh) and total number of parameters; the hour-by-hour approach is found to explain a greater part of the variability. Both models are calibrated on the 2009-2012 data.

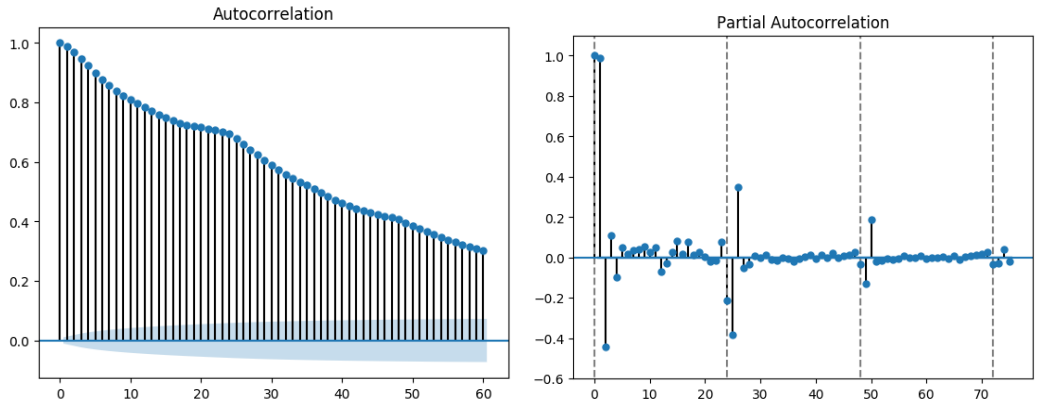


Figure 5.4. Autocorrelation structure of the residuals. The PACF highlights the presence of serial correlation on a regular daily basis, characterized by a short-term serial dependency. Furthermore, because of the huge sample size, the confidence band for PACF is so small that it is not even visible.

In any case, since the final goal is to produce a time-series of probabilistic forecasts, a modelling assumption on the ideal form of the residuals has to be introduced: they are supposed to be independent and normally distributed random variables, each one with an appropriate mean and variance. Analogously to what happens in the case of NAX, a Gaussian distribution is thus selected to describe the log-demand of each hour, and similarly the two characteristic parameters - the mean and the standard deviation - are predicted by means of a suitable Recurrent Neural Network.

However, because of the peculiar autocorrelation structure of the time-series of residuals, the network should be able to take into consideration not only the 1-lag autoregressive dependency - as the RNN of NAX does - but a larger period of time. Up to now, the weather effects have not been considered yet, thus for instance unusually warm and cold days are likely to generate a higher demand for several consecutive hours, and this fact is reflected on the corresponding residuals.

In order to tackle this problem, a specific Neural Network has to be designed: this is the most innovative part of the present thesis and currently, no similar architectures can be found in the literature of the energy sector. As anticipated, it is composed of two hidden layers and produces as output the pair (μ_t, σ_t), respectively the mean and the standard deviation of the distribution of the residual at time t. However, in order to reproduce the intricate dependency structure of the residuals, the network has to be able to provide as a recurrent input:

- ▷ any previous output to any layer,
- ▷ the average of the last n outputs to any layer.

In particular, we are interested $n = 24$, i.e. in providing as autoregressive input the average mean and standard deviation of the previous day. A scheme of the

proposed Neural Network is reported in Figure 5.5. In this regard, the presence of a single recurrent input in the hidden layers is just for graphical purposes; as said, it is indeed possible to feed each layer with more than one of the previous outputs, if required. Obviously implementing such a complex architecture is not a trivial task, and the following subsection is indeed devoted to offering more details.

As happens in the case of NAX, the loss function selected for the purpose is the negative Gaussian log-likelihood

$$\mathcal{L}(\mu_t, \sigma_t | r_t) = -\log \varphi(r_t, \mu_t, \sigma_t)$$

where clearly r_t is the residual at time t: therefore again a maximum likelihood approach is exploited for the calibration of the model.

The input layer of the network contains the dry-bulb and the dew-point temperatures plus the date, which has to be suitably transformed: *one-hot* encoding can be utilised, as well as a sinusoidal approach. In particular, when dealing with cyclical features, this latter can be preferred since it suggests to the network some further information about the relationships between the variables, for instance the fact that 11:00 is close to 12:00, Tuesday is close to Wednesday and September is close to October - something that *one-hot* encoding cannot do. Since it is not possible to determine *a priori* which is the ideal type of date encoding for this network, we will try them both during the validation phase (cf. Section 5.3.3).

Incidentally, each of the input features is *min-max* scaled in order to obtain variables that assume values in the same range, allowing them to contribute equally to the model fitting. In this regard, is important to remark that when a validation or test set is present, i.e. when the model is used not for *fitting* data but for *predicting*, the scaling has to be performed by taking the maximum and the minimum of each feature with respect to the training set only; indeed in a real-world problem, at the time of calibration, the only available data are the ones in the training set.

The output of the network represents instead the probability distribution of each residual, that has to be summed to the previously determined seasonal term $\text{Trend}_t + S_t$. As a consequence, the estimated distribution for the log-demand Y_t is a Gaussian random variable with mean $\text{Trend}_t + S_t + \mu_t$ and variance σ_t^2.

5.3.2 The network

The Recurrent Neural Network in Figure 5.5 is the core of the D-NAX model. As mentioned it is designed so as to admit a wide range of recurrent connections between the output and the other layers, which should enhance its ability to take advantage of long-term context. In this sense, their main effect is that they are useful to combat the problem of the vanishing gradient because they create *shortcuts*: this is the same principle at the base of *ResNets* and *Highway Networks* (cf. Subsection

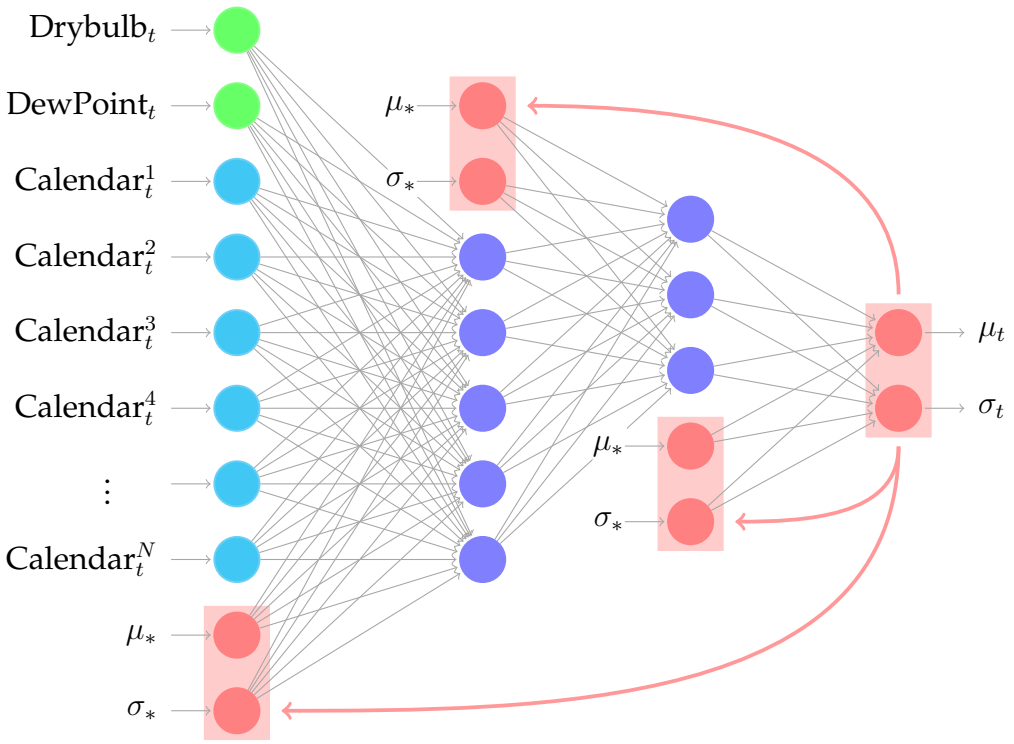


Figure 5.5. Scheme of the Recurrent Neural Network that is employed in the D-NAX model. It is formed by two hidden layers and has recurrent connections from the output to every layer. Moreover, as denoted by the symbols (μ_*, σ_*), these latter can involve any of the previous outputs (even more than one per each layer or the mean of the last ones).

2.1.8) and also NARX models (cf. Subsection 2.2.5). Moreover a valuable feature is the possibility for each layer of using as input the average of an arbitrary number of last outputs, which in principle could be helpful for the analysis of electrical demand.

In the general case, the hidden state of the first layer at time t is defined by the following equation:

$$\underline{h}_1^{(t)} = \mathcal{A}_1\Big(K_1\, \underline{x}^{(t)} + \sum_{i \in \mathcal{R}_1} R_{1,i}\, \underline{\hat{y}}^{(t-i)} + \frac{1}{n} M_1 \sum_{i=1}^{n} \underline{\hat{y}}^{(t-i)} + \underline{b}_1 \Big) \tag{5.1}$$

where K_1 and $\underline{b}_1$ are the usual kernel and bias of the layer, $R_{1,i}$ is the kernel associated with the recurrent output at lag i and M_1 is instead the kernel accounting for the average output. Is important to remark that the first sum is computed over an arbitrary set $\mathcal{R}_1$ of time-lags: in principle it is not necessary that the considered lags are consecutive.

Let us assume that the set that contains the indexes of the recurrent lags for the first layer, $\mathcal{R}_1$, has the following form: $\mathcal{R}_1 = \{i_1, \dots, i_m\}$. Then the previous expression can be rewritten as

$$\underline{h}_1^{(t)} = \mathcal{A}_1\Big(K_1\, \underline{x}^{(t)} + R_1^*\, \underline{y}_1^{*(t)} + \underline{b}_1 \Big) \tag{5.2}$$

with

$$R_1^* = [\, R_{1,i_1} \mid \dots \mid R_{1,i_m} \mid M_1 \,]$$

and

$$\underline{y}_1^{*(t)} = \begin{bmatrix} \underline{\hat{y}}^{(t-i_1)} \\ \vdots \\ \underline{\hat{y}}^{(t-i_m)} \\ \underline{\hat{y}}^{(t)}[n] \end{bmatrix}$$

where $\underline{\hat{y}}^{(t)}[n]$ is here used as a shorthand for the average of the last n outputs. In this way, the term $\underline{y}_1^{*(t)}$ accounts for all the recurrent connections from the output to the *first* layer. The advantage of this representation is that the recurrent inputs are separated by the exogenous variables: the overall formulation for the first layer is in practice equivalent to the one obtained for the original NAX (cf. equation (3.4)) with the difference that now the recurrent connections can involve multiple lags.

Considering analogous writings for the other layers, the network can be expressed by the equation

$$\begin{aligned} \underline{\hat{y}}^{(t)} &= K_3\, \underline{h}_2^{(t)} + R_3^*\, \underline{y}_3^{*(t)} + \underline{b}_3 = \\ &= K_3\, \mathcal{A}_2\Big(K_2\, \underline{h}_1^{(t)} + R_2^*\, \underline{y}_2^{*(t)} + \underline{b}_2 \Big) + R_3^*\, \underline{y}_3^{*(t)} + \underline{b}_3 = \\ &= K_3\, \mathcal{A}_2\Big(K_2\, \mathcal{A}_1\Big(K_1\, \underline{x}^{(t)} + R_1^*\, \underline{y}_1^{*(t)} + \underline{b}_1 \Big) + R_2^*\, \underline{y}_2^{*(t)} + \underline{b}_2 \Big) + R_3^*\, \underline{y}_3^{*(t)} + \underline{b}_3 \end{aligned} \tag{5.3}$$

The main point that has to be considered is how to train such a network: in order to perform the training procedure using a gradient-based algorithm, it is necessary to compute the gradient of the loss with respect to the weights $\underline{\boldsymbol{\theta}}$. However it is immediate to realize that BPTT may not be the most indicated procedure: indeed unrolling a network with more than a single recurrent connection leads to an exponential growth of the resulting graph, as shown in Figure 5.6. A Truncated Backpropagation Through Time can be in principle employed to tackle this issue; however, since we want to be able to use the average of the last, say, 24 or 48 outputs, even TBPTT is likely to require a huge computational cost.

Hence, a different approach has to be identified: we decide to use RTRL algorithm, which is introduced in Subsection 3.4.4 for *continually operating* Neural Network. Anyway, this algorithm is extremely general and can be applied - with more or less the same strengths and weaknesses - also in the case of *epochwise operating* networks. In the light of the results of Section 3.4, an *epochwise* operation approach with random shuffling is indeed selected for the occasion. Thus the training set for this network will be composed of a set of time windows: each of them is processed by the D-NAX network in order to produce a single output (*many-to-one* architecture).

An aspect that should be remarked is that when RTRL is used without allowing the real-time update of the weights - as in this case - the computed gradient is exact, contrarily to what happens in *continually* operating networks (cf. Subsection 3.4.4). This is another point in favour of RTRL, if compared for instance to TBPTT or other algorithms that are just able to produce an approximation of the gradient.

The network is implemented in C++ in order to obtain reasonably high per-

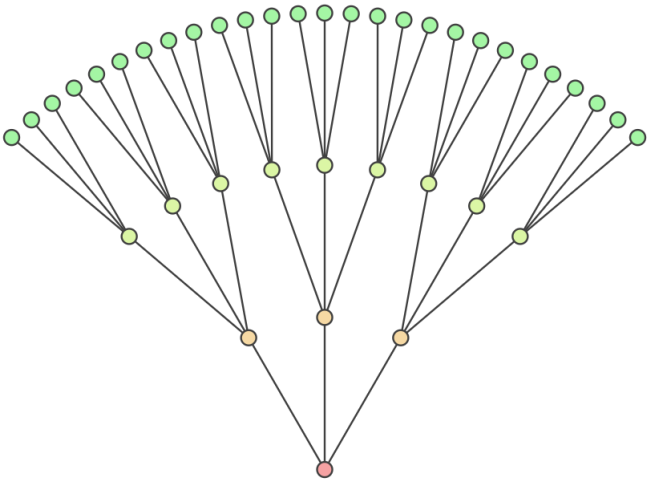


Figure 5.6. Exponential growth in RNN unrolling. When a network with more than a recurrent connection is unrolled, the number of branches in the resulting graph grows exponentially.

formances; the mathematical formulae used for computing the gradients for the present network can be found in Appendix C. Because of the strong computational effort required by the training - due to the large number of matrix multiplications involved and the increased magnitude of the dataset (if compared to the daily case) - the code is designed for parallel computing using MPI protocol.

In this regard, training this network on few years of data is not a task that can be accomplished easily by a standard PC. In our case, the algorithm is run on the MOX clusters for parallel applications (HPC) at Politecnico di Milano, in particular on 1 node with 20 cpu Intel Xeon E5-4610v2 @2.30GHz of *Gigat*.

5.3.3 Random search of hyperparameters

Because of the increased complexity of the architecture, a larger amount of hyperparameters should be determined. For each of the two hidden layers one has to define the number of neurons, the activation function and the type of required recurrent connections; as in the NAX case, the activation function of the output layer is instead considered as the linear one - a common practice in regression problems. Moreover, the learning rate, the batch size, the regularization parameter, the time range of the training set and the length of each sub-sequence are all hyperparameters to be selected. Due to the intense computational effort associated with every calibration of the model, a *random search* is performed instead of a complete *grid search*, based on the values listed in Table 5.2. Therefore not all the possible combinations of the hyperparameters are tested, but just a smaller random subset. In all the cases, Adam solver is used as optimizer to train the network.

As one may notice from the table, we decide not to consider recurrent connections from the output layer to itself: this is a design choice. Since the activation function of the output layer is the linear one, the presence of such a recurrent connection would just introduce a simple linear dependency of the output at time t from the previous one(s). Instead, we would like to encourage the network to study in-depth the intricate relationships between the variables.

Moreover, in addition to the standard hyperparameters, we also determine during this phase which is the best way to encode the input. In detail, we consider the pure *one-hot* encoding of all the calendar variables - which is, in general, the approach that produces the largest number of input variables - and a hybrid version, that uses *cyclical* encoding for the month-of-the-year and the hour-of-the-day and a *one-hot* encoding of the day-of-the-week. This second choice is motivated by the fact that the drop during weekends might be too relevant to be explained without a dummy variable approach, i.e. without indicating explicitly the day-of-the-week.

The overall procedure is performed using the year 2013 as validation set, as explained in Section 5.1, and the model is thus calibrated on a suitable number of

Hyperparameter	Value
Learning rate	0.003, 0.001
Batch size	50, 100
Regularization parameter	0.0001, 0
Time range of training set	3, 4 years
Sliding-window size	50, 100
Hidden neurons (1st layer)	3, 5
Hidden neurons (2nd layer)	8, 10, 15
Activation function (both layers)	ReLU, Sigmoid, Softmax, Swish
Recurrent connections (both layers)	None, {1,24}, {1,2,24}, {1,2,3,24}, {mean24}

Table 5.2. Set of considered hyperparameters for D-NAX model.

previous years (3 or 4 years, as indicated in the table).

5.3.4 The final version

As a result of the random search, the best hyperparameters associated with the optimization algorithm are found to be a learning rate equal to 0.001, a batch size of 50, no regularization and a 4-years long training set. Moreover, the length of each sub-sequence is selected to be equal to 50; in this regard, the choice of the maximum size of 100 for this hyperparameter is due to the computational cost associated with processing sequences that are too long.

The most interesting finding concerns the hyperparameters that define the structure of the network. Indeed, the best configuration is found to be the one that has a first layer composed of 10 hidden neurons, a *swish* activation function and the first three lagged output plus the output one day before ({1,2,3,24}). The second hidden layer is instead formed by 3 neurons, has a *softmax* activation function and the mean of the last 24 outputs as recurrent input. In practice, if we consider just the last three layers, we obtain a structure that is exactly analogous to the one of the original NAX, as shown in Figure 5.7.

This fact has a strong meaning from the point of view of the *feature extraction* that was discussed in the opening of the section. Indeed the intuition is that the second hidden layer is in charge of modelling the weekly and yearly seasonal part, just like NAX does, while the first hidden layer accounts for the extraction of the relevant features that are associated with the daily seasonality. To do so, the first hidden layer needs more precise details that are associated with the short-term context (i.e. the last 3 outputs) and the distribution of the residual that is observed

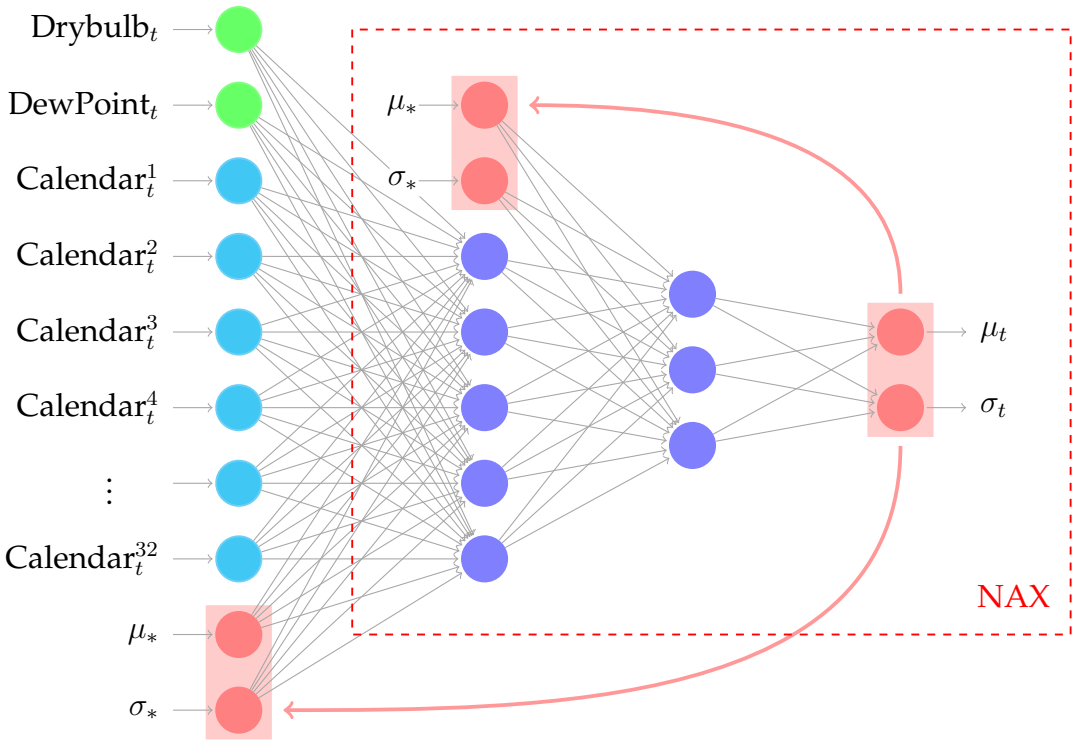


Figure 5.7. Scheme of the optimal D-NAX: as highlighted, the best performing model is found to be actually based on the NAX architecture.

one day before, which is found to be very relevant according to the initial PACF in Figure 5.4. Instead, the previous day context is enclosed in the mean of the last 24 outputs, which is provided as input to the second hidden layer. Since the residuals are supposed to be Gaussian and independent, the predicted distribution of the sum of the last 24 residuals is supposed to be a Gaussian as well, with mean and standard deviation

$$\mu_{\#} = \sum_{i=1}^{24} \mu_{t-i} \qquad \sigma_{\#} = \left(\sum_{i=1}^{24} \sigma_{t-i}^2 \right)^{\frac{1}{2}}$$

Hence the information transmitted via this recurrent connection is composed of a scaled version of the former and a scaled proxy for the latter. Moreover, the presence of the *swish* activation function is not particularly surprising, since it has been shown to outperform the standard *ReLU* in a multitude of cases (cf. Ramachandran et al. 2017).

This interpretation of the functioning of this complex network is a further reason for which the name D-NAX seems to be very appropriate for the proposed model.

Finally, the calendar variables are encoded with the hybrid approach: in detail, the best performing network is found to be the one fed with six harmonic pairs of

sines and cosines with period equal to 1 year to encode the day-of-the-year d

$$\{\cos(k\Omega d)\} \cup \{\sin(k\Omega d)\} \quad k \in [1,6] \quad \omega = \frac{1}{365.25}$$

six harmonic pairs of sines and cosines with period equal 1 day to encode the hour-of-the-day h

$$\{\cos(k\Theta h)\} \cup \{\sin(k\Theta h)\} \quad k \in [1,6] \quad \Theta = \frac{1}{24}$$

and a dummy encoding of the day of the week, plus an additional dummy variable for the holiday. Although it may seem a bizarre encoding, it should be noticed that it has the advantage of encoding the hour-of-the-day with just 12 variables, reducing thus the total number of neurons in the input layer. The final size of the exogenous input vector is 34, namely the 32 calendar variables and the 2 temperatures. However, it is also to be remarked that a complete *grid search* might have lead to the choice of different encoding.

5.4 Results

As mentioned in the introduction of this chapter, the proposed models are validated on the year 2013 and then tested on the years 2014 and 2015. In doing this, the optimal size of the training set - in terms of number of years - is selected during the validation phase and then kept fixed: as a consequence, the performances of D-NAX are tested using the data of 2010-2013 and 2011-2014 as training set, respectively.

In this regard, being H-NAX based on the suitable use of NAX, the optimal combination of hyperparameters that is found and discussed in Chapter 3 is used. The validation of H-NAX is just aimed at determining how many years of data should be used as training set: since, in practice, no relevant differences in terms of RMSE and APL are found, we decide to calibrate both H-NAX and D-NAX on the data from January 2010 to December 2013, to have a fair comparison.

Figure 5.8 shows the load forecasts of D-NAX on two different periods of the year, namely ten days of mid-March and of mid-August. The solid red line represents the point forecast of each hour, that is given by the mean of the predicted distribution; in this regard, we recall that for both D-NAX and H-NAX the load at every hour is modelled as a lognormal random variable X_t with parameters (μ_t, σ_t^2): hence each point forecast is given by

$$\mathbb{E}[X_t] = e^{\,\mu_t + \frac{1}{2}\sigma_t^2}$$

The shaded area around the point forecast is instead the 95% confidence interval associated with the point prediction, which in principle should provide us with

some information about the uncertainty thereof. As one can notice, even though the two depicted profiles are characterized by different intra-daily patterns (the ones that in Chapter 4 we called "M"-shaped and "arc"-shaped, respectively) the model is capable of predicting the realized dynamics with remarkable accuracy in both cases.

Analogous plots for model H-NAX are instead reported in Figure 5.9: in terms of predictive performances, its accuracy seems to be very close to the one of D-NAX. However, these two specific time ranges have been chosen to highlight the differences that may arise between the models in terms of confidence intervals. The ones of H-NAX indeed are larger and exhibit more irregular profiles: this is a clear consequence of the methodology of this model, which constructs the final forecast as a juxtaposition of the single predictions made by the 24 different models. The fact that, instead, D-NAX is able to partially capture the correlation between the terms enables it to reduce uncertainty. Nevertheless, on many other days the discrepancy between the confidence intervals produced by the two models is very subtle.

In order to have a practical intuition of the considerable accuracy of the two presented models, we can compare their predictions to the ones generated by the Tao model, which are plotted in Figure 5.10. At first sight, what is extremely evident is that the point forecast are far less accurate with respect to the ones of the RNN-based models and, as a consequence, the associated confidence intervals are inevitably larger; moreover similar results are also found for extended Fourier model.

In the remaining part of the section, we compare the four models under the quantitative perspective: we analyse firstly the accuracy of the point forecasts and then the property of sharpness and reliability of the predicted probability densities, by considering the results on the test year 2014; in this case, also the two linear models are calibrated using as training set the time-interval 2010-2013.

As mentioned in Chapter 1, point predictions are extremely important for the power industry in order to make operational decisions. An evaluation of the accuracy of the four models for the year 2014 is reported in Table 5.3. It can be noticed that both D-NAX and H-NAX outperform the linear models: in detail the RMSE of D-NAX is 45% less than the one of the Tao vanilla model, which is for sure a noticeable result. The two RNN-based models are quite similar in terms of accuracy; however, D-NAX can achieve better precision by exploiting the autocorrelation structure. Similarly, the two linear models are somehow equivalent in terms of RMSE and Tao is characterized by a lower relative error.

The relevant difference in terms of accuracy highlights the power of Recurrent Neural Networks in capturing the non-linear relationships that characterize the electrical demand. Moreover, it should be noticed that both D-NAX and H-NAX

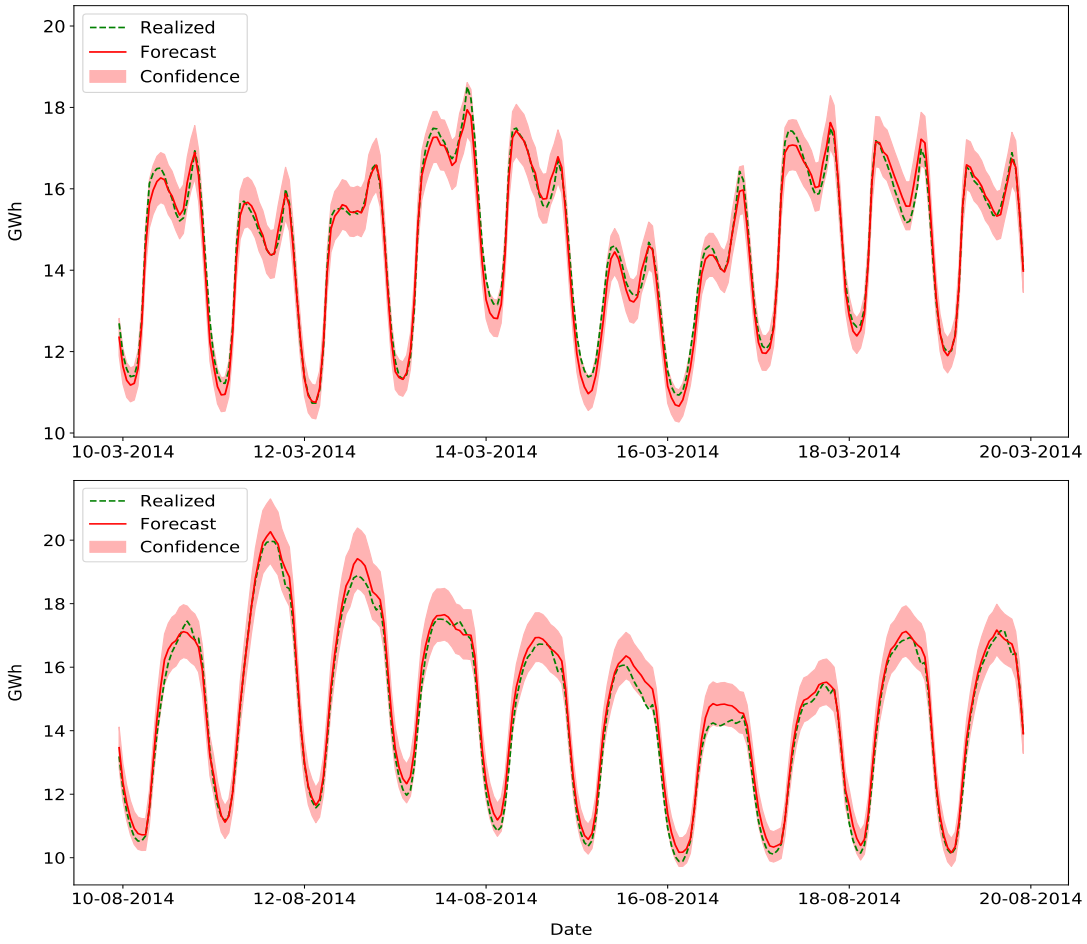


Figure 5.8. D-NAX load density forecast: realized (green line) and predicted power consumption (red line) during two different months, March and August; the shaded area instead represents the 95% confidence interval for the prediction. The forecasts seem to follow effectively the intra-daily behaviour of the load during every hour of the day, and in these particular cases, confidence intervals seem to behave appropriately.

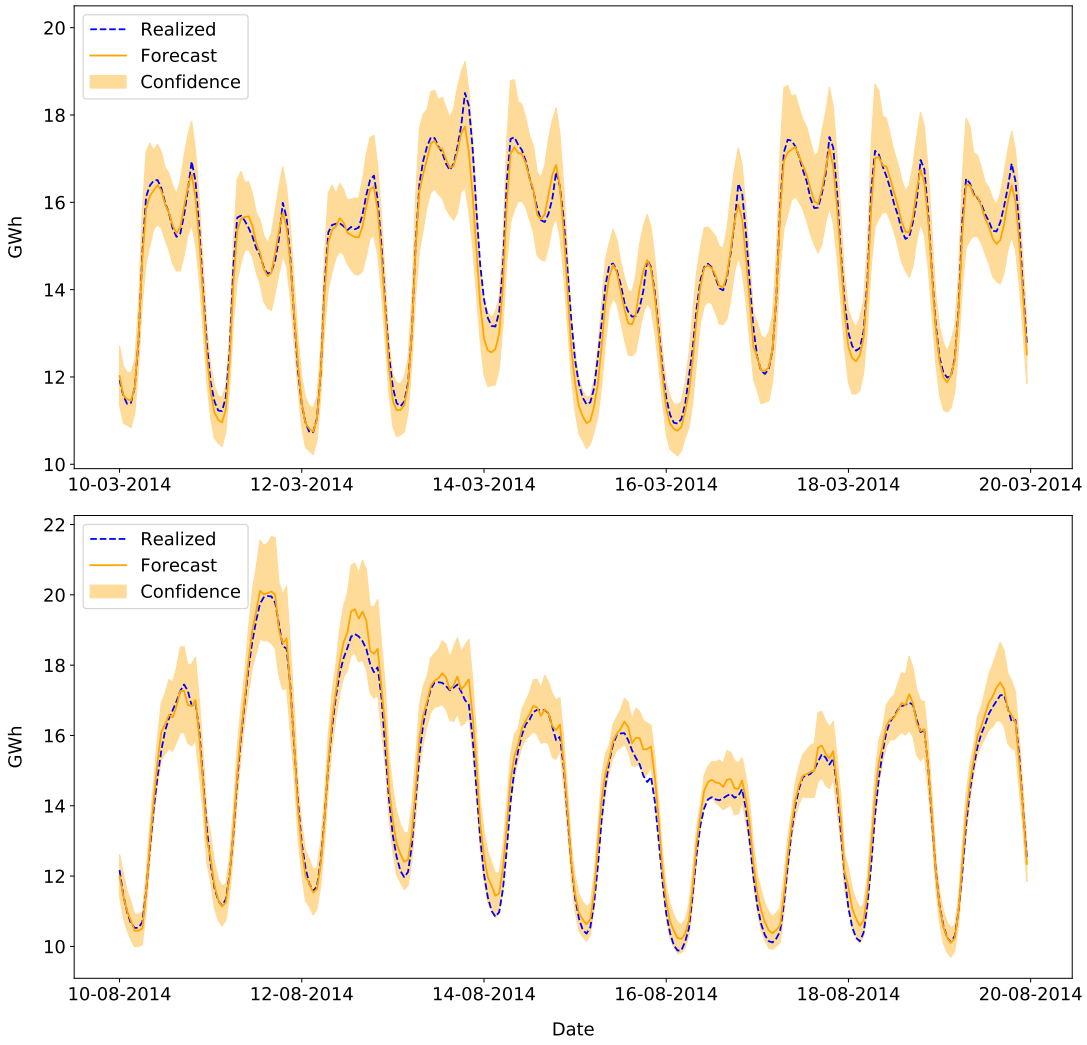


Figure 5.9. H-NAX load density forecast: realized (blue line), predicted power consumption (orange line) and 95% confidence interval (yellow area) during March and August, as done in Figure 5.8. Also in these cases the predictions seem to be accurate; moreover, it can be noticed how the confidence intervals associated with night hours are smaller than the ones that are generated for daytime hours.

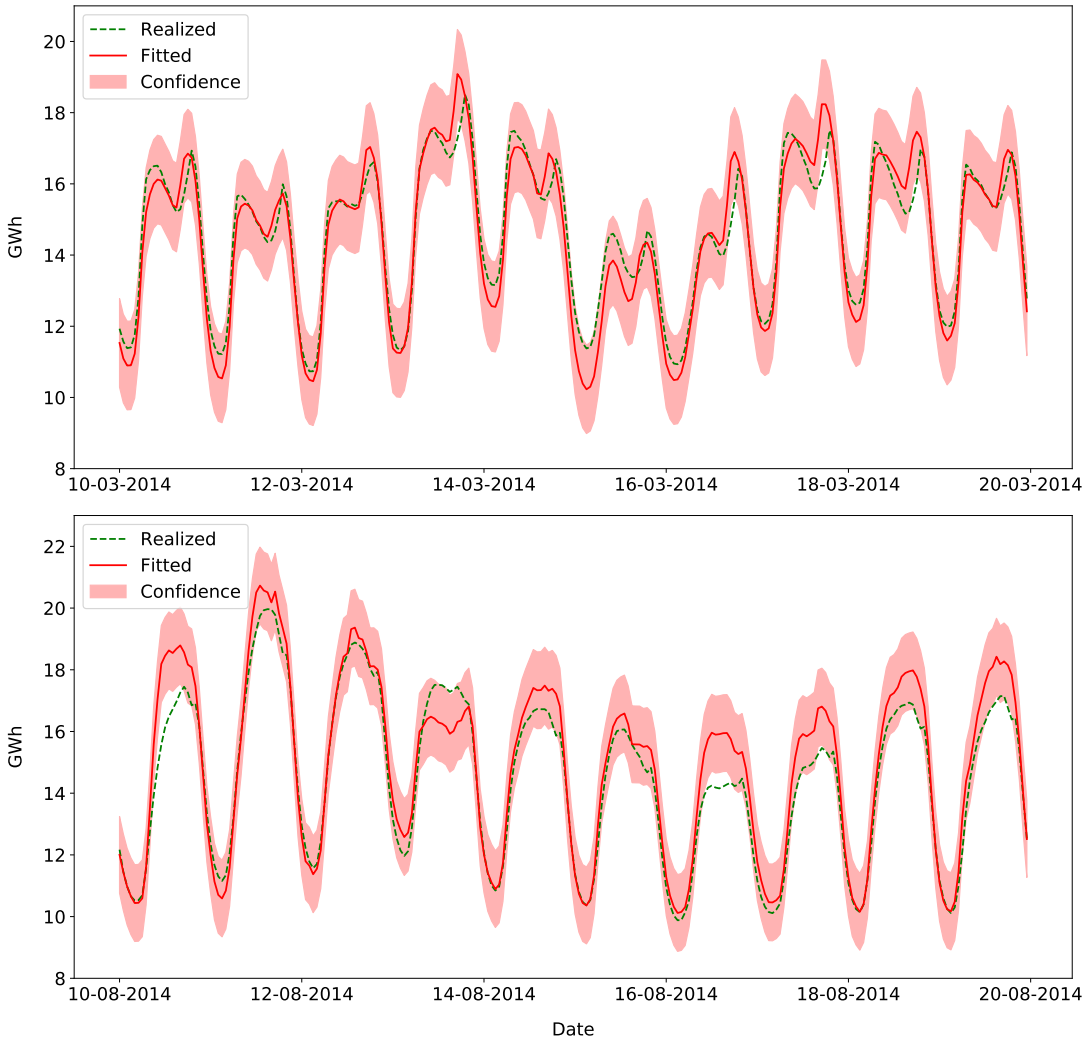


Figure 5.10. Tao model load density forecast. As before, realized (green line), predicted power consumption (red line) and 95% confidence interval (shaded red area). These forecasts, however, are significantly less accurate and precise than the ones generated by H-NAX and D-NAX, and the associated prediction intervals are indeed larger.

take as input variable also the dew-point temperature, whilst the other two models do not: this variable is very important because, if suitably combined with dry-bulb temperature, it allows to deduce the relative humidity, a central climatic factor that has to be taken into account. The advantage of using Neural Networks is that they learn from historical data how to use at best this new information, without the need of specifying how the impact of temperatures changes, for instance, according to the season or to the period of the day.

For evaluating the *sharpness* of the predicted distributions, instead, we consider the Pinball Loss (cf. Subsection 1.4): the results on percentiles are shown in Figure 5.11. Also in this case D-NAX obtains slightly better performances with respect to H-NAX on average; anyway, both models are able to produce more realistic distributions with respect to the ones of Tao and extended Fourier. This is mainly due to the fact that, in the former models, the shape of the densities is allowed to vary, since every prediction comes with a different standard deviation; this feature is very important in order to describe the heteroskedastic behaviour of the time-series.

Furthermore, we can also consider Winkler score, that - as already said - is built in such a way to reward the models that are able to produce narrower confidence intervals. Indeed it is possible to notice that the Winkler score of D-NAX is always lower than the one of H-NAX, and more in general that, also according to this criterion, the two proposed models achieve better performances with respect to the linear ones.

Lastly, concerning the *reliability* of the produced distributions, we use the backtesting procedure to evaluate whether the confidence intervals are actually accurate. We consider different confidence levels and, for each of them, we compute how many times the realized hourly load falls within the predicted interval; the results of this analysis are shown in Figure 5.12. In this case, we notice that H-NAX and extended Fourier obtain the best results, while the tendency of D-NAX to produce confidence intervals that are too small causes the model to be less reliable than the other ones. This is for sure something that should be considered in the overall evaluation, since in general the inability of predicting the right tail of a

	D-NAX	H-NAX	Tao	EF
RMSE [MWh]	362.0	419.2	694.3	693.8
MAPE [%]	1.90	2.11	3.27	3.69

Table 5.3. Accuracy results for the 4 models on the test set 2014. MAPE and RMSE (in MWh) are considered in the comparison; the two proposed models are able to achieve better predictive performances with respect to the linear models.

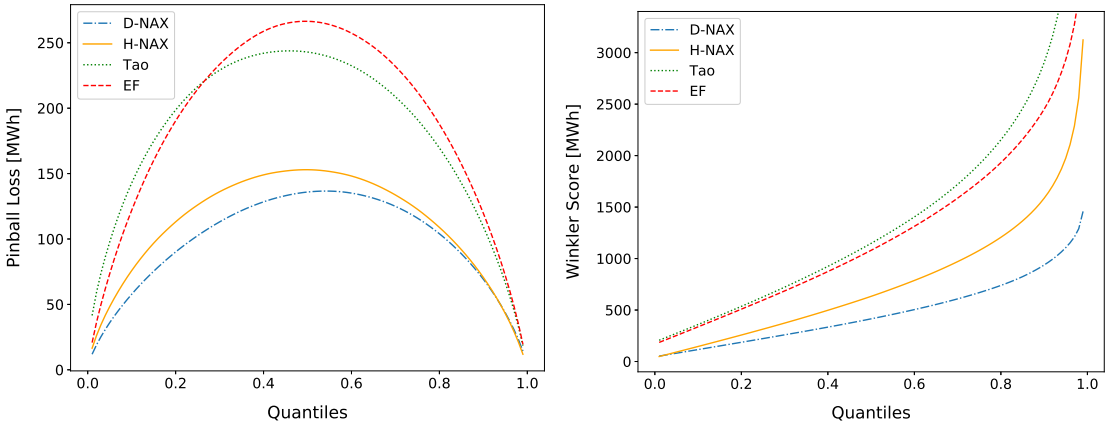


Figure 5.11. Pinball loss (*left*) and Winkler score (*right*) for the 4 models on the test set 2014. Sharpness is evaluated by means of these quantitative criteria, which are computed for each percentile. In detail, it is possible to notice that the two proposed models are characterized by lower values of the statistics in both cases.

probability distribution may lead to severe consequences.

Lastly, Table 5.4 contains a summary of the results of the models in different years: the validation set (2013) and the two test sets (2014 and 2015). Also in these cases, D-NAX and H-NAX are reported to outperform the benchmarks (here, for the sake of compactness, only the Tao model is considered).

	D-NAX			H-NAX			Tao		
Year	MAPE	RMSE	APL	MAPE	RMSE	APL	MAPE	RMSE	APL
2013	1.90	372	101	2.23	458	120	3.11	679	179.5
2014	1.90	362	99	2.11	419	112	3.27	694	183
2015	2.44	435	124	2.58	477	132	3.57	722	194

Table 5.4. Comparison of the models in terms of MAPE, RMSE and APL on different years; MAPE is measured in %, RMSE and APL in MWh. The first year (2013) reflects the results on the validation set, while the other two show the results of the testing phase of the proposed models.

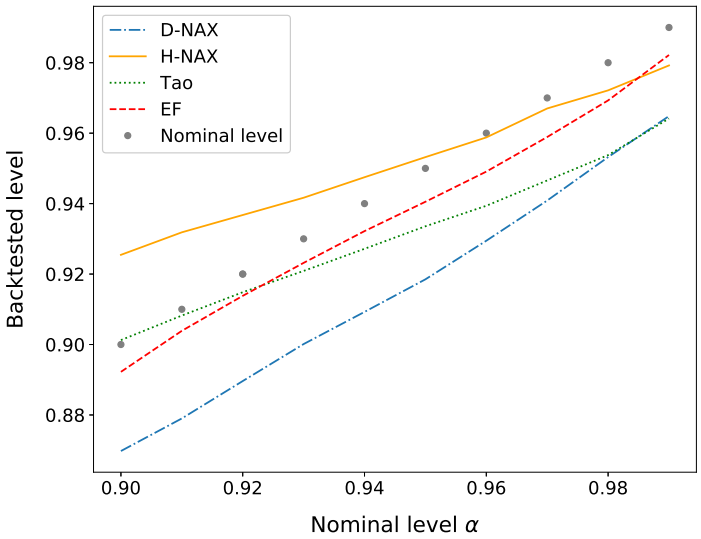


Figure 5.12. Backtested confidence intervals for the 4 models on the test set 2014. Their reliability is tested by considering different confidence levels α from 90% to 99%: it is possible to notice that the behaviour of H-NAX (in orange) is extremely compatible with the nominal level; instead D-NAX (in blue) is not very reliable, as a consequence of the fact that its confidence intervals are too narrow to contain large deviations.

Chapter 6

Conclusions

In this thesis we have addressed the problem of intra-daily probabilistic load forecasting, focusing on the impact that weather conditions have on power consumption. The obtained results can be summarized as follows:

- we have presented two novel models for *hourly* PLF, called D-NAX and H-NAX. Both rely on the use of a parametric approach: each predicted probability density is indeed a lognormal distribution. This modelling assumption is found to be effective in capturing the most relevant features of the intra-daily load dynamics. Besides, we have managed to model effectively the serial correlation that characterizes the electrical load by means of Recurrent Neural Networks. In this regard, we stress that we have implemented the D-NAX model in C++ because its structure is new to the literature and not available in `Keras` or in other standard machine learning libraries.

- we have assessed the predictive accuracy of the two proposed models, showing that both are able to outperform the benchmark models. The most remarkable result is the very high accuracy that characterizes the point forecasts of D-NAX and H-NAX: for instance, in 2014 the RMSE is, respectively, about 45% and 40% lower than the one of the benchmarks. Also in terms of sharpness and reliability, the predicted distributions are found to be definitely more appropriate than the ones prescribed by the linear models.

- we have shown that a fine-tuning of the hyperparameters in the NAX model can lead to a further improvement of its ex-post predictive accuracy. This is important not only to obtain better forecasts of daily load, but also because the NAX model is a central building block for both D-NAX and H-NAX.

Some final words should be spent on the comparison between D-NAX and H-NAX: the two models have proven to have excellent characteristics and different strengths. H-NAX is simpler. This is always a point in favour of a model because it

is easier to be implemented, faster to be calibrated and more straightforward to be used.

D-NAX has instead a more challenging and ambitious architecture, and somehow is in between *vanilla* RNNs and usual NARX models. It can be noted that it obtains in general better results than H-NAX, both in terms of accuracy and sharpness; however, it requires a longer training time and is less reliable than H-NAX. As already mentioned, this fact can be dangerous for practical applications since the underestimation of the electrical load increases - for instance - the risk of power outages; in this sense, we may say that H-NAX is more conservative. For all these reasons, we do not claim that D-NAX is definitely the best choice, at the moment; anyway, there is still room for improvements that in principle may allow this latter model to outperform the rival.

6.1 Further Developments

The research on this topic could proceed in several ways: in the following, we propose some suggestions to anyone who wants to perform further insights.

First of all, the architecture and the training of D-NAX could be further investigated. For instance, the initial *random* search can be extended to a complete *grid* search to analyse the goodness of the selected hyperparameters and - hopefully - enhance the accuracy of the model. In this regard, one of the most critical issues that we faced during the phase of model design was the huge amount of hyperparameters that could in principle be selected: the number and the type of recurrent connections, the number of neurons, the activation functions, the training-related parameters, the encoding of the input, *etcetera*. Due to the computational time required for each training, we had to make some choices, but there are still many configurations that can be tested.

Second, the ex-ante predictions should be assessed. In this regard, Azzone & Baviera (2021) addressed this issue by simulating a large number of temperature scenarios and deducing the ex-ante distributions (cf. Azzone & Baviera 2021 and references therein).

Lastly, it is important to mention that nowadays in some fields of Artificial Intelligence - above all the one of Natural Language Processing (NLP) - both RNNs and LSTMs are less frequently used (cf. Karita et al. 2019). This fact is mainly due to the introduction of the *Transformer* (Vaswani et al. 2017), an architecture that makes use of the so-called *attention mechanisms*. These are some techniques which are designed to mimic the human cognitive attention; in short, *attention* is purely based on a more refined notion of *context*. Applications of these mechanisms to time-series analysis are not yet very popular (cf. e.g. Du et al. 2020): it could be a possibility to investigate how to introduce attention in hourly PLF.

Appendix A

Activation Functions

As mentioned in Chapter 2, the excitation level of each neuron, and therefore its ability to communicate with its neighbours, is expressed by means of an *activation function*: a list of the most used ones is here provided. For all of them, in order to perform the backpropagation procedure, the computation of the Jacobian matrix is required: in almost all the cases we obtain a diagonal matrix, because of the componentwise action of the involved functions. In accordance with the usual notation, each activation function is denoted as

$$\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^n$$

and its Jacobian matrix as $\mathcal{A}'$.

Sigmoid

Sigmoid activation function transforms each component x_i as follows

$$\sigma(x_i) = \frac{1}{1 + e^{-x_i}} = \frac{e^{x_i}}{e^{x_i} + 1}$$

Its derivative is given by

$$\sigma'(x_i) = e^{-x_i}\sigma(x_i)^2 = \sigma(x_i)(1 - \sigma(x_i))$$

and therefore for each entry of the Jacobian matrix the following relation holds

$$[\,\mathcal{A}'\,]_{ij} = \frac{\mathrm{d}\sigma(x_i)}{\mathrm{d}x_j} = \sigma(x_i)(1 - \sigma(x_i))\;\delta_{ij} \leq \frac{1}{4}$$

entailing that $\|\mathcal{A}'\|_\infty = \left\|(\mathcal{A}')^T\right\|_\infty \leq \frac{1}{4}$.

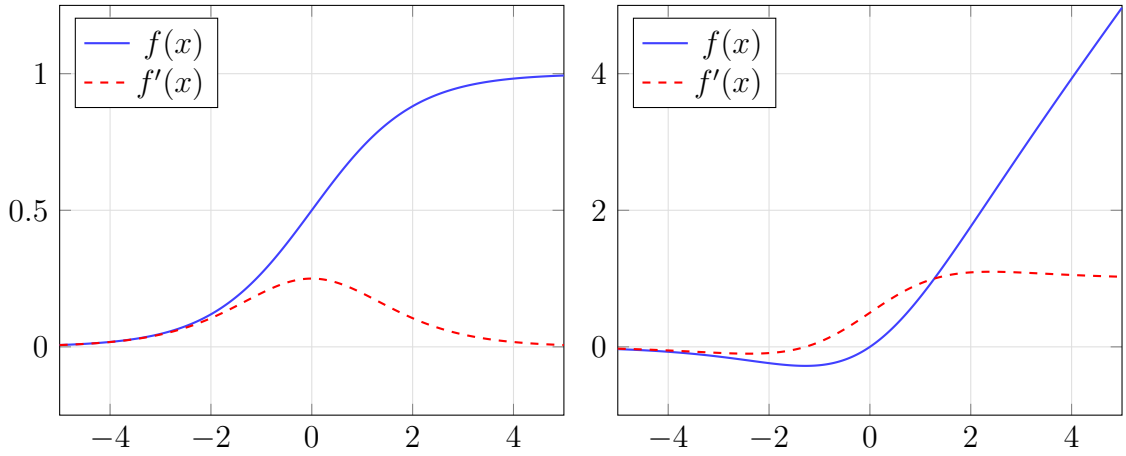


Figure A.1. Plot of Sigmoid (*left*) and of Swish (*right*) with their first derivatives

Swish

Swish activation function is a modification of the sigmoid which has been recently introduced by a team of researchers of Google Brain (cf. Ramachandran et al. 2017). It is defined as

$$f(x_i) = x_i \sigma(x_i)$$

and its componentwise derivative reads

$$f'(x_i) = \sigma(x_i) + x_i \sigma(x_i)(1 - \sigma(x_i))$$

In this case it is easy to show that such derivative is greater than 1 for $x_i \geq x^*$, with x^* solution of $x^* - 1 = e^{-x^*}$. Hence for Swish the infinity norm of the Jacobian matrix is not less than 1; however it is possible to prove its boundedness and numerically it is found that $\|\mathcal{A}'\|_\infty = \left\|(\mathcal{A}')^T\right\|_\infty < 1.1$.

ReLU

The *Rectified Linear Unit* (ReLU) activation function is expressed by the componentwise equation

$$f(x_i) = \begin{cases} x_i & \text{if} \quad x_i \geq 0 \\ 0 & \text{if} \quad x_i < 0 \end{cases}$$

Thus its derivative is just the Heaviside step function and the infinity norm of its Jacobian matrix (and of its transpose) is less or equal than 1.

Many variations of the ReLU have been introduced to tackle a typical issue of ReLU, the *dead neuron* problem, which consists in the fact that both the value of the function and of its first derivative are zero when x_i is negative (cf. Goodfellow et al. 2016, Chapter 6.3.1).

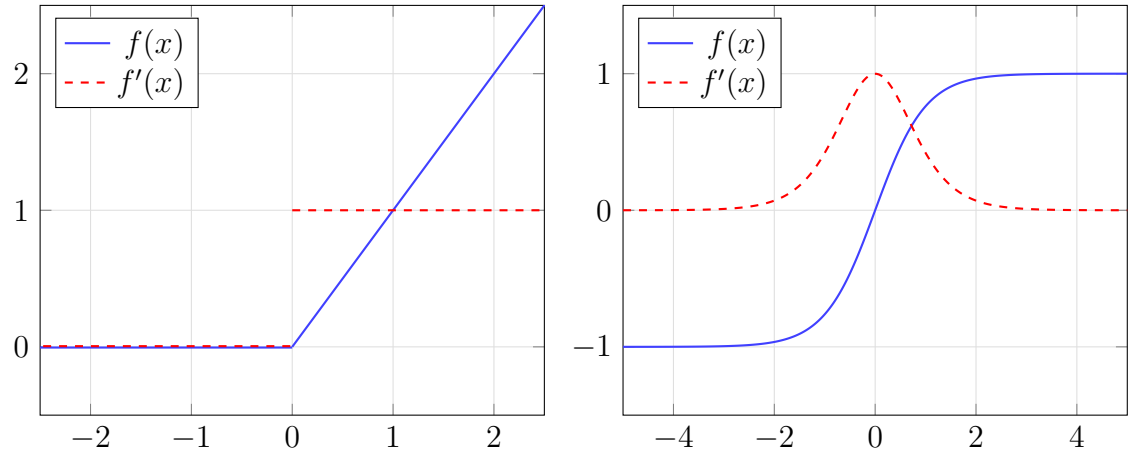


Figure A.2. Plots of ReLU (*left*) and of Tanh (*right*) with their first derivatives

Tanh

The *hyperbolic tangent* is another popular activation function, that is by construction allowed to assume negative values. Its mathematical definition is

$$f(x_i) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} = 2\sigma(2x_i) - 1$$

It is thus possible to notice that it is a rescaled version of the sigmoid; for this reason, it can be deduced that even in this case $\|\mathcal{A}'\|_\infty = \left\|(\mathcal{A}')^T\right\|_\infty \leq 1$.

Softmax

The last activation function that has a relevant role in Deep Learning is called *Softmax*. This is a very particular activation function because it does not act componentwise, and is indeed designed for classification problems: in these cases, the output of the network is expected to be a (discrete) probability distribution on the values of the target space (cf. Aggarwal 2018, Chapter 3.2.5). It is defined as follows

$$[\,\mathcal{A}\,]_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}$$

Hence it is possible to deduce that the Jacobian matrix of the application is a dense matrix; the elements that compose its diagonal are

$$[\,\mathcal{A}'\,]_{ii} = [\,\mathcal{A}\,]_i - [\,\mathcal{A}\,]_i^2$$

while the off-diagonal components read

$$[\,\mathcal{A}'\,]_{ij} = -\frac{e^{x_j}}{e^{x_i}}[\,\mathcal{A}\,]_i^2$$

Moreover, also in this case it is possible to compute $\left\|(\mathcal{A}')^T\right\|_\infty$: the absolute row sum S_j for row j is given by

$$\begin{aligned} S_j &= \sum_{i=1}^{n} [\ (\mathcal{A}')^T\ ]_{ji} = \sum_{i=1}^{n} [\ \mathcal{A}'\ ]_{ij} = [\ \mathcal{A}\ ]_j - [\ \mathcal{A}\ ]_j^2 + \sum_{i \neq j} \frac{e^{x_j}}{e^{x_i}} [\ \mathcal{A}\ ]_i^2 = \\ &= [\ \mathcal{A}\ ]_j - [\ \mathcal{A}\ ]_j^2 + [\ \mathcal{A}\ ]_j \sum_{i \neq j} [\ \mathcal{A}\ ]_i = [\ \mathcal{A}\ ]_j - 2[\ \mathcal{A}\ ]_j^2 + [\ \mathcal{A}\ ]_j \sum_{i=1}^{n} [\ \mathcal{A}\ ]_i = \\ &= 2[\ \mathcal{A}\ ]_j - 2[\ \mathcal{A}\ ]_j^2 \leq \frac{1}{2} \end{aligned}$$

This entails that $\left\|(\mathcal{A}')^T\right\|_\infty = \max_j S_j \leq \frac{1}{2}$.

Appendix B

RTRL for NAX network

B.1 Derivation of formulae

The neural network employed in NAX is represented by the equation (3.4), which is reported for convenience:

$$\underline{\hat{y}}^{(t)} = K_2 \underbrace{\mathcal{A}_1\big(K_1\,\underline{x}^{(t)} + R_1\,\underline{\hat{y}}^{(t-1)} + \underline{b}_1\big)}_{\underline{h}^{(t)}} + \underline{b}_2$$

In alternative, it can be written as:

$$\underline{\hat{y}}^{(t)} = \mathcal{D}_2 \circ \mathcal{A}_1 \circ \mathcal{D}_1\,(\,\underline{\tilde{x}}^{(t)}\,) \tag{B.1}$$

where both $\mathcal{D}_1$ and $\mathcal{D}_2$ are affine applications and

$$\underline{\tilde{x}}^{(t)} = \begin{bmatrix} \underline{x}^{(t)} \\ \underline{\hat{y}}^{(t-1)} \\ 1 \end{bmatrix}$$

In Chapter 2 the symbols $\mathcal{D}_1$ and $\mathcal{D}_2$ are used to express a feed-forward dense connection between two consecutive layers. This still holds in the case of $\mathcal{D}_2$ (because of the absence of recurrent connections from the output layer to itself); however for simplicity of notation the symbol $\mathcal{D}_1$ with the convention that

$$\mathcal{D}_1\,(\,\underline{\tilde{x}}^{(t)}\,) = K_1\,\underline{x}^{(t)} + R_1\,\underline{\hat{y}}^{(t-1)} + \underline{b}_1$$

Now, we may imagine to call $\underline{\theta}$ the vector that collects all the weights of $\mathcal{D}_1$ and $\underline{\varphi}$ the one for $\mathcal{D}_2$, so that actually

$$\underline{\hat{y}}^{(t)} = \mathcal{D}_2\Big(\,\mathcal{A}_1\,\big(\,\mathcal{D}_1\,(\,\underline{\tilde{x}}^{(t)}, \underline{\theta}\,)\big)\,, \underline{\varphi}\Big) \tag{B.2}$$

while the loss is given by

$$\ell^{(t)} = \mathscr{L}(\underline{y}^{(t)}, \hat{\underline{y}}^{(t)}) \tag{B.3}$$

It is possible to compute explicitly the Jacobian matrices of the inferred output $\hat{\underline{y}}^{(t)}$ with respect to $\underline{\theta}$ and $\underline{\varphi}$, getting

$$\begin{aligned} \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\theta}} &= K_2 \, \mathcal{A}_1' \, \frac{\mathrm{d}\mathcal{D}_1(\tilde{\underline{x}}^{(t)}, \underline{\theta})}{\mathrm{d}\underline{\theta}} \\ \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\varphi}} &= \frac{\partial \mathcal{D}_2}{\partial \underline{\varphi}} + K_2 \, \mathcal{A}_1' \, \frac{\mathrm{d}\mathcal{D}_1(\tilde{\underline{x}}^{(t)}, \underline{\theta})}{\mathrm{d}\underline{\varphi}} \end{aligned} \tag{B.4}$$

Because of the recurrent connections, $\tilde{\underline{x}}^{(t)} = \tilde{\underline{x}}^{(t)}(\underline{\theta}, \underline{\varphi})$. In particular the recurrent dependency affects only $\hat{\underline{y}}^{(t-1)}$ and not the $\underline{x}^{(t)}$ which is provided to the network as an exogenous vector. As a consequence, the equations in (B.4) can be expanded as

$$\begin{aligned} \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\theta}} &= K_2 \, \mathcal{A}_1' \left(\frac{\partial \mathcal{D}_1}{\partial \underline{\theta}} + R_1 \frac{\mathrm{d}\hat{\underline{y}}^{(t-1)}}{\mathrm{d}\underline{\theta}} \right) \\ \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\varphi}} &= \frac{\partial \mathcal{D}_2}{\partial \underline{\varphi}} + K_2 \, \mathcal{A}_1' \, R_1 \, \frac{\mathrm{d}\hat{\underline{y}}^{(t-1)}}{\mathrm{d}\underline{\varphi}} \end{aligned} \tag{B.5}$$

Notice that $\partial \mathcal{D}_1 / \partial \underline{\theta}$ is a matrix of the form of equation (2.7), but now it contains also the recurrent inputs. This means that its structure is given by

$$\frac{\mathrm{d}\mathcal{D}_1}{\mathrm{d}\underline{\theta}} = \begin{bmatrix} \underline{x}^T & \cdots & \underline{0}^T & [\hat{\underline{y}}^{(t-1)}]^T & \cdots & \underline{0}^T & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \underline{0}^T & \cdots & \underline{x}^T & \underline{0}^T & \cdots & [\hat{\underline{y}}^{(t-1)}]^T & 0 & \cdots & 1 \end{bmatrix} \tag{B.6}$$

Anyway, a different choice for the vectorisation may lead to a slightly different formulation of this matrix. Instead $\partial \mathcal{D}_2 / \partial \underline{\varphi}$, which does not involve any recurrent connection, has exactly the same structure as (2.7).

Equation (B.5) can be rewritten as

$$\begin{aligned} \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\theta}} &= K_2 \, \mathcal{A}_1' \, \frac{\partial \mathcal{D}_1}{\partial \underline{\theta}} + K_2 \, \mathcal{A}_1' \, R_1 \frac{\mathrm{d}\hat{\underline{y}}^{(t-1)}}{\mathrm{d}\underline{\theta}} \\ \frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\varphi}} &= \frac{\partial \mathcal{D}_2}{\partial \underline{\varphi}} + K_2 \, \mathcal{A}_1' \, R_1 \, \frac{\mathrm{d}\hat{\underline{y}}^{(t-1)}}{\mathrm{d}\underline{\varphi}} \end{aligned} \tag{B.7}$$

In practice, the intuition is that the Jacobian matrices have the following form: they are the sum of a first term due to the FNN-like flow at time t and a second term due to the recurrent connections. It is clear that, in absence of feedback links, R_1 is the null matrix and the standard FNN equations are obtained.

Once the two Jacobian matrices are known, the two gradients that are used in the optimization procedure are just obtained as

$$\begin{aligned} \frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\theta}} &= \nabla^T \mathscr{L} \, \frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\theta}} \\ \frac{\mathrm{d}\ell^{(t)}}{\mathrm{d}\underline{\varphi}} &= \nabla^T \mathscr{L} \, \frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\varphi}} \end{aligned} \tag{B.8}$$

B.2 Estimate of complexity

In Chapter 3 the time complexity for each iteration of the RTRL for a vanilla RNN is estimated as $\mathcal{O}(k^2|\underline{\boldsymbol{\theta}}|)$, with k size of recurrent state and $|\underline{\boldsymbol{\theta}}|$ total number of parameters. It is easy to show that this is applies also to this case: consider for instance the first equation in (B.7):

$$\underbrace{\frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\theta}}}_{k\times|\underline{\theta}|} = \underbrace{K_2}_{k\times h} \underbrace{\mathcal{A}_1'}_{h\times h} \left(\underbrace{\frac{\partial \mathcal{D}_1}{\partial \underline{\theta}}}_{h\times|\underline{\theta}|} + \underbrace{R_1}_{h\times k} \underbrace{\frac{\mathrm{d}\underline{\hat{y}}^{(t-1)}}{\mathrm{d}\underline{\theta}}}_{k\times|\underline{\theta}|} \right) \tag{B.9}$$

where h is the number of hidden neurons (i.e. the size of the hidden state). Since in the NAX case $k = 2$ and $h = 3$, it is optimal to compute $K_2 \, \mathcal{A}_1'$, with a cost of $\mathcal{O}(kh^2)$, or $\mathcal{O}(kh)$ in case the activation function is not a *softmax*.

Then the obtained matrix is multiplied together with the first term in parenthesis, which is a sparse Jacobian of the form in equation (2.7), leading to a $\mathcal{O}(k|\underline{\theta}|)$. Eventually the products between $K_2 \, \mathcal{A}_1'$ (suitably stored to avoid duplicate calculation) and the last two terms are performed, which cost a number of operations of the order of $\mathcal{O}(k^2 h)$ and $\mathcal{O}(k^2|\underline{\theta}|)$, respectively. The very last operation is the one involving the gradient $\nabla^T \mathscr{L}$: $\mathcal{O}(k|\underline{\theta}|)$.

Summing up all the contributions, it is possible to notice that the most expensive task is the one associated to the Jacobian matrix $\mathrm{d}\underline{\hat{y}}^{(t-1)} \Big/ \mathrm{d}\underline{\theta}$. Notice that $|\underline{\theta}|$, which here is the number of parameters of the first layer, is equal to $k(m + h + 1)$, where m is the number of purely exogenous inputs. This can thus result into a very large computational cost of the algorithm when many inputs are provided, or when the size k of the recurrent state is significant.

Obviously an analogous estimate holds also for the second equation of (B.7), leading to a total complexity of $\mathcal{O}(k^2|\underline{\boldsymbol{\theta}}|)$, as claimed.

Appendix C

RTRL for D-NAX network

C.1 Derivation of formulae

The RTRL formulae for the D-NAX network are analogous to the ones derived in Appendix B, with the greater difficulty that multiple recurrent connections are now involved.

The network is represented by equation (5.3), which reads:

$$\underline{\hat{y}}^{(t)} = K_3\, \mathcal{A}_2\Big(K_2\, \mathcal{A}_1\Big(K_1\, \underline{x}^{(t)} + R_1^*\, \underline{y}_1^{*(t)} + \underline{b}_1\Big) + R_2^*\, \underline{y}_2^{*(t)} + \underline{b}_2\Big) + R_3^*\, \underline{y}_3^{*(t)} + \underline{b}_3$$

However, for the sake of simplicity, we split the previous relationship into three parts as follows

$$\underline{h}_1^{(t)} = \mathcal{A}_1\Big(K_1\, \underline{x}^{(t)} + R_1^*\, \underline{y}_1^{*(t)} + \underline{b}_1\Big) = \mathcal{A}_1\, \tilde{K}_1\, \underline{\tilde{x}}^{(t)} \tag{C.1}$$

$$\underline{h}_2^{(t)} = \mathcal{A}_2\Big(K_2\, \underline{h}_1^{(t)} + R_2^*\, \underline{y}_2^{*(t)} + \underline{b}_2\Big) = \mathcal{A}_2\, \tilde{K}_2\, \tilde{\underline{h}}_1^{(t)} \tag{C.2}$$

$$\underline{\hat{y}}^{(t)} = K_3\, \underline{h}_2^{(t)} + R_3^*\, \underline{y}_3^{*(t)} + \underline{b}_3 = \tilde{K}_3\, \tilde{\underline{h}}_2^{(t)} \tag{C.3}$$

As usual the *tilde* notation is meant to separate the trainable weights of the layer from the state variables. For instance

$$\tilde{K}_3 = [\; K_3 \mid R_3^* \mid \underline{b}_3 \;]$$

and

$$\tilde{\underline{h}}_2^{(t)} = \begin{bmatrix} \underline{h}_2^{(t)} \\ \underline{y}_3^{*(t)} \\ 1 \end{bmatrix}$$

In this case, one should remember that R_3^* and $\underline{y}_3^{*(t)}$ are respectively a matrix and a vector that are formed in the same way (cf. equation (5.2)).

Now, we consider that all the weights of the $\tilde{K}_1$ are suitably collected in a vector $\underline{\theta}$, the weights of $\tilde{K}_2$ in a vector $\underline{\varphi}$ and the ones of $\tilde{K}_3$ in a vector $\underline{\psi}$. Let us differentiate equation (C.3) with respect to $\underline{\psi}$:

$$\frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\psi}} = \frac{\mathrm{d}\big(\tilde{K}_3\,\underline{\tilde{h}}_2^{(t)}\big)}{\mathrm{d}\underline{\psi}} = \frac{\mathrm{d}\tilde{K}_3}{\mathrm{d}\underline{\psi}}\,\underline{\tilde{h}}_2^{(t)} + \tilde{K}_3\,\frac{\mathrm{d}\underline{\tilde{h}}_2^{(t)}}{\mathrm{d}\underline{\psi}} \tag{C.4}$$

The first term of the RHS is a sparse matrix, extension of the one in equation (B.6), suitably adapted so that all the different vectors that compose the extended vector $\underline{\tilde{h}}_2^{(t)}$ are taken into account. It is clear that in principle it is possible to write $\tilde{K}_3\,\underline{\tilde{h}}_2^{(t)}$ as an affine map $\mathcal{D}_3(\underline{\tilde{h}}_2^{(t)})$, as done both in Chapter 2 and in Appendix B; however we are convinced that in this case is better to use this other notation because of the complex structure of matrix $\tilde{K}_3$.

The second term instead can be expanded as

$$\tilde{K}_3\,\frac{\mathrm{d}\underline{\tilde{h}}_2^{(t)}}{\mathrm{d}\underline{\psi}} = K_3\,\frac{\mathrm{d}\underline{h}_2^{(t)}}{\mathrm{d}\underline{\psi}} + R_3^*\,\frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\psi}} \tag{C.5}$$

According to equation (5.2),

$$R_3^*\,\frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\psi}} = R_{3,i_1}\,\frac{\mathrm{d}\underline{\hat{y}}^{(t-i_1)}}{\mathrm{d}\underline{\psi}} + \cdots + R_{3,i_m}\,\frac{\mathrm{d}\underline{\hat{y}}^{(t-i_m)}}{\mathrm{d}\underline{\psi}} + M_3\,\frac{\mathrm{d}\underline{\hat{y}}^{(t)}[n]}{\mathrm{d}\underline{\psi}} \tag{C.6}$$

where $\{i_1, \ldots, i_m\}$ are suitable indexes in a set $\mathcal{R}_3$, that express the considered time-lags in the recurrent connections for that layer. The term accounting for the average, instead, can be easily treated considering the linearity of the derivative operator. Thus the last Jacobian matrix of the previous equation can be written as

$$\frac{\mathrm{d}\underline{\hat{y}}^{(t)}[n]}{\mathrm{d}\psi} = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}\underline{\hat{y}}^{(t-i)}}{\mathrm{d}\underline{\psi}}$$

Summing up, the Jacobian matrix of the output $\underline{\hat{y}}^{(t)}$ with respect to the vector of parameters $\underline{\psi}$ is given by

$$\frac{\mathrm{d}\underline{\hat{y}}^{(t)}}{\mathrm{d}\underline{\psi}} = \frac{\mathrm{d}\tilde{K}_3}{\mathrm{d}\underline{\psi}}\,\underline{\tilde{h}}_2^{(t)} + K_3\,\frac{\mathrm{d}\underline{h}_2^{(t)}}{\mathrm{d}\underline{\psi}} + R_3^*\,\frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\psi}} \tag{C.7}$$

where the first term of the RHS is a sparse matrix and the third is a linear combination of the previous Jacobian matrices. The second instead has to be expressed by developing an analogous expression for the Jacobian of the hidden state $\underline{h}_2^{(t)}$ with respect to $\underline{\psi}$. It is not difficult to convince ourselves that similarly

$$\frac{\mathrm{d}\underline{h}_2^{(t)}}{\mathrm{d}\underline{\psi}} = \mathcal{A}_2'\Big(\,K_2\,\frac{\mathrm{d}\underline{h}_1^{(t)}}{\mathrm{d}\underline{\psi}} + R_2^*\,\frac{\mathrm{d}\underline{y}_2^{*(t)}}{\mathrm{d}\underline{\psi}}\,\Big) \tag{C.8}$$

and that an analogous expression holds for $\underline{h}_1^{(t)}$

$$\frac{\mathrm{d}\underline{h}_1^{(t)}}{\mathrm{d}\underline{\psi}} = \mathcal{A}_1' \Big(K_1 \frac{\mathrm{d}\underline{x}^{(t)}}{\mathrm{d}\underline{\psi}} + R_1^* \frac{\mathrm{d}\underline{y}_1^{*(t)}}{\mathrm{d}\underline{\psi}} \Big) = \mathcal{A}_1' R_1^* \frac{\mathrm{d}\underline{y}_1^{*(t)}}{\mathrm{d}\underline{\psi}} \tag{C.9}$$

since the input $\underline{x}^{(t)}$ is exogenous and thus does not depend on $\underline{\psi}$. To conclude,

$$\frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\psi}} = \frac{\mathrm{d}\tilde{K}_3}{\mathrm{d}\underline{\psi}} \tilde{\underline{h}}_2^{(t)} + R_3^* \frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\psi}} + K_3 \mathcal{A}_2' \left(R_2^* \frac{\mathrm{d}\underline{y}_2^{*(t)}}{\mathrm{d}\underline{\psi}} + K_2 \mathcal{A}_1' R_1^* \frac{\mathrm{d}\underline{y}_1^{*(t)}}{\mathrm{d}\underline{\psi}} \right) \tag{C.10}$$

In other words, the main result of this discussion is that the Jacobian matrix $\mathrm{d}\hat{\underline{y}}^{(t)} \Big/ \mathrm{d}\underline{\psi}$ is a function of the previously computed Jacobian matrices $\mathrm{d}\hat{\underline{y}}^{(t-k)} \Big/ \mathrm{d}\underline{\psi}$.

Two similar equations can be deduced also for the other two vectors of weights $\underline{\theta}$ and $\underline{\varphi}$, obtaining

$$\frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\varphi}} = R_3^* \frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\varphi}} + K_3 \mathcal{A}_2' \left(\frac{\mathrm{d}\tilde{K}_2}{\mathrm{d}\underline{\varphi}} \tilde{\underline{h}}_1^{(t)} + R_2^* \frac{\mathrm{d}\underline{y}_2^{*(t)}}{\mathrm{d}\underline{\varphi}} + K_2 \mathcal{A}_1' R_1^* \frac{\mathrm{d}\underline{y}_1^{*(t)}}{\mathrm{d}\underline{\varphi}} \right) \tag{C.11}$$

$$\frac{\mathrm{d}\hat{\underline{y}}^{(t)}}{\mathrm{d}\underline{\theta}} = R_3^* \frac{\mathrm{d}\underline{y}_3^{*(t)}}{\mathrm{d}\underline{\theta}} + K_3 \mathcal{A}_2' \left(R_2^* \frac{\mathrm{d}\underline{y}_2^{*(t)}}{\mathrm{d}\underline{\theta}} + K_2 \mathcal{A}_1' \left(\frac{\mathrm{d}\tilde{K}_1}{\mathrm{d}\underline{\theta}} \tilde{\underline{x}}^{(t)} + R_1^* \frac{\mathrm{d}\underline{y}_1^{*(t)}}{\mathrm{d}\underline{\theta}} \right) \right) \tag{C.12}$$

Implementing these equations it is possible to train the D-NAX network with RTRL.

Bibliography

1. Aggarwal, C. C. (2018). *Neural networks and deep learning: A textbook*. Springer.

2. Agrawal, R. K., Muchahary, F. & Tripathi, M. M. (2018). Long term load forecasting with hourly predictions based on long-short-term-memory networks. In *2018 IEEE Texas Power and Energy Conference (TPEC)* (pp. 1–6).

3. Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M. & Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. arXiv: 1801.07648

4. Azzone, M. & Baviera, R. (2021). Neural network middle-term probabilistic forecasting of daily power consumption. *Journal of Energy Markets*, *Early online*.

5. Barron, A. R. (1994). Approximation and estimation bounds for artificial neural networks. *Machine Learning*, *14*(1), 115–133.

6. Baviera, R. & Messuti, G. (2020). Daily middle-term probabilistic forecasting of power consumption in north-east England. arXiv: 2005.13005

7. Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In G. Montavon, G. B. Orr & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade: Second edition* (pp. 437–478). Springer.

8. Bengio, Y., Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

9. Benth, F. E., Benth, J. S. & Koekebakker, S. (2008). *Stochastic modeling of electricity and related markets*. World Scientific.

10. Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A. & Jenssen, R. (2017). *Recurrent neural networks for short-term load forecasting - an overview and comparative analysis*. Springer Briefs in Computer Science. Springer.

11. Billings, S. (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons.

12. Bloomfield, P. (2013). *Fourier analysis of time series*. John Wiley & Sons.

13. Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J. & Dahl, G. E. (2020). On empirical comparisons of optimizers for deep learning. arXiv: 1910.05446

14. Chollet, F. et al. (2015). Keras. https://keras.io.

15. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 303–314.

16. Du, S., Li, T., Yang, Y. & Horng, S.-J. (2020). Multivariate time series forecasting via attention-based encoder–decoder framework. *Neurocomputing*, *388*, 269–279.

17. Fahad, M. & Arbab, N. (2014). Factor affecting short term load forecasting. *Journal of Clean Energy Technologies*, *2*, 305–309.

18. Feilat, E., Al-Sha'abi, D. & Momani, M. (2017). Long-term load forecasting using neural network approach for Jordan's power system. *Engineering Press*, *1*, 43–50.

19. Forcada, M. L. & Carrasco, R. C. (1995). Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural Computation*, *7*(5), 923–930.

20. Gasparin, A., Lukovic, S. & Alippi, C. (2019). Deep learning for time series forecasting: The electric load case. arXiv: 1907.09207

21. Gers, F., Schmidhuber, J. & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation*, *12*, 2451–71.

22. Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, *9*, 249–256.

23. Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT Press.

24. Guerini, A. (2016). *Long and short term forecasting of daily and quarter-hourly electrical load and price data: A torus-based approach* (Doctoral dissertation, Università di Pavia, Pavia).

25. Guerini, A. & De Nicolao, G. (2015). Long-term electric load forecasting: A torus-based approach. In *2015 European Control Conference (ECC)* (pp. 2768–2773).

26. He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778).

27. Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*, 107–116.

28. Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780.

29. Hong, T. (2010). *Short term electric load forecasting* (Doctoral dissertation, North Carolina State University, Charlotte, NC).

30. Hong, T. & Fan, S. (2016). Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, *32*(3), 914–938.

31. Hong, T., Xie, J. & Black, J. (2019). Global energy forecasting competition 2017: Hierarchical probabilistic load forecasting. *International Journal of Forecasting*, *35*(4), 1389–1399.

32. Hyndman, R. (2020). A brief history of forecasting competitions. *International Journal of Forecasting*, *36*(1), 7–14.

33. Hyndman, R. & Fan, S. (2010). Density forecasting for long-term peak electricity demand. *Power Systems, IEEE Transactions on*, *25*, 1142–1153.

34. IEA. (2020). *World Energy Outlook 2020*. Paris. Retrieved from https://www.iea.org/reports/world-energy-outlook-2020

35. Kandil, M. S., El-Debeiky, S. M. & Hasanien, N. E. (2002). Long-term load forecasting for fast-developing utility using a knowledge-based expert system. *IEEE Power Engineering Review*, *22*(4), 78–78.

36. Kapoor, A., Guili, A. & Pal, S. (2019). *Deep learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API, 2nd Edition*. Packt Publishing, Ltd.

37. Karita, S., Wang, X., Watanabe, S., Yoshimura, T., Zhang, W., Chen, N., Hayashi, T., Hori, T., Inaguma, H., Jiang, Z., Someki, M., Yalta, N. & Yamamoto, R. (2019). A comparative study on Transformer vs RNN in speech applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)* (pp. 449–456).

38. Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv: 1412.6980

39. Mahsereci, M., Balles, L., Lassner, C. & Hennig, P. (2017). Early stopping without a validation set. arXiv: 1703.09580

40. McCulloch, W. & Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 127–147.

41. Menezes, J. M. P. & Barreto, G. A. (2008). Long-term time series prediction with the NARX network: An empirical evaluation. *Neurocomputing*, *71*(16), 3335–3343.

42. Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K. & Graves, A. (2020). A practical sparse approximation for real time recurrent learning. arXiv: 2006.07232

43. Miller, J. & Hardt, M. (2019). Stable recurrent models. arXiv: 1805.10369

44. Nawi, N. M., Atomi, W. H. & Rehman, M. (2013). The effect of data preprocessing on optimized training of artificial neural networks. *Procedia Technology*, *11*, 32–39.

45. Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2). *Doklady Akademii Nauk USSR*, *269*, 543–547.

46. Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (Vol. 28, *3*, pp. 1310–1318). Proceedings of Machine Learning Research. Atlanta, Georgia, USA: PMLR.

47. Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, *8*, 143–195.

48. Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151.

49. Rakitianskaia, A. & Engelbrecht, A. (2015). Measuring saturation in neural networks. In *2015 IEEE symposium series on computational intelligence* (pp. 1423–1430).

50. Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for activation functions. arXiv: 1710.05941

51. Ramanathan, R., Engle, R., Granger, C., Vahid, F. & Brace, C. (1997). Short-run forecasts of electricity loads and peaks. *International Journal of Forecasting*, *13*(2), 161–174.

52. Reneses, J., Centeno, E. & Barquin, J. (2006). Coordination between medium-term generation planning and short-term operation in electricity markets. *IEEE Transactions on Power Systems*, *21*(1), 43–52.

53. Riedmiller, M. A. & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *IEEE International Conference on Neural Networks*, 586–591 vol.1.

54. Roach, C. (2019). Reconciled boosted models for GEFCom2017 hierarchical probabilistic load forecasting. *International Journal of Forecasting*, *35*(4), 1439–1450.

55. Rodrigues, F., Cardeira, C. & Calado, J. (2014). The daily and hourly energy consumption and load forecasting using artificial neural network method: A case study using a set of 93 households in Portugal. *Energy Procedia*, *62*, 220–229.

56. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408.

57. Rumelhart, D., Hinton, G. & Mcclelland, J. (1986). A general framework for parallel distributed processing. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, *1*.

58. Schmidhuber, J. (1992a). A Fixed Size Storage O(n3) Time Complexity Learning Algorithm for fully recurrent continually running networks. *Neural Computation*, *4*(2), 243–248.

59. Schmidhuber, J. (1992b). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, *4*(2), 234–242.

60. Smyl, S. & Hua, N. G. (2019). Machine learning methods for GEFCom2017 probabilistic load forecasting. *International Journal of Forecasting*, *35*(4), 1424–1431.

61. Soares, L. J. & Souza, L. R. (2006). Forecasting electricity demand using generalized long memory. *International Journal of Forecasting*, *22*(1), 17–28.

62. Srivastava, R. K., Greff, K. & Schmidhuber, J. (2015). Highway networks. arXiv: 1505.00387

63. Stegemann, J. A. & Buenfeld, N. (1999). A glossary of basic neural network terminology for regression problems. *Neural Computing and Applications*, *8*, 290–296.

64. Sutskever, I. (2013). *Training recurrent neural networks* (Doctoral dissertation, University of Toronto, Toronto, Canada).

65. Tallec, C. & Ollivier, Y. (2017). Unbiased online recurrent optimization. arXiv: 1702.05043

66. Tien, D. (2003). Common mistakes in neural networks training. In D. Feng & E. Carson (Eds.), *5th IFAC symposium on modelling and control in biomedical systems* (pp. 383–386).

67. van den Oord, A., Kalchbrenner, N. & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. arXiv: 1601.06759

68. Vanegas Cantarero, M. M. (2020). Of renewable energy, energy democracy, and sustainable development: A roadmap to accelerate the energy transition in developing countries. *Energy Research & Social Science*, *70*, 101716.

69. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is all you need. arXiv: 1706.03762

70. Vossen, J., Feron, B. & Monti, A. (2018). Probabilistic forecasting of household electrical load using artificial neural networks. In *International conference on probabilistic methods applied to power systems (PMAPS) : June 24-28, 2018, Boise, Idaho, USA : Conference proceedings / IEEE*. International Conference on Probabilistic Methods Applied to Power Systems, Boise, Idaho (USA), 24 Jun 2018 - 28 Jun 2018. Piscataway, NJ: IEEE.

71. Wang, H. & Raj, B. (2017). On the origin of deep learning. arXiv: 1702.07800

72. Werbos, P. J. (1994). *The roots of backpropagation: From ordered derivatives to neural networks and political forecasting*. John Wiley & Sons.

73. Weron, R. (2014). Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, *30*(4), 1030–1081.

74. Williams, R. J. & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, *2*(4), 490–501.

75. Williams, R. J. & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280.

76. Williams, R. J. & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin & D. Rumelhart (Eds.), *Developments in connectionist theory. backpropagation: Theory, architectures, and applications* (pp. 433–486). Lawrence Erlbaum Associates, Inc.

77. Winkler, R. L. (1972). A decision-theoretic approach to interval estimation. *Journal of the American Statistical Association*, *67*(337), 187–191.

78. Xie, J. & Hong, T. (2018). Temperature scenario generation for probabilistic load forecasting. *IEEE Transactions on Smart Grid*, *9*(3), 1680–1687.

79. Yam, J. Y. & Chow, T. W. (2000). A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, *30*(1), 219–232.

80. Ziel, F. (2018). Modeling public holidays in load forecasting: A German case study. *Journal of Modern Power Systems and Clean Energy*, *6*(2), 191–207.

81. Ziel, F. (2019). Quantile regression for the qualifying match of GEFCom2017 probabilistic load forecasting. *International Journal of Forecasting*, *35*(4), 1400–1408.