Bergische Universität Wuppertal

Fakultät für Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 20/54

Emma Viviani, Luca Di Persio and Matthias Ehrhardt

Energy markets forecasting. From Inferential Statistics to Machine Learning: the German case

November 24, 2020

http://www.imacm.uni-wuppertal.de

MDPI

Article

Energy markets forecasting. From Inferential Statistics to Machine Learning: the German case

Emma Viviani [1,*,†], Luca Di Persio [2,†] and Matthias Ehrhardt [3,†]

[1] College of Mathematics, Department of Computer Science, University of Verona, Verona, Italy; emma.viviani@studenti.univr.it
[2] College of Mathematics, Department of Computer Science, University of Verona, Verona, Italy; luca.dipersio@univr.it
[3] Chair of Applied Mathematics and Numerical Analysis, University of Wuppertal, Wuppertal, Germany; ehrhardt@uni-wuppertal.de
* Correspondence: emma.viviani@studenti.univr.it
‡ These authors contributed equally to this work.

Version November 24, 2020 submitted to Energies

Abstract: In this work we investigate a probabilistic method for electricity price forecasting that overcomes traditional forecasting methods. We will start from a statistical method for a point forecast. We compare these approaches in terms of efficiency, accuracy and reliability. Last but not least, we also aim to compare the previously cited results with those obtained using neural networks, with the ultimate goal of developing hybrid solutions for a more general set of electricity forecasting tasks.

Keywords: electricity price; statistical method; autoregressive; probabilistic forecast; neural network

1. Introduction

The electricity market has always aroused the interest of many people because of the importance of its product. One can find the beginnings of electricity price forecasting in the early 90s. We must remember that electricity is a special commodity. It cannot be stored economically and therefore requires a constant balance between production and consumption; it depends on weather conditions, the season and the nature of human activity. These characteristics are unique and lead to a price dynamic that is not found on any other market. It is often possible to recognize a day, weekly or yearly seasonality and possible sudden, short-term and generally unforeseen price peaks. Moreover, the presence of possible cycles characterizing this product can be characterized by irregularities, being then difficult to be properly modelled.

Electricity Price Forecasting (EPF) is a branch of energy forecasting that focuses on predicting spot and forward prices on the wholesale electricity markets. It has become a fundamental input for energy companies in their decision-making and investment process. An important aspect in many areas of finance is the time horizon. In the literature, there are usually three types of forecasting horizons: short, medium and long-term. *Short-term* generally means a period of time ranging from few minutes to few days in advance; *medium-term* means a period of time ranging from few days to few months; and *long-term* means a period of time ranging from some months to years. In most cases, the forecast horizons are of the first two types, with particular emphasis on *day ahead* previsions. Nevertheless, it is worth underlying that each of the above mentioned horizons has its own importance, a company being obliged to program its activities at different time scales. Within such a scenario, different modeling approaches can be exploited. In particular, mainly used techniques belong to the following five groups: statistical models, intelligent computing models, reduced form, fundamental and multi-agent models. In this paper we will start focusing on the first two families, to then combine them as to end up with a hybrid proposal which represents the major novelty of the present paper.

Furthermore, we can also classify the type of prediction. For point forecasts the goal is to predict the *energy price* at a given hour or day, usually within the short time horizon framework. To this end, a high accuracy level is mandatory. During recent years, previous task lost in relevance in favour of so called *probabilistic predictions*. based on intervals forecasting w.r.t. the determination of related quantiles or even aiming at predicting the characterizing entire *probability distribution*. Next we will look at all the EPF's aforementioned aspects in more detail and then try to apply some models to a specific data set. It contains the average daily prices of the German electricity market from 2016 to 2018.

This paper is structured as follows: Section 2 briefly describes the possible types of forecasts and the main characteristics of the German electricity market. Section 3 explains the special case of a probabilistic forecast. Section 4 presents in detail the models we will use for data analysis, especially about the SARIMA model (Seasonality AutoRegressive Integrate Moving Average) and the methods of artificial neural networks (ANN). The last chapter deals with the implementation of these models and the analysis of the data available to us. At the end we give some conclusions and critical remarks.

2. General overview

Price forecasts are a fundamental topic for making decisions and determining a possible market strategy for an industry or company. Especially in the electricity price forecasting (EPF) literature, we can distinguish three types of predictions based on this term. First, we can speak of short-term horizons, ranging from minutes to days, horizons for which reliable meteorological forecasts for temperature, wind speed, cloud cover, etc. are available. Short-term forecasts are mainly relevant for market operations, intraday trading and system stability, see [1].

Medium-term forecasts cover horizons beyond reliable meteorological predictions, but without the major impact of political and technological uncertainty, with lead times measured in weeks, months or even years. The practical relevance results mainly from maintenance planning, reallocation of resources, bilateral contracts, valuation of derivatives, risk management and budgeting. After all, long-term horizons refer to everything from a few years to several decades. This type of forecasting is needed, for example, to address problems as an investment in the future, see [2].

Price forecasting is essential for the energy market with its special features. It is typically a day-ahead market that does not allow continuous trading. Agents place their bids and offers for electricity deliveries during each hour of the next day before a certain market close time. Then prices are determined for all load periods of the next day at the same time during a unit price auction, [1]. In electricity markets with zonal price formation, as in Europe, there are day-ahead markets and intraday markets. These markets start operating after the announcement of the results of the day-ahead auction and run until a few minutes before delivery. They attempt to compensate for deviations resulting from positions in day-ahead contracts and unexpected changes in demand or supply.

2.1. Type of forecasts

Time series forecasting is the prediction of system behavior in the future, based on information about the current and past status of the phenomena we observe. There are different types of prediction approaches, and the right choice depends on the goal to be achieved.

Point forecast

The day-ahead price series is usually the result of an auction held once a day, in which all hourly prices for the next day are announced at once. In the intraday markets, load periods can be shorter than one hour, e.g. at the European Power Exchange (EPEX), half-hourly and quarter-hourly products are also traded. In both cases the day can be divided into a finite number of load periods $h = 1, \dots, H$. Therefore it is obvious to use $P_{d,h}$ to denote the price for the day d and the load period h. A point forecast of $P_{d,h}$ denoted by $\hat{P}_{d,h}$ is usually understood in the EPF literature as the expected value of the price random variable, i.e. $\mathbb{E}(P_{d,h})$. This term can easily be extended to quantile forecasts, which are a possible starting point for probabilistic predictions.

Probabilistic forecast

There are two main approaches for probabilistic predictions. The more popular one is based on point prediction and the associated error distribution. The other approach directly considers the distribution of the spot price and is used for example in Quantile Regression Average (QRA), see Nowotarski and Weron [3]. In both cases, the focus can be on prediction intervals, selected quantiles or the overall predictive distribution. Let us consider the mean price at a future point in time, i.e. $\hat{P}_{d,h} = \mathbb{E}(P_{d,h})$, as a "point prediction", then we can write $P_{d,h} = \hat{P}_{d,h} + \epsilon_{d,h}$, which implies:

$$F_P(x) = F_\epsilon(x - \hat{P}_{d,h}), \tag{1}$$

where F_ϵ is the distribution of errors associated with $\hat{P}_{d,h}$. This means that the distribution of errors has an identical form as the distribution of prices, only it is shifted to the left by $\hat{P}_{d,h}$ on the horizontal axis. The corresponding quantile function $\hat{q}_{\alpha,\epsilon}$ is also shifted with respect to $\hat{q}_{\alpha,P}$. Equivalent in the sense of the inverse empirical cumulative distribution function (CDF) one can write: $\hat{F}_P^{-1}(\alpha) = \hat{P}_{d,h} + \hat{F}_\epsilon^{-1}(\alpha)$. They also form the basic framework for the generation of probabilistic predictions from distributions of prediction errors. The choice of a probabilistic prediction can be useful, for example, if you need some information for a huge investment in the future.

Ensemble forecast

The probabilistic prediction concept is very general, sometimes it is not sufficient to solve many problems in energy prediction and it might be difficult to verify its validity. The reason for this is that $\hat{P}_{d,h}$ is considered on its own, independent of the predictions for the neighboring hours. However, instead of considering the $\mathcal{H}$ univariate price distributions, we should focus on the $\mathcal{H}$-dimensional distribution $\mathbf{F_P}$ of the $\mathcal{H}$-dimensional price vector $\mathbf{P_d} = (P_{d,1}, \ldots, P_{d,\mathcal{H}})$. We therefore need a prediction for the multivariate distribution. Unfortunately, many models cannot provide such a direct distribution forecast. The solution to the latter problem could be to compute an ensemble forecast. An ensemble is defined as a collection of M paths $\mathcal{E}_M(\hat{\mathbf{P}}_{\mathbf{d,h}}) = (\hat{\mathbf{P}}_{\mathbf{d}}^{\mathbf{1}}, \ldots, \hat{\mathbf{P}}_{\mathbf{d}}^{\mathbf{M}})$ simulated from a forecast model, typically using Monte Carlo methods.

2.2. Modelling Approaches

Usually the techniques for electricity price forecasting (EPF) are divided into five groups of models: *statistically, computationally intelligent, reduced form, fundamental and multi-agent*. We will focus on statistical and computational intelligent (CI) models.

2.2.1. Statistical Approaches

Statistical approaches forecast the current price by a weighted combination of past prices and/or current values of exogenous variables that could be associated with the electricity (e.g. demand or weather forecasts), typically in a linear regression. Autoregressive terms take into account the dependencies between today's prices and those of the previous days, e.g. could be a possible structure:

$$\begin{aligned} P_{d,h} = {} & \beta_{h,0} + \beta_{h,1} P_{d-1,h} + \beta_{h,2} P_{d-2,h} + \beta_{h,3} P_{d-7,h} \\ & + \beta_{h,4} P_{d-1,min} + \beta_{h,5} L_{d,h} \\ & + \beta_{h,6} D_{sat} + \beta_{h,7} D_{sun} + \beta_{h,8} D_{mon} + \epsilon_{d,h}, \end{aligned} \tag{2}$$

where $P_{d,h}$ denotes the price for day d and hour h, $P_{d-1,min}$ is the minimum of the 24-hour prices of the previous day (example of a non-linear component), $L_{d,h}$ refers to the load forecast for day d and hour h (known on day *d-1*), and the three dummies $(D_{sat}, D_{sun}, D_{mon})$ model the weekly seasonality. Some authors refer to such parsimonious structures as *expert models*, because they are based on a certain amount of prior knowledge of experts.

The standard approach for estimating the model (2) is Ordinary Least Squares (OLS). The procedure uses the electricity prices of the past $\mathcal{D}$ days to predict the prices for the following day(s). The optimal value $\mathcal{D}$ is not given a priori, so it should be chosen so that the estimation sample is "long enough" to extract samples, but not "too long" to give too much weight to the distant past. Overall, there are no standard parameters. Many studies assume one or two years with hourly prices, but some use 10-13 days as short time windows, while others are up to four years long, see Marcjasz et al. [4].

While autoregression models represent the largest subset of statistical models, this class includes

- similar-day methods, like the naive method which sets $\hat{P}_{d,h} = P_{d-7,h}$ for Monday, Saturday or Sunday, and $\hat{P}_{d,h} = P_{d-1,h}$ otherwise,
- the generalized autoregressive conditional heteroskedasticity (GARCH) models, typically in connection with volatility forecasting,
- shrinkage techniques, as the least absolute shrinkage and selection operator (LASSO) but also Ridge regression and elastic nets.

Statistical models are attractive because physical interpretations can be added to the regressors, can therefore be useful to identify significant variables and it allows operators to better understand their behavior and obtain a model with a meaningful structure. A well-known disadvantage of statistical models is the representation of nonlinear factors, even if they can be approximated by linear ones under certain conditions. Nevertheless, non-linear dependencies can be included explicitly by non-linear variables, like $P_{d-1,min}$ in (2). Alternatively, the spot prices for electricity (as well as exogenous variables) can be transformed using nonlinear functions before fitting a statistical model.

2.2.2. Computational Intelligence Methods (CI)

CI methods are a completely different group of methods that were developed to solve problems that cannot be treated efficiently with statistical methods, for example non-linearity or when a distribution-free approach is required. They combine different elements to create approaches which are able to adapt to complex dynamic systems, i.e. they do not need exact knowledge, but they can also be confronted with incompleteness. CI models are flexible and can deal with different types of nonlinearity, but at the cost of high computational complexity. Nevertheless, they are used for short-term predictions. Statistics is more concerned with the analysis of finite samples, misspecification of models and computational considerations, while probabilistic modeling is inherent in computational intelligence. There are several ANN algorithms; in general, one can mainly focus on three aspects to identify a specific ANN algorithm for a prediction task: the complexity of the solution, the desired prediction accuracy and the data characteristics, see [5].

To summarize the conceptual difference between these two approaches, we can observe Figure 1. Computational intelligence works more like a "black box" in which we can decide only a few parameters; each time we train a network, we can get similar but slightly different results; the statistical approach on the right models a certain relationship between the data.

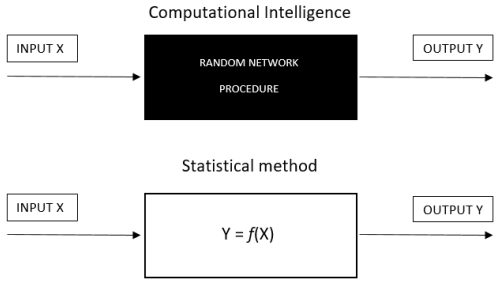


Figure 1. The CI method and statistical approaches.

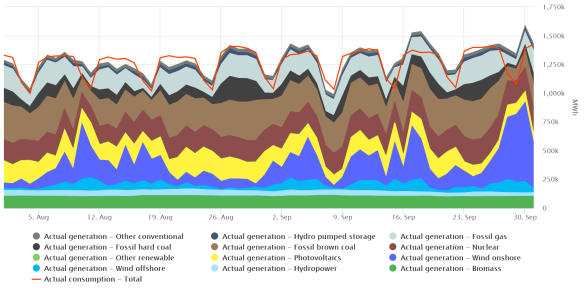


Figure 2. Electricity generation and consumption in August and September 2019. It shows the total electricity generation and consumption on each day in the period, see [7] (CC BY 4.0 license).

2.3. The German electricity market

We will use price data from the German electricity market. The EPEX is the most important trading platform for electricity prices in Europe, it is the most important in terms of volume. It is also relevant from a policy point of view, since, for example, several regulatory calculations are based on the day-ahead EPEX price, such as the feed-in tariffs for renewable energies. It offers trading, clearing and settlement in both the day-ahead and intraday market. Day-ahead hourly prices in Germany are traded on EPEX and are referred to as "Phelix", Physical electricity index. This index is the daily mean value of the system price for electricity traded on the spot market, calculated as the arithmetic mean of the 24-hour market clearing price.

The day-ahead market is the primary market for electricity trading. Here, buyers and sellers set up hourly contracts for the delivery of electricity the following day. This is done through a daily auction, usually at 12:00, where the market clearing price, commonly known as the spot price, is determined by matching supply and demand. Most market designs choose a uniform-price auction market (see [6]): buyers with bids above the clearing price pay that price, and suppliers with bids below that price receive the same price.

The demand for electricity is a function of temperature, seasonality and consumption patterns, from which the periodic character of electricity prices is derived. Since consumers have few options available to them in response to price changes, demand is very price inelastic in the short term. Positive price peaks are often caused by high (unexpected) demand. On the other hand, the merit-order curve plays a decisive role in the electricity price formation process. The merit order curve, also called supply curve, is derived by arranging the suppliers' offers according to rising marginal costs. The point of intersection of the demand curve with the merit order curve defines the market clearing price, i.e. the electricity spot market price.

In times of low demand, base load power plants such as nuclear power plants and coal-fired power plants generally serve as price-determining technologies. These plants are inflexible due to their high costs. In contrast, in times of high demand, the prices are set by expensive peak load power plants such as gas and oil-fired power plants. These plants have high flexibility, high marginal costs and lead to a convex shape of the merit order curve. With the lowest marginal costs, renewable energy sources are at the lower end of the merit order curve. Wind and solar energy have attracted the most attention in Germany, and regulatory changes are also the main driver for the growth of renewable energies, as several subsidies and political measures have been introduced recently. There is a large share of intermittent renewable energy sources, which makes the difficult task of forecasting and describing prices of changing energy markets. The hours of increased renewable energy supply are causing difficulties for inflexible plants that should be running continuously. This is because inflexible base-load plants have shutdown and start-up costs that force them to accept negative marginal returns in order to generate electricity continuously. This has a lowering effect on electricity prices. Negative prices are mainly caused by high wind production in times of low demand. Therefore negative price peaks occur mainly at night. For further details we refer the reader to [8].

3. The Probabilistic Approach

The probabilistic approach means a forecast in the form of a probability distribution of future quantities or events. It is possible to treat different types of problems and to extend the spectrum of the point estimation approach. For example, a long-term forecast becomes possible and feasible.

3.1. The Problem

For the definition of the probabilistic forecasting problem one can use a point forecast of the electricity spot price (i.e. the "best estimate" or expected value of the spot price), see [9]. Note that the actual price at t, P_t, can be expressed as:

$$P_t = \hat{P}_t + \epsilon_t, \tag{3}$$

where $\hat{P}_t$ is the point forecast of the spot price at time t, which was made at an earlier time, and ϵ_t is the corresponding error. In the vast majority of the EPF papers the analysis ends at this point because the authors focus only on point forecasts, see [1].

The most common extension from point to probabilistic forecasts is the construction of prediction intervals (PI). For this purpose, a number of methods can be used, the most popular of which consider both the point prediction and the corresponding error: the center of the PI at the $(1-\alpha)$ confidence level is set equal to $\hat{P}_t$ and its bounds are defined by the $\frac{\alpha}{2}$th and $1-\frac{\alpha}{2}$th quantile of the CDF of ϵ_t. For example, for the 90% PIs, the 5% and 95% quantiles of the error term are required.

A forecaster can further expand his study and construct multiple PIs. The final result can be a series of quantiles at many levels. Such a set of 99 quantiles ($q = 1\%, 2\%, \ldots, 99\%$) is also a reasonable discretization of the price distribution. In general, a density forecast corresponding to (3) can be defined as a set of PIs for all $\alpha \in (0,1)$. In other words, the calculation of a probabilistic prediction requires an estimate of $\hat{P}_t$ and the distribution of ϵ_t. Equivalently, the problem can be formulated in terms of the inversion of the CDF of P_t and ϵ_t:

$$F^{-1}_{P_t}(q) = \hat{P}_t + F^{-1}_{\epsilon_t}(q). \tag{4}$$

We can define the probabilistic forecast as a forecast in the form of a probability distribution over future quantities or events and assign them to a random variable. In our case, this means, the distribution of the electricity spot price itself, i.e. $\hat{F}_{P_t}$. The latter approach is used in QRA.

Now two important aspects of the problem have to be mentioned. First, we do not mention the Probability Density Function (PDF) of ϵ_t in the discussion above. The second point relates to the statistical nature of the prediction of day-ahead prices. Since 24-hour prediction distributions must be created at once, their interdependencies and their common distribution should be taken into account.

In [9] two main approaches are presented. The first is to model and predict the correlation between the boundary distributions. However, this has a major disadvantage: it allows to capture only linear relationships between hours and raises the question of how to evaluate them correctly. The second solution requires the simulation of 24-hour paths of day-ahead prices, which then can be treated as vectors from the joint 24-dimensional distribution (multivariate models).

3.2. Construction of Probabilistic Forecasts

There are several ways to construct a probabilistic interval. We report four of them from [9].

Historical simulation

The method for calculating empirical (or sample) PIs is simple and is called historical simulation in the Value-at-Risk (VAR) literature. It is a model-independent approach, which consists of the calculation of sample quantiles of the empirical distribution of one-step-ahead prediction errors, ϵ_t.

Distribution-based probabilistic predictions

For time series models driven by Gaussian noise (AR, ARIMA, etc.), the density forecasts can be set equal to the Gaussian distribution approximating the error density and the PIs can be calculated analytically as quantiles of this distribution. This approach differs from the historical simulation in that first the standard deviation of the error density, $\hat{\sigma}$, is calculated and then the lower and upper bounds of the PI are set equal to the selected quantiles of the $\mathcal{N}(0, \hat{\sigma}^2)$ distribution.

Bootstrapped PIs

The third approach often used in studies on neural networks EPF is the bootstrap. For a step-ahead forecast the method consists of the following steps:

1. Estimate the set of model parameters, $\hat{\theta}$, obtain a fit and the corresponding residuals, ϵ_t.
2. Generate pseudo-data recursively using $\hat{\theta}$ and sampled normalized residuals ϵ_t^*: – For a model with no autoregression on P_t (like the neural network model) simply set $P_t^* = \hat{f}(X_t) + \epsilon_t^*$ where ϵ_t^* is the sampled residual and $\hat{f}(X_t)$ is an estimated function of exogenous variables X_t. – For a more general case of an autoregressive model of order r with exogenous variables first set $P_1^* = P_1, \ldots, P_r^* = P_r$ and then recursively set:

$$P_t^* = \hat{\beta}_1 P_{t-1}^* + \cdots + \hat{\beta}_r P_{t-r}^* + \hat{f}(X_t) + \epsilon_t^*, \quad \text{for all} \quad t \in \{r+1, \ldots, T\}. \tag{5}$$

3. Estimate the model again and compute the bootstrap-implied one step-ahead (point) forecast for time $t = T + 1$.
4. Repeat steps 2 and 3 B times and obtain the bootstrap sample of the predicted price, $\{\hat{P}_{T+1}^i\}_{i=1}^B$.
5. Compute desired quantiles of $\{\hat{P}_{T+1}^i\}_{i=1}^B$ to obtain PIs.

On the one hand, this type of construction has the advantage that it takes into account not only the historical forecast errors but also the parameter uncertainty and is therefore more accurate. On the other hand, it is less convenient from a computational point of view because it requires more effort.

Quantile Regression Averaging

The QRA method, proposed by Nowotarski and Weron [3], involves applying a quantile regression to a pool of point forecasts of individual (i.e. not combined) forecast models. It works directly with the distribution of the electricity spot price, $\hat{F}_{P_t}$ without having to split the probabilistic forecast into a point forecast and the distribution of the error term. The quantile regression problem can be written as

$$Q_{P_t}(q|X_t) = X_t \beta_q, \tag{6}$$

where $Q_{P_t}(q|X_t)$ is the conditional q-th quantile of the electricity price distribution, X_t are the explanatory variables (or regressors) and β_q is a vector of parameters for the q quantile. The parameters are estimated by minimizing the loss function for a given q-th quantile. There is no limitation of the components of X_t. As long as it includes forecasts of individual models, it is considered QRA.

3.3. Validity

In case of a probabilistic forecast, the most important problem is that the true distribution of the underlying process is not known. We cannot compare the predictive distribution with the actual distribution of the electricity spot price only with observed prices in the past. There are several ways to evaluate probabilistic forecasts and the approach depends on the final intention. At this point we need some tests and parameters to check the validity of the model and to have criteria for choosing the optimal model. An evaluation is usually based on reliability, sharpness and resolution.

The *reliability* (also called calibration or unbiasedness) refers to the statistical consistency between the distributional forecasts and the observations. For example, if a 90% PI covers 90% of the observed prices, then this PI is considered reliable, well calibrated, or unbiased.

Sharpness, on the other hand, refers to how closely the predicted distribution covers the actual distribution, i.e. the concentration of the predicted distributions. Unlike reliability, which is a joint property of predictions and observations, sharpness is a property of the forecasts only.

Then *resolution* refers to how strongly the predicted density varies over time, in other words, to the ability to provide probabilistic forecasts (e.g. wind power) depending on the forecast conditions (e.g. wind direction).

To formally check whether there is an "unconditional coverage" (UC), i.e. whether $\mathbb{P}(I_{d,h} = 1) = (1 - \alpha)$ where $I_{d,h} = 1$ if $P_{d,h}$ is in the interval, we can use the approach of Kupiec (1995), which checks whether $I_{d,h}$, also known as an indicator for "hits and misses", is i.i.d. Bernoulli with an average of $(1 - \alpha)$, i.e. violations are assumed to be independent. Since the Kupiec test is not based on the order of the PI violations, but only on the total number of violations, Christoffersen (1998) introduced the independence test and the conditional coverage test (CC).

Testing for the goodness-of-fit of a predictive distribution is generally more difficult than assessing the reliability of a PI. The most common approach is to use the probability integral transform (PIT)

$$PIT_{d,h} = \hat{F}_P(P_{d,h}). \tag{7}$$

If the distribution forecast matches the true distribution of the spot price process, then $PIT_{d,h}$ is independent and uniformly distributed, which can be shown with a formal statistical test, see [10].

In contrast to reliability, which is a joint property of predictions and observations, sharpness is only one property of predictions. Sharpness is closely linked to the concept of correct scoring rules. Indeed, the scoring rules evaluate reliability and sharpness simultaneously, [10]. The pinball loss (PL) for quantile predictions and the continuous ranked probability score (CRPS) for distribution predictions are the two most popular correct valuation rules for energy forecasting. The pinball loss (PL) is a special case of an asymmetric piecewise linear loss function:

$$PL(\hat{Q}_{P_{d,h}}(\alpha), P_{d,h}, \alpha) = \begin{cases} (1-\alpha)\,(\hat{Q}_{P_{d,h}}(\alpha) - P_{d,h}) & \text{for } P_{d,h} < \hat{Q}_{P_{d,h}}(\alpha) \\ \alpha\,(\hat{Q}_{P_{d,h}}(\alpha) - P_{d,h}) & \text{for } P_{d,h} > \hat{Q}_{P_{d,h}}(\alpha) \end{cases} \tag{8}$$

so PL depends on the quantile function and the actually observed price. The PL is a strictly correct score for the α-th quantile. To get an aggregated score, the PL can be averaged over different quantiles.

It is also necessary to have statistically significant conclusions about the outperformance of the forecasts of one model by those of another model. For this purpose we use the Diebold Mariano (DM) test, which is an asymptotic z-test of the hypothesis that the mean value of the loss differential series:

$$\delta_{d,h} = S_1(\hat{F}_{P_t}, P_t) - S_2(\hat{F}_{P_t}, P_t) \tag{9}$$

is zero, where $S_i(*,*)$ is the score of the forecasts of the model $(i = 1, 2)$. In the context of probabilistic or ensemble forecasts, any strictly correct scoring rule can be used, for example the pinball loss. Given the loss difference series, we calculate the statistics:

$$DM = \sqrt{T}\,\frac{\hat{\mu}(\delta_{d,h})}{\hat{\sigma}(\delta_{d,h})}, \tag{10}$$

where $\hat{\mu}(\delta_{d,h})$ and $\hat{\sigma}(\delta_{d,h})$ is the sample mean or standard deviation of $\delta_{d,h}$ and T is the length of the test period outside the sample. The key hypothesis of equal predictive accuracy (i.e. equal expected loss) corresponds to $\mathbb{E}(\delta_{d,h}) = 0$, in which case, assuming a steady-state covariance of $\delta_{d,h}$, the DM statistic is asymptotically standard normal and one- or two-sided asymptotic tail probabilities are easily calculated. The DM test compares the forecasts of two models, not the models themselves.

In day-ahead power markets, forecasts for all 24 hours of the next day are made at the same time with the same information, so forecast errors for a given day usually have a high serial correlation. It is therefore advisable to run the DM tests separately for each load period (e.g. each hour of the day) [11].

4. Models

We now present the theory behind some possible models for forecasting the energy price. We will see two statistical models and then one from the group of computational intelligence.

4.1. (S)ARIMA Models

Auto Regressive Moving Average (ARMA) models represent an important class of statistical models for analyzing and forecasting time series data. The autoregressive component uses the dependencies between observations and a given number of delayed observations; the moving average component uses the dependency between an observation and a residual error from a moving average model applied to delayed observations. This type of model relates the signal to its own past and does not explicitly use information from other possible related time series. An ARMA(p, q) is defined as

$$\phi(B)\, X_t = \theta(B)\, \epsilon_t, \tag{11}$$

where B denotes the backward shift operator, i.e. $B^h = x_{t-h}$, $\phi(B)$ is the notation for the polynomial of the autoregressive component

$$\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p,$$

$\theta(B)$ for the polynomial of the moving average component

$$\theta(B) = 1 + \theta_1 B + \cdots + \theta_p B^p,$$

and ϵ_t denotes a white noise $WN(0, \sigma^2)$. Thus p and q are the orders of the two polynomials. The observed time series is considered as realization of a stochastic process, whose parameters are the coefficients of the polynomials of the operator B, which determine the properties of persistence (and also the variance of the white noise). This kind of model assumes that the time series is stationary, i.e. mean and covariance of the time series shall be time independent. Usually, the series has to be transformed into the stationary form by differentiation. A model that explicitly includes differentiation is a generalization of the ARMA model for non-stationary time series: the "Auto Regressive Integrated Moving Average" (ARIMA) models. The equation of a ARIMA(p, d, q) is given by

$$\phi(B)\, \nabla^d X_t = \theta(B)\, \epsilon_t, \tag{12}$$

where $\nabla x_t \equiv (1 - B)x_t$ is the lag 1 differencing operator, a particular case of the general lag-h differencing operator given by

$$\nabla^h x_t \equiv (1 - B)^h x_t. \tag{13}$$

If $d = 0$, ARIMA$(p, 0, q) \equiv$ ARMA(p, q), i.e. ARIMA processes reduce to ARMA processes when differenced finitely many times.

Seasonality fluctuations are indeed caused by changing climatic conditions that influence demand and supply. For this reason, we introduce a model that can handle this behavior: the Seasonal Autoregressive Integrated Moving Average (SARIMA) process. The notation for a SARIMA model with both seasonal and non-seasonal parameters is ARIMA$(p, d, q) \times (P, D, Q)_s$. Here (p, d, q) indicates the order of the non-seasonal part, while $(P, D, Q)_s$ is the seasonal part. The parameter s represents the number of observations in the seasonal pattern, e.g. for monthly data we would have $s = 12$, for quarterly observations $s = 4$, etc.. The SARIMA model is defined by the following formula:

$$\phi(B)\Phi(B^s)\nabla^d \nabla_s^D X_t = \theta(B)\Theta(B^s)\epsilon_t. \tag{14}$$

Any SARIMA model can be transformed into an ordinary ARMA model in the variable $\tilde{B} = \nabla^d \nabla_s^D B$. Consequently, the estimation of the parameters of ARIMA and SARIMA models is analogous to that for ARMA processes.

Time series analysis in the simple case of an ARMA model

We now briefly discuss the steps of time series analysis in the simple case of an ARMA model, for more details see [12]. This approach is known as Box and Jenkins method: they let the data drive to the model and not vice versa, so that the time series can "speak for itself". Here is the main scheme:

1. Preliminary analysis: It is necessary to know the main features of the series and one checks e.g. the stationarity. There are so-called "unit roots tests" and often the Dickey-Fuller test [13] is used.
2. Order selection or identification: It is important to select appropriate values for the orders p and q. One can start by analyzing the total Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) and make some initial guesses based on their characteristics, see [12]. From these, one can obtain more reliable results using various "Information Criteria" (IC), such as Akaike Information Criteria (AIC) and Baesyan Information Criteria (BIC). ICs are an index, which tries to find a balance between the goodness of fit and the number of parameters. A "penalty factor" is included in the ICs to discourage the fitting of models with too many parameters. Hence, the preferred model is the one that minimizes the IC.
3. Estimation of the coefficients: once p and q are known, the coefficients of the polynomials can be estimated, e.g. by a least squares regression or with a maximum likelihood method. In most cases, this problem can be solved with numerical methods.
4. Diagnostic check: to check whether the residuals follow a random process. If the fitted model is suitable, the rescaled residuals should have similar properties to a "white noise" $WN(0,1)$ sequence. One can observe the sample "autocorrelation function" (ACF) of the residuals and perform tests for randomness, e.g. the Ljung-Box test (see Chapter 1 in [12]).

If the results are not satisfactory, one can restart with a different order selection and repeat the procedure. Once the model has successfully passed the verification phase, it can be used for forecasting.

4.2. Seasonality and its decomposition

There are different ways to deal with seasonality. One possible idea is to try to find out what kind of seasonality we have and see if it is possible to include it in the model, for example with the SARIMA model. Another possibility is to remove the seasonality.

For deseasonalization we will use the MATLAB function "deseasonalize" written by Weron, which was described in [6], and now we will describe in more detail the mathematical basis of this function. This function returns the deseasonalized data, the Short-Term Seasonal Component (STSC) and the Long-Term Seasonal Component (LTSC) obtained from the original data series. It also creates the periodograms of the original and deseasonalized data. With more detail, spectral analysis is about identifying possible cyclical patterns of data. The main goal of this type of analysis is to break down a seasonal time series into a few underlying sine (sine and cosine) functions of certain wavelengths. It is known that the variability of many physical phenomena can depend on frequency. Therefore, the information about frequency dependence could lead to a better knowledge of the true mechanisms. Spectral analysis and its basic tools, such as the periodogram, can help in this direction. For an observation vector $x_1, \ldots, x_n$ the periodogram (or the pattern analog of spectral density) is defined as:

$$I_n(\omega_k) = \frac{1}{n} \mid \sum_{t=1}^{n} x_t \, e^{-i(t-1)\omega_k} \mid^2, \tag{15}$$

where $\omega_k = 2\pi(k/n)$ are the Fourier frequencies expressed in radians per unit time, $k = 1, \ldots, [n/2]$, and $[x]$ denotes the largest integer less than or equal to x, for further technical details see e.g. [14].

This function implements a simple but quite efficient method to eliminate the short-term seasonal component. The idea is to divide the time series into a matrix with rows of length T (e.g. 7 element rows for a week period, calculated in average daily data) and taking the mean or median of the data in each column. The resulting row vector of length T is the estimate of the seasonal component and can be subtracted from the original data.

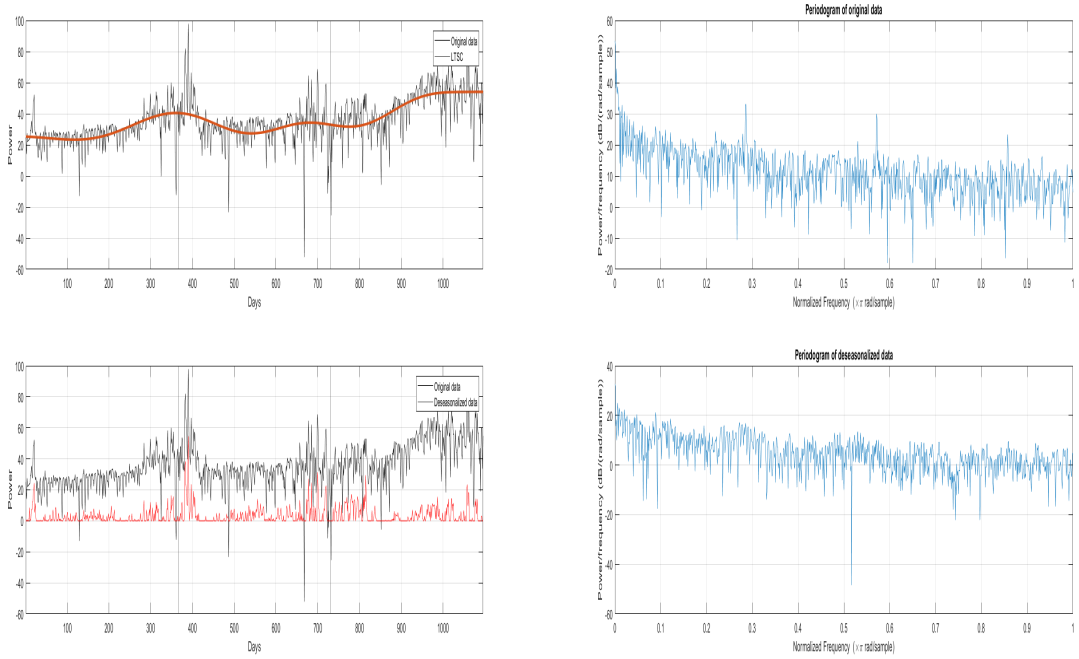
The wavelet decomposition for the long-term seasonal component

After removing the weekly or daily seasonality from the data, one often has to deal with the annual cycle. Although in many cases a sine function is a good first approximation of the annual cycle, there are markets where it could hardly be used, and the German market is one of them. This market has no clear annual seasonality, and spot prices behave similarly throughout the year, with peaks sometimes in winter and sometimes in summer, as can be seen in Figure 3. As suggested by Weron in [6], one possible option is to use a wavelet decomposition.

A wavelet family consists of pairs of a "father" (φ) and a "mother" (ψ) wavelet. The former represents the "low-frequency" smooth components: those that require wavelets with the greatest support, while the latter intercepts the "higher-frequency" detail components. In other words, father wavelets are used for the trend or cycle components, and parent wavelets are used for any deviation from the trend, see [15] and [6]. More specifically, wavelet decomposition of a signal uses a sequence of mother wavelets and only one father wavelet:

$$f(t) = S_J + D_J + \cdots + D_1, \tag{16}$$

where $S_J = \sum_k s_{J,k}\varphi_{J,k}(t)$ and $D_j = \sum_k d_{j,k}\psi_{j,k}(t)$. The coefficients $s_{J,k}$, $d_{J,k}$, $d_{J-1,k}, \ldots, d_{1,k}$ are the wavelet transform coefficients that measure the contribution of the corresponding wavelet function to the approximating sum, while $\varphi_{J,k}$ and $\psi_{j,k}(t)$ are the approximating father and mother wavelet functions, respectively. As soon as the signal is decomposed with (16), the procedure can be inverted to get an approximation to the original signal. If you want to use an unrefined scale, $f(t)$ can be estimated by S^J. For a higher degree of refinement the signal can be estimated by $S_{J-1} = S_J + D_J$. At each step we obtain a better estimate of the original signal by adding a mother wavelet D_j to a lower scale $j = J-1, J-2, \ldots$. The reconstruction process can always be interrupted, especially when we reach the desired accuracy. The resulting signal can be treated as a de-noised (or filtered or smoothed) time series. Figure 3 shows the results of the deseasonalization for the three years and the periodogram.



(a) *Deseasonalisation* **(b)** *Periodogram*

Figure 3. Example of results of "deseasonalize" on the three years

4.3. Expert models

The basic structure of an expert model is an autoregressive model, but other exogenous or input variables are also introduced. This setting is because electricity prices are not only related to their own past, but it is easy to understand that they are influenced by the current and past values of various exogenous factors, such as energy loads and weather changes.

The ARX Model

Within this expert model the centred log-price on day d and hour h is given by:

$$\begin{aligned} P_{d,h} = \beta_{h,1}P_{d-1,h} + \beta_{h,2}P_{d-2,h} + \beta_{h,3}P_{d-7,h} + \beta_{h,4}P_{d-1,\min} + \beta_{h,5}L_{d,h} \\ + \beta_{h,6}D_{\text{sat}} + \beta_{h,7}D_{\text{sun}} + \beta_{h,8}D_{\text{mon}} + \epsilon_{d,h} \end{aligned} \tag{17}$$

where the $\epsilon_{d,h}$ are assumed to be independent and identically distributed (i.i.d.) normal random variables. We abbreviate this autoregressive benchmark with *ARX* to reflect the fact that in (17) the (zonal) load forecast is used as an exogenous variable. In contrast to the naive models, which did not require parameter estimation, we have to estimate the parameters in this type of linear model.

The mARX Model

It can be advantageous to use different model structures for different days of the week, not only different parameter sets, see [11]. For example, the so-called multi-day ARX model or mARX reads

$$\begin{aligned} P_{d,h} = \Big(\sum_{i\in I}\beta_{h,1,i}D_i\Big)\, P_{d-1,h} + \beta_{h,2}P_{d-2,h} + \beta_{h,3}P_{d-7,h} + \beta_{h,4}P_{d-1,\min} + \beta_{h,5}L_{d,h} \\ + \beta_{h,6}D_{\text{sat}} + \beta_{h,7}D_{\text{sun}} + \beta_{h,8}D_{\text{mon}} + \beta_{h,11}D_{\text{mon}}P_{d-3,h} + \epsilon_{d,h} \end{aligned} \tag{18}$$

where $I \equiv \{0, \text{Sat}, \text{Sun}, \text{Mon}\}$, $D_0 \equiv 1$ and the term $D_{\text{mon}}P_{d-3,h}$ explains the autoregressive effect of Friday's prices on the prices for the same hour on Monday. Note that this structure resembles periodic autoregressive models to a certain extent (i.e. PAR, PARMA). Both autoregressive models (ARX and mARX) could be estimated with least squares (LS) methods.

4.4. Artificial Neural Networks (ANNs)

These algorithms gradually learn from input data, and they are based on the idea of the structure and functions of the neural networks of the brain and for this reason they are called Artificial Neural Networks (ANNs, or we can simply call them NNs). A remarkable property of NNs that distinguishes them from statistical methods is the fact that they are trained from data through a "learning process".

An NN is based on a collection of interconnected nodes called artificial neurons, like the structure of a human brain. Each connection, like the synapses, can transmit a signal to other neurons. They are organized in the form of layers. We have an input layer of source nodes that projects onto an output layer of neurons. Then we have the presence of one or more hidden layers whose function is to mediate between the external input and the network output. The more hidden layers, the more higher order statistics are accessible, [16].

Over the years, many varieties of NNs with different properties and applications have been introduced. An important distinction is made between NNs whose compounds form "cycles" and those whose compounds are "acyclic". NNs with cycles are called feedforward neural network (FNN). However, they have the three basic components of an NN in common: a set of synapses, each of which is characterized by a weight representing the strength of the synapses between neurons; an adder for summing the input signals; an activation function for limiting the amplitude of a neuron and introducing nonlinearity into the output of a neuron.

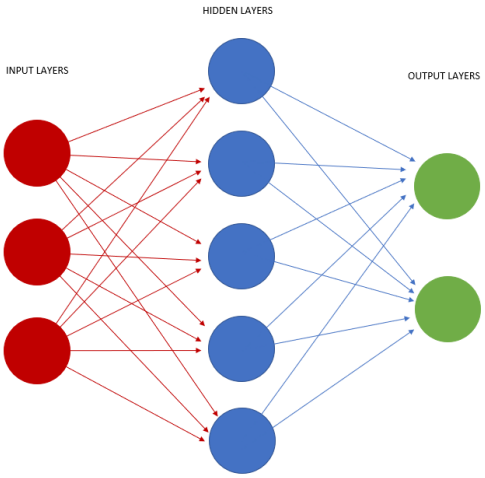


Figure 4. Example of a structure of a simple NN

In mathematical terms we can describe a neuron k by writing the following:

$$v_k = \sum_{j=1}^{m} w_{kj}\, x_j \quad \text{and} \quad y_k = \varphi(v_k), \tag{19}$$

where x_j are the signals, w_j are the synaptic weights of a neuron k, φ is the activation function, that maps a node's inputs to its corresponding output, and y_k the output signal of the neuron.

There exist different activation functions, e.g. the logistic sigmoid function or the tanh function:

$$\sigma(x) = \frac{1}{1+\exp(-x)}, \qquad \text{or} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}. \tag{20}$$

One can observe that these two functions are related by the following linear transformation:

$$\tanh(x) = 2\sigma(2x) - 1. \tag{21}$$

This means that any function computed by a neural network with a hidden layer of tanh units can be computed by another network with logistic sigmoid units and vice-versa.

The weights are optimized using an optimization algorithm that minimizes a loss function. The standard optimizer in this context is called Stochastic Gradient Descent (SGD) and one common loss function is the Root Mean Squared Error (RMSE). The basic idea of gradient descent is to find the derivative of the loss function with respect to each of the network weights and then adjust the weights in the direction of the negative slope. A possible efficient method for calculating this gradient can be the technique known as "back-propagation", for details we refer to [17,18].

Once all data points have passed through the network, we say that an epoch is complete, i.e. an epoch refers to a single pass of the entire data set through the network during training. This procedure is repeated several times: this is what "training the network" means. At the end of each epoch the loss of a single output is evaluated and it is possible to calculate the gradient of this loss in relation to the selected weight. The path to this minimized loss occurs in several stages, and its magnitude depends on the learning rate, which is typically between 0.01 and 0.0001. Therefore, the new weight is the old weight minus the learning rate times the gradient.

In summary, the workflow for the general NN design process comprises six primary steps: creating the network, configuring the network, initializing the structure, training the network, validating the network (post-training analysis) and using the network, see [16]. For our data analysis, we will use a special category of Recurrent Neural Network (RNN) called Long Short Term Memory (LSTM).

4.4.1. Recurrent and Long-Short Term Memory Networks

RNNs allow cyclic connections, i.e. when time series are involved and we do not want to lose information about past relationships between data. In fact, an RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping with any degree of accuracy [19]. The recursion allows the network to remember earlier inputs: the output of the network at time t is not only connected to the input at time t, but also to recursive signals before time t. A problem is that in some cases the gradient becomes too small, which effectively prevents the weight from changing its value, and sometimes this can completely prevent the neural network from further training, see [20]. This weakness of the RNN therefore makes it unsuitable when time series prediction applications require learning of dependencies over long distances or long-term storage of contexts. This problem can be solved by a variant of the RNN: the long-term short-term memory (LSTM) architecture [20].

We will now briefly review how LSTM works, for more details see [20,21]. For the description of these steps, σ and tanh from (20) are used as activation functions, but others can be implemented depending on the purpose.

- We define an input in the time step t as (X_t) and the hidden state from the previous time step as (S_{t-1}), which is inserted into the LSTM block. Then the hidden state (S_t) is to be calculated.
- It is important to decide which information from the cell state should be discarded. This decision is made by the following "forget gate":

$$f_t = \sigma(X_t\, U^f + S_{t-1}\, W^f + b_f). \tag{22}$$

- Then you must decide which new information should be stored in the cell state. This part consists of two steps: First, the "Input-Gate" layer (i_t) decides which values are updated. Secondly, a tanh layer, which creates a vector of new candidate values $\tilde{C}_t$. These two folds can be described as

$$i_t = \sigma(X_t\, U^i + S_{t-1}\, W^i + b_i), \qquad \tilde{C}_t = \tanh(X_t\, U^c + S_{t-1}\, W^c + b_c). \tag{23}$$

- Then, update the old cell state, C_{t-1} into the new cell state C_t, which can be given as:

$$C_t = C_{t-1} \otimes f_t \oplus i_t \otimes \tilde{C}_t. \tag{24}$$

- Finally, the chosen output will be based on the cell state, but will be a filtered version. In this step the output gate (o_t) decides which parts of the cell state should be produced as output. Then the cell state passes through the tanh layer and is multiplied by the output gate as

$$o_t = \sigma(X_t\, U^o + S_{t-1}\, W^o + b_o), \qquad S_t = o_t \otimes \tanh(C_t). \tag{25}$$

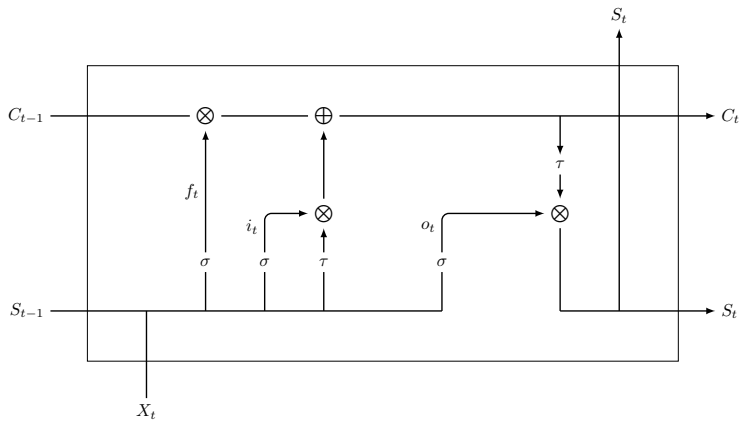
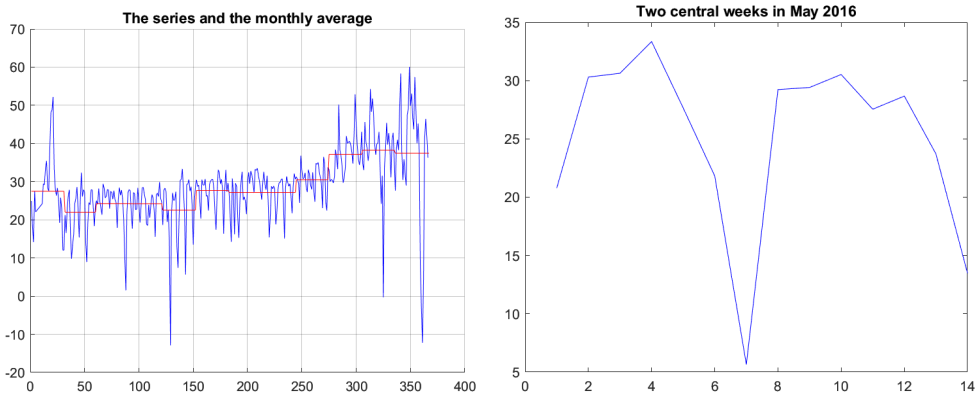


Figure 5. Example of a LSTM's memory block, where f_t , i_t , o_t are forget, input, and output gates respectively; τ and σ represents the tanh and sigma activation functions of [21].

5. Data Analysis

We have three years with hourly energy prices on the German electricity market, from 01.01.2016 to 31.12.2018. For our analysis we will consider the daily average for each day, for a total of 1096 observations. We use historical observations, our endogenous variable, as input, and we want a forecast for future values as output. This is a problem of the regression type, because we want to create a multilevel forecast of a numerical quantity. Our time series is coherent because the observations are collected and structured uniformly over time, for example, because we have a seasonal pattern.

First of all, it is important to have a general idea of the behavior of the data. As an example, we record the data for the year 2016 and additionally their monthly average as well as two weeks in May. We can see the pattern of weekly seasonality, see Figure 6(b): We can clearly see the weak seasonality and the difference between Mon-Fri and the weekend. This example proves the dependence of the price on external factors like business activities etc. There are some outliers and on average the same days show a negative price. For more details on how peaks are handled in the energy market, see [22].



(a) *Plot of daily (blue) and monthly (red) average for the year 2016*

(b) *Two "ordinary" working weeks in May*

Figure 6. Data visualization

5.1. From the probability density function to a possible realization of the stochastic process

There are some interesting questions that we would like to answer, for example: whether it is possible to estimate a stochastic process that obeys a certain PDF; whether we can create a kind of forecast from it; whether it could be a suitable starting point for more structured forecasts. This approach could help a user to have a better knowledge of his future probable data for a mid-term period. We start by describing the general idea from a mathematical point of view, and we see how we can try to apply it. We introduce the rejection method and one of its improvements, for more details see e.g. [23]. They generate random variables with the non-uniform distribution.

The rejection method

The rejection method creates a random variable with a given PDF f by using two independent generators $U(0,1)$. The idea of John von Neumann is to create a box $[a,b] \times [0,c]$ containing the graph of f. Then a random point (U,Y) in the box and U is accepted if the point is below the graph. This is a simple rejection algorithm that can be generalized for a d-dimensional PDF:

1. Generate $U \sim U(a,b)$ from $U = a + (b-a)U_1$ with $U_1 \sim U(0,1)$.
2. Generate $Y \sim U(0,c)$ from $Y = cU_2$ with $U_2 \sim U(0,1)$.
3. If $Y \leq f(U)$ then accept $X = U$, else reject U and return to step 1.

Then we know from the Fundamental Theorem of Simulation that the random variable X generated by the general rejection algorithm has the PDF f.

5.1.1. Marsaglia's Ziggurat Method

The Marsaglia's Ziggurat method is a highly efficient rejection method [24], implemented by the MATLAB function `randn`. It is based on an underlying source of uniformly distributed random numbers, but can also be applied to symmetric unimodal distributions, such as the normal distribution. An additional random sign $\pm$ is generated for the value determined by the Ziggurat method applied to the distribution in $[0, \infty)$. Instead of covering the graph of a given PDF f with a box, the idea now is to use a "ziggurat of rectangles", a cap and a tail, which all have the same area. These sections are selected so that it is easy to choose uniform points and determine whether to accept or reject them.

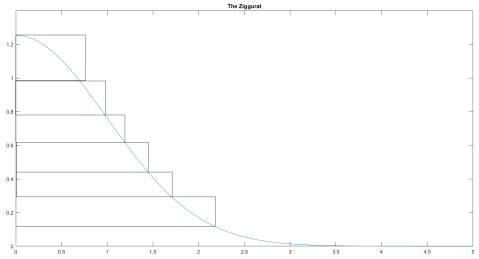


Figure 7. The Ziggurat basic idea

For simplicity we assume that f has support in $[0, \infty)$ and decreases monotonously. For a given number M we ask for the nodes $0 = x_0 < \cdots < x_M < \infty$, so that the ziggurat rectangles Z_i and the tail Z_0 have the same area. Then the Ziggurat method reads:

1. Choose $i \in 0, \ldots, M$ at random, uniformly distributed. Generate $U \sim U(0,1)$.
2. If $i \geq 1$ then
   - Let $X = Ux_i$.
   - If $X < x_{i-1}$ then return X,
     else generate $Y \sim U(0,1)$ independent of U.
     If $f(x_i) + Y\big(f(x_{i-1}) - f(x_i)\big) < f(X)$ then accept X, otherwise reject.
3. If $i = 0$ then
   - Set X = $(vU)/f(x_M)$
   - If $X < x_M$, accept X,
     else generate a random value $X \in [x_M, \infty)$ from the tail.

It is important to realize that the resulting distribution is exact, even if the ziggurat step function is only an approximation of the probability density function.

5.1.2. The function `randn` and its possible application

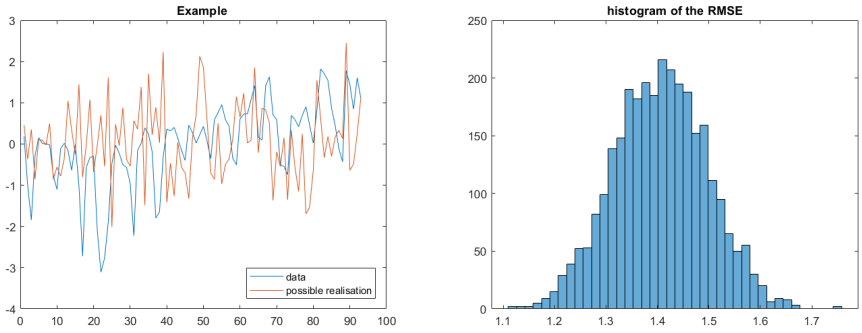
We consider the summer season (01/06 - 31/08) of the average daily price for the two years 2016 and 2017 to obtain similar data. The question arises whether it is possible to create a probable realization for the summer of 2018 (or only for one period), taking into account the probability density functions of the previous data. Before we continue, we standardize the data so that they have a normal distribution in the mean. In the following figure we can observe the behavior of our data for this particular period. In Figure 8(a), we can observe that 2016 and 2017 are very similar, while the 2018 data are much higher on average.

Then we use the MATLAB function `randn`, which is implemented with the Ziggurat's algorithm with the options: $X = \text{randn}(\_, 'like', p)$ returns an array of random numbers like p. X will be our possible realization for summer 2018. For example, if we generate 1000 paths and we evaluate the $RMSE$, we get in Figure 9 the following histogram, where we can check that the values are mainly in the range $[1.35, 1.46]$. As an example we could get with $RMSE = 1.33$:

It is important to see what could happen if we perform the inverse standardization procedure; the results are shown in Figure 9 and Figure 10. This big difference could be due to the fact that summer

(a) *Original data* **(b)** *Standardised data* **(c)** *Densities*

Figure 8. Behaviour of data and their densities.

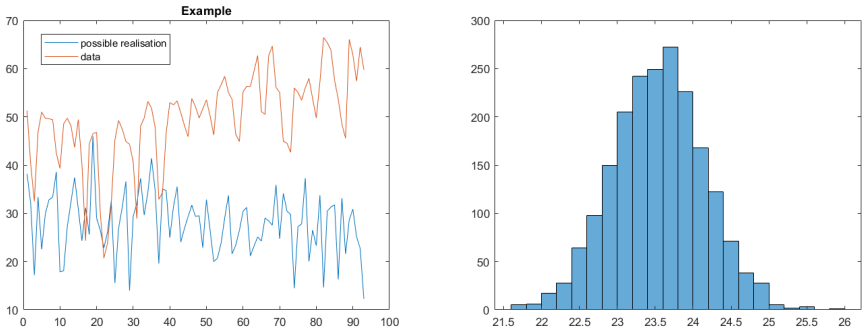


(a) *Example* **(b)** *Histogram of RMSE*

Figure 9. Example of a possible realisation with standardised data

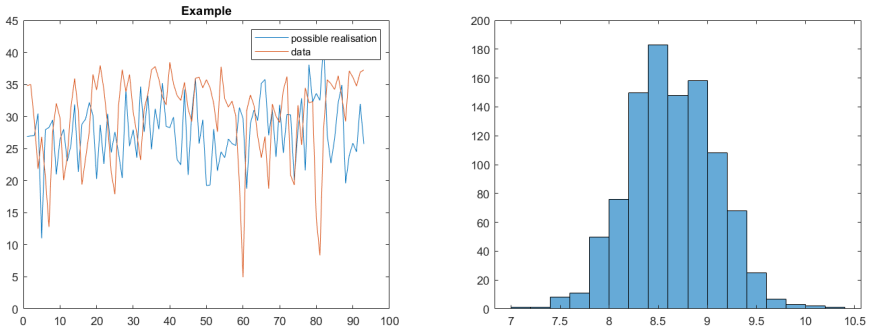
2018 was quite different from the last two summer times. Also, the data densities are not completely unimodal and symmetric, so the algorithm may not work properly. Therefore, one can say that this first approach gives better results if the situation is regular over time and always follows the same "trend", or if one has access to additional information and variables that can improve performance. If we apply the same procedure only with 2016 for a forecast for 2017, we get in fact the Figure 11. We can also note that any possible realization never reaches an "outlier" or a particularly high spike.

Finally, our opinion is that this approach cannot be very useful from a practical point of view. It can help to get a general idea of the medium and long term, but you cannot really rely on its accuracy.



(a) *Example* **(b)** *Histogram of RMSE*

Figure 10. Use of summer of 2016 and 2017 for possible data of summer in 2018



(a) *Example* **(b)** *Histogram of RMSE*

Figure 11. Use of only the summer of 2016 for possible data of summer in 2017

5.2. Example with SARIMA model

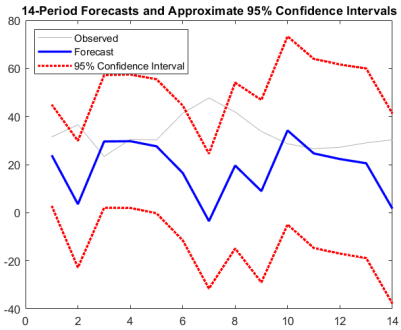
In this section we present the results of a forecast obtained by implementing a SARIMA model. To calculate the forecast we use the Matlab model `sarima` with the parameters using the estimator und the function `forecast` to predict the average prices for the next few days. We try to predict two weeks of the year 2018, and we will use three different training periods: one year (2017), the spring/summer season from April to September 2017, and the autumn/winter period from October 2016 to March 2017. Table 1 shows the parameters used for each analyzed period. With $s = 7$ we refer to the fact that we implement a weekly seasonality as our kind of data suggests.

Training Period	SARIMA$(p,d,q) \times (P,D,Q)_s$
One year	$(2,0,2) \times (1,0,1)_7$
Spring/Summer	$(2,1,2) \times (1,1,1)_7$
Autumn/Winter	$(3,1,3) \times (1,1,1)_7$

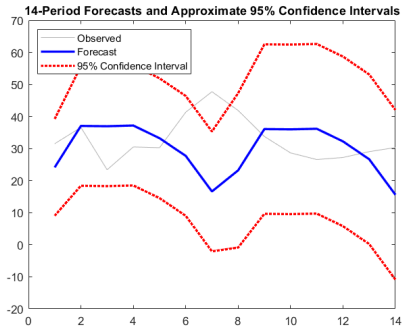
Table 1. SARIMA model's parameters

In addition, a test can also be performed to check the fit of the model, e.g. the well-known "Ljung-Box Q-Test", for more details see [25]. We test it for 20 lags with a significance level of 0.05. For all the models tested, we do not have to discard the null hypothesis, so that we can consider them as good and reasonable models for our data; the correlations in the population from which the sample is drawn are 0, so that all the correlations observed in the data result from the randomness of the sampling procedure.

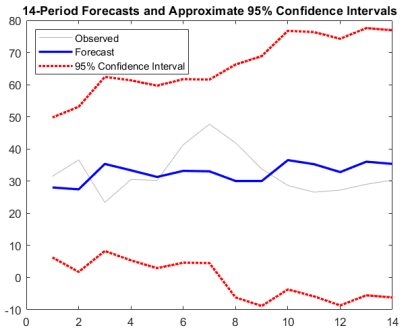
Figure 12 shows the results of the forecasts and their confidence intervals at 95%. In Table 2 we see the values of the RMSE: a shorter and more specific time period for the train has a better outcome.



(a) *Comparison between target and forecast values for two weeks with a training of the whole year 2017.*



(b) *Comparison between target and forecast values for two weeks with a training of spring/summer of 2017.*



(c) *Comparison between target and forecast values for two weeks with a training of autumn/winter of 2016/2017*

Figure 12. Comparison between target and forecast values for two weeks with the given training data.

Train Period	One Year	Spring/Summer	Autumn/Winter
RMSE	21.5418	12.5163	8.1396

Table 2. Summary of the RMSE.

The results may not be very accurate, but we must remember that we are considering a daily average of the price, and therefore it is more important that the general trend shows similar behavior. In addition, we can observe that fluctuations occur for this type of data for certain periods of time, but the period is not as regular as for other types of data, so it is more difficult to deal with.

5.3. ARIMA model

Since our series are quite long, it may not be sufficient to model a weekly seasonality, but it may be necessary to include other components. As in the SARIMA example, we will start by modeling 2017 to produce a forecast for the first two weeks of 2018, and then we will see results when we look at only one period of the year (spring/summer; fall/winter).

- We decide to deseasonalize the series using the Matlab function written by Rafał Weron called `deseasonalize`, see [6], as we saw earlier in the section 4.2 on seasonality. It considers both a short-term and a long-term seasonal component.

- We have used an Augmented Dickey-Fuller test to verify that our deseasonalized series is stationary. It tests the null hypothesis that a unit root is present in a time series sample. In Matlab we can use the function *adftest*, it returns a logical value with the rejection decision for a unit root in a univariate time series. For example, the result $adf = 0$ indicates that this test does not reject the null hypothesis of a unit root against the trend-stationary alternative.

  In our cases we have that the series is not stationary, so we can differentiate it: with a differentiation the series becomes stationary, and this is equivalent to the parameter d equals 1.
- Our goal now is to find a suitable ARIMA(p, d, q) model to estimate the series. To guess a plausible order of the parameters, we consider the autocorrelation and partial autocorrelation functions as proposed in the procedure of Box and Jenkins.
- We try to estimate different types of models, and we choose the one that minimizes the information criterion AIC and BIC. We choose ARIMA(1,1,2), and we can represent it with

$$(1 - \phi_1 B)(1 - B)\, x_t = (1 + \theta_1 B + \theta_2 B^2)\, \epsilon_t \tag{26}$$

- Our idea now is to calibrate our parameters to optimize the error in the L^2 norm of the difference between the Probability Density Function (PDF) of the data and the forecast we calculated based on the estimated model. The PDF is estimated by the Matlab function *ksdensity*. It uses a non-parametric method called kernel density estimation. We are interested in estimating the form of the density function f from a sample of data $(x_1, \ldots, x_n)$; its kernel density estimator is

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{N} K\Big(\frac{x - x_i}{h}\Big), \tag{27}$$

  where K is the kernel, $h > 0$ denotes a smoothing parameter called bandwidth and N denotes the number of observations. By default K is set as the normal density function.
- The optimal prediction results from the solution of this problem:

$$\min_{\phi_1,\ldots,\phi_p,\theta_1,\ldots,\theta_q} \| PDF(\text{forecast})_{\{\phi_1,\ldots,\phi_p,\theta_1,\ldots,\theta_q\}} - PDF(\text{data}) \|_2, \tag{28}$$

  where $\phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ are parameters of the chosen model and we choose a compact interval of $\mathbb{R}$ where they can vary before solving the problem. With $PDF(\text{forecast})$ we denote the estimated density function from the obtained forecasts. To solve this minimization problem we use the Matlab function `fminsearch`, which determines the minimum of a multi-variable function using a derivative-free method. In this case, we use the Matlab function `arma_forecast` for the point forecasts, see [26]. In particular, `fminsearch` uses the Nelder-Mead simplex algorithm. It is a direct search method that does not use numerical or analytical gradients. This method depends on the given initial values: we use the parameters that come from the first estimate of the ARIMA model.

  This approach differs from the one used by Ziel and Steinert [2]. The optimal parameters of the problem are given by the solution of minimizing the BIC criteria using the lasso technique. The general idea is that for a linear model $Y = \beta' \mathbf{X} + \epsilon$ the LASSO estimator is given by:

$$\hat{\beta} = \min_{\beta} \| Y - \beta' \mathbf{X} \|_2^2 + \lambda \| \beta \|_1 \tag{29}$$

  where $\lambda \geq 0$ is the penalty term.
- We plot the density functions and see the results in Figure 13.
- In the Table 3 one can compare between the error of the difference before and after optimization:

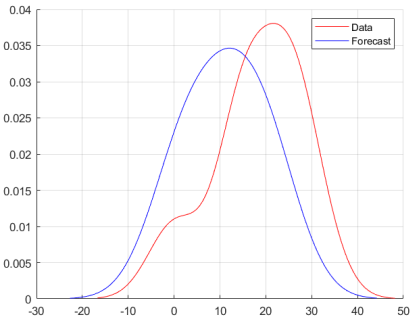
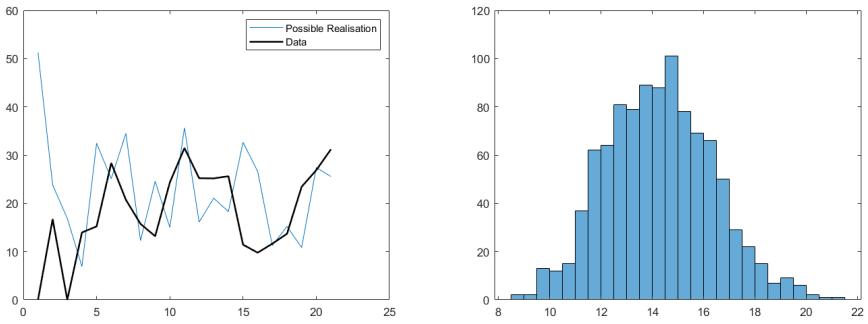


Figure 13. Estimated density functions of forecasted and original data

Forecasted days	Error before	Error after
14	$1.83 \cdot 10^3$	0.1098

Table 3. Errors before and after the optimization

- Finally we try to implement the idea described in Section 5.1. In view of the PDF of the point estimates, we can guess a possible realization of the future period, for example for the first three weeks of January 2018. A plausible result can be observed in Figure 14.



(a) *Possible path for three future weeks of 2018.*

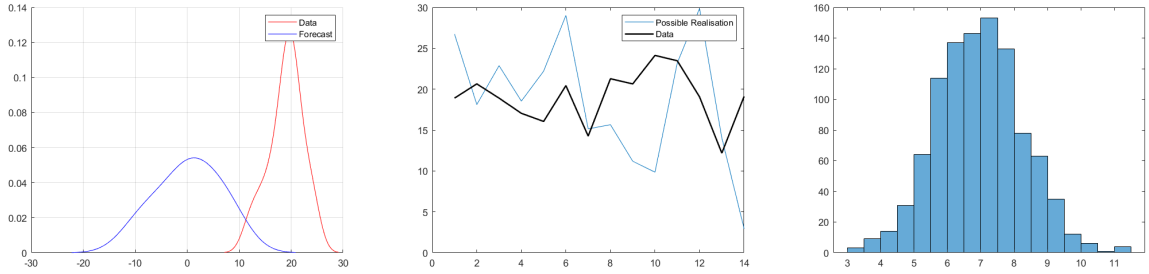
(b) *Histogram*

Figure 14. From PDF to the stochastic process.

Now we follow the same steps - test for stationary, identification of the model, calibration of the parameters - to see if we can achieve an improvement of the RMSE by reducing the number of observations and choosing a certain time period for training and testing. So we use first the autumn/winter period and then the spring/summer period.

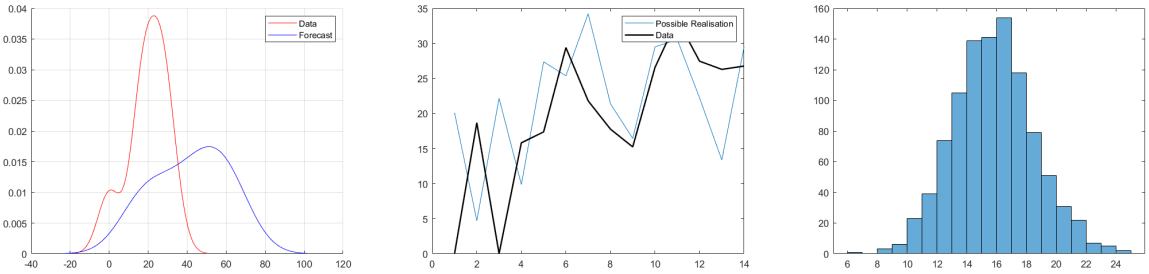
Period	ARIMA	Forecasted days	Error before	Error after
Autumn/Winter	(2,0,0)	14 days in January	1.7390	0.1223
Spring/Summer	(1,1,4)	14 days in June	11.1979	0.1375

Table 4. Summary for the two restricted period.



(a) *Estimated PDF* **(b)** *Possible path for two weeks of June 2018* **(c)** *Histogram*

Figure 15. Summary of results for spring/summer period.



(a) *Estimated PDF* **(b)** *Possible path for two weeks of January 2018* **(c)** *Histogram*

Figure 16. Summary of results for autumn/winter period.

We can note that the values of $RMSE$ are now generally not as high as in the case considered in Section 5.1, and this is due to the fact that we use a more structured procedure in the preparation of our forecast. We are aware that this type of approach may not be appropriate if we want to achieve a high degree of accuracy; there are already a lot of other methods for this purpose. Here we try to imagine a possible and plausible trend of future data for a medium-term period. As already mentioned and as is the case with statistical models, this kind of approach in combination with ARIMA leads to better results when the behavior of the series is quite regular.

5.4. Neural Networks

For the same data we try to use an ANN to calculate the forecast. In particular, we implement in MATLAB the LSTM architecture from Section 4.4.1. The selection of a suitable representation and preprocessing of the input data is a very important part of any machine learning task. However, neural networks tend to be relatively robust to the choice of input representation, see [17]. The only requirements for input representations of NNs are that they are complete, i.e. that they contain all the information needed to successfully predict the output. Although irrelevant inputs are not a major problem for NNs, a large input space could lead to too many input weights and poor generalization.

A common procedure is the standardization of the components of the input vectors. This means that first the mean value and the standard deviation are calculated with the classical estimators:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \text{and} \qquad \sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu)^2}, \tag{30}$$

then calculate the standardised input vector $\hat{x}$ using $\hat{x} = \frac{x-\mu}{\sigma}$.

This procedure does not change the information in the training set, but it does improve the performance by moving the input values to a range that is more appropriate for the standard activation functions. Standardizing the input values can have a huge impact on network performance, [17].

Once we have standardized, we train the network with Matlab. We use the function `trainNetwork`, where we specify the train set, the `lstmLayers` and appropriate options for the training. With the function `trainingOptions`, which is part of the Deep Learning Toolbox, we can set all the parameters we need, see Table 5. There is no specific procedure for selecting these values. Contrary to what we saw for the (S)ARIMA example, we do not have any criteria or index to justify a choice, but we have another one besides the RMSE. Here the decision is made after several attempts. There are two main problems to pay attention to. The first is over-fitting. Therefore it is important to prescribe an appropriate number of epochs for the training. For example, we can stop once we see that the loss function is stable. The second is to try to optimize the parameters: a very high number of hidden layers could be complex to handle from a computational point of view.

In Table 5 we also see the duration of the training. The training time is another difference to the statistical methods, which are more immediate. Here the elapsed time is about 1 minute, but for larger data sets this value can become hours.

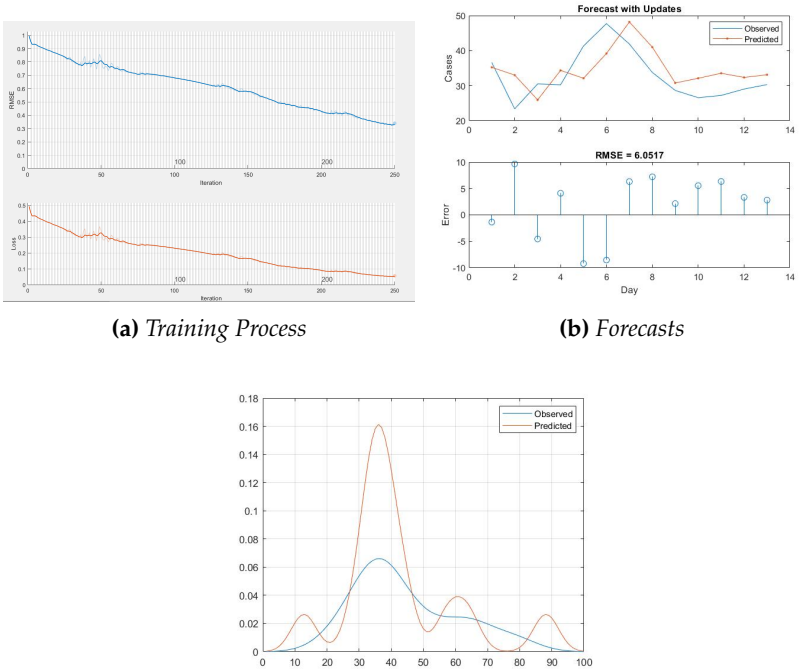
Number of Epochs	250
Hidden Layers	200
Initial Learn Rate	0.005
Elapsed Time	1 min 8 sec

Table 5. Summary of Neural Network architecture in Matlab

We know that the purpose of the SGD during training is to minimize the loss between actual performance and the performance expected by our training masters. Loss minimization is progressive. The training process begins with randomly set weights, and then we update these weights incrementally as we get closer to the minimum. The size of these steps depends on the learning speed. We know that during training, once the loss has been calculated for our inputs, then the gradient of that loss is calculated in relation to each of the weights in our model. We can observe this type of process in Figure 17*(a)*: We see that the loss function, the RMSE, is minimized epoch by epoch.

Once we know the value of these gradients, they are multiplied by the learning rate. This is a small number, usually between 0.01 and 0.0001, but the actual value can vary and any value we get for the gradient will be smaller once we multiply it by the learning rate. If we set the learning rate to a number on the higher side of this range, we risk the possibility of overshoot. This happens when we take a "too big" step towards the minimized loss function and overshoot beyond this minimum and miss it. To avoid this, we can set the learning rate to a number on the lower side of this range. Since our steps will be very small with this option, it will take much longer to reach the point of loss minimization. Overall, the act of choosing between a higher and a lower learning rate leaves us with this kind of compromise idea. All these starting parameters are something to test, and then you can choose the more suitable one for the problem, it is impossible to determine them in advance. With regard to the example with statistical methods, where we have information criteria that can lead us to optimal parameters, in this case we can only observe the final RMSE and the course of the training process. For example, if we see high peaks or irregularities until the end of the training, it might be necessary to add some epochs; conversely, if the loss function suddenly becomes very low and remains constant, we can decide to reduce the number of epochs or decrease the learning rate to avoid over-fitting phenomena.

In the following figures we can see the results: the training procedure, a comparison between forecasts and original data and the comparison between densities.



(a) *Training Process* **(b)** *Forecasts*

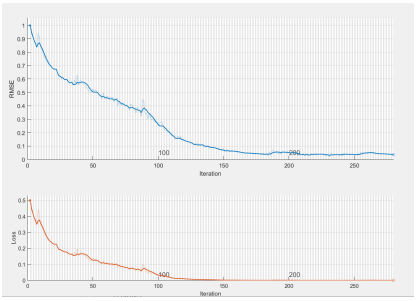
(c) *Densities*

Figure 17. Summary of results of LSTM network on one year data for two weeks forecast.

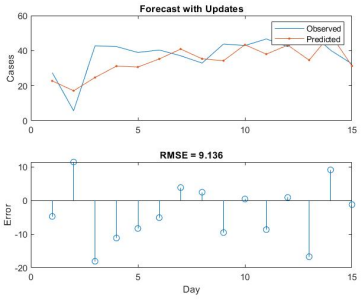
As we have done in the previous sections, we can repeat the same procedure using the spring/summer period, Figure 18; then using the autumn/winter period, Figure 19.

	Spring/Summer	Autumn/Winter
Epochs	280	250
Hidden Layers	190	190
Initial Learn Rate	0.009	0.007
Elapsed Time	56 sec	43 sec

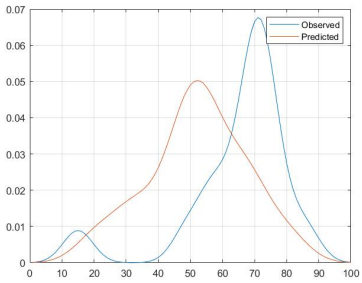
Table 6. Summary of Neural Network architecture in Matlab for the summer and winter period.



(a) *Training Process*

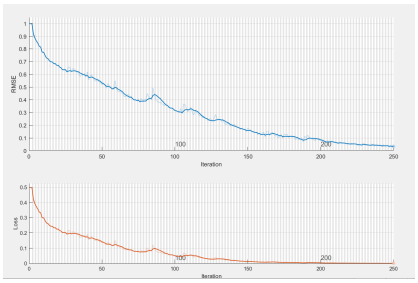


(b) *Forecasts*

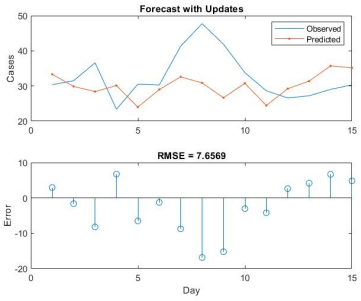


(c) *Densities*

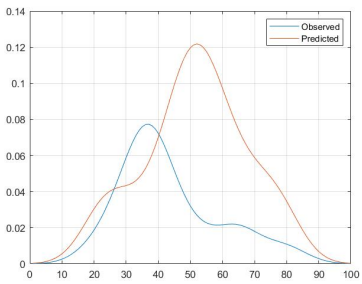
Figure 18. Summary of results of LSTM network on spring/summer period for two weeks forecast.



(a) *Training Process*



(b) *Forecasts*



(c) *Densities*

Figure 19. Summary of results of LSTM network on autumn/winter period for two weeks forecast.

As in the previous analysis, we can observe that we obtain good results and simplify the calculation effort if we shorten the period and make it more precise for a given season.

5.5. Hybrid Approach

In the literature there are other important approaches used to forecast the price of electricity: hybrid methods. The reason for this term is intuitive: they are obtained by combining other models, and their main feature is the fact that they can capture different patterns that characterize time series. A common example of these hybrid models is a combination between a statistical model, such as the ARIMA methods, and a model of computational intelligence, as we will present. For a better understanding we refer the interested reader to [27].

We will now see if it is possible to combine our methods to obtain significant results. The simplest approach is to look at an average of them or find suitable weights w_i:

$$forecast = w_1 SARIMA(results) + w_2 LSTM(results) \tag{31}$$

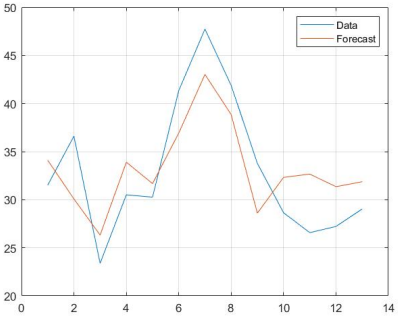
This procedure is quite common, it allows good advantages in deciding which component needs to be highlighted. In Table 7 we specify the weight of each component we use for the hybrid approach. While in Table 8 we can see the final comparison between RMSE. As we can see from these values, the hybrid approach can be a valid tool to obtain a lower RMSE.

	One Year	Autumn/Winter	Spring/Summer
SARIMA	10%	30%	50%
NN	90%	70%	50%

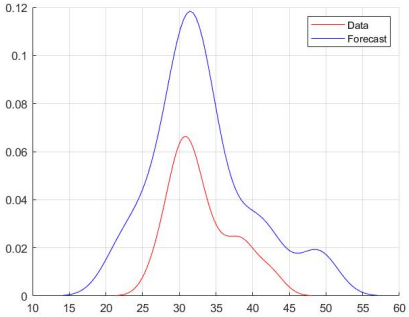
Table 7. Percentage of component for each training period.

	One Year	Autumn/Winter	Spring/Summer
SARIMA	21.54	8.13	12.53
NN	6.05	7.65	9.13
Hybrid	4.15	7.08	9.38

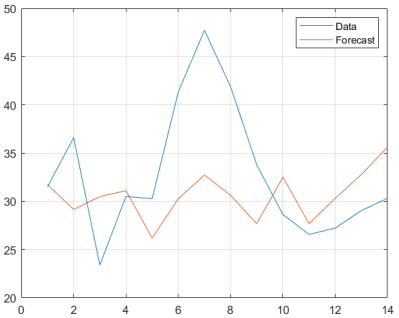
Table 8. A comparison between RMSE.



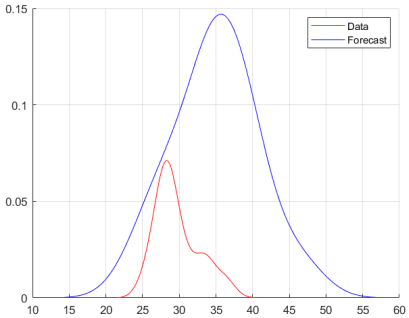
(a) *Hybrid forecast for one year of training.*



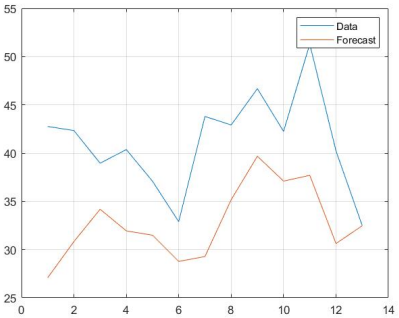
(b) *Densities.*



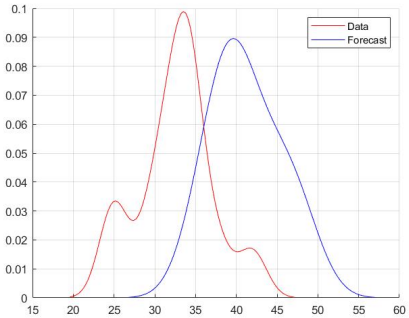
(c) *Hybrid forecast for autumn/winter period of training.*



(d) *Densities.*



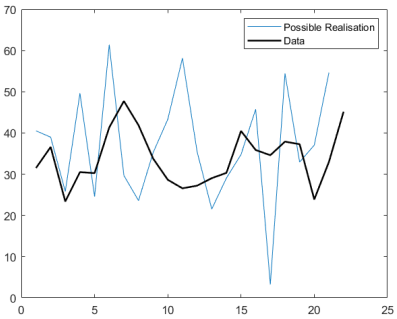
(e) *Hybrid forecast for spring/summer period of training.*



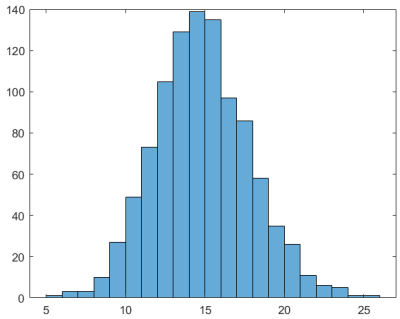
(f) *Densities.*

Figure 20. Summary of results of hybrid approach for each time period of training.

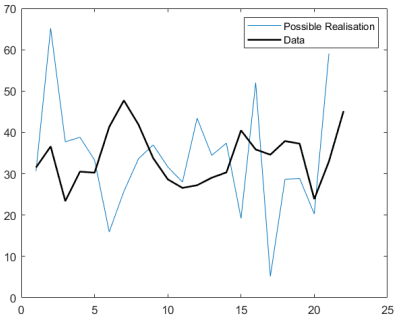
In the following we also test our idea to capture the underlying possible stochastic process, results are shown in Figure 21. We try to get good approximations to the investigated processes on the basis of the examined data, assuming that their development can be well shaped by a certain (previously specified) type of stochastic process. As we can see from the previous figure, the estimated densities are not precise, so our approach does not yield better results. From this point of view, a hybrid approach does not yield any improvement.



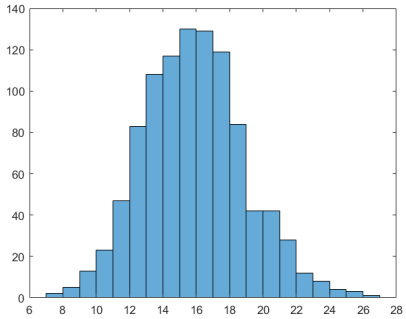
(a) *Hybrid forecast for one year of training.*



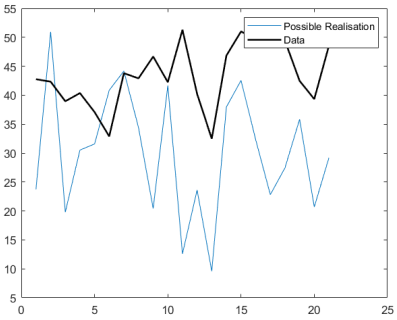
(b) *RMSE*



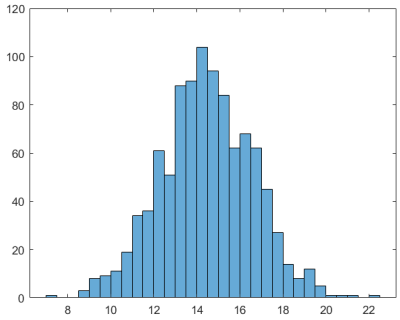
(c) *Hybrid forecast for autumn/winter period of training.*



(d) *RMSE*



(e) *Hybrid forecast for spring/summer period of training.*



(f) *RMSE*

Figure 21. Summary of results of probabilistic hybrid approach for each time period of training for a time horizon of three weeks.

6. Conclusions

In this paper we gave a general overview of energy price forecasting and in particular highlighted its most important quantitative aspects using two of the most commonly used approaches to analyze the corresponding behavior over time. We also included applications to a real-time series. We first focused our attention on (S)ARIMA models, which are the starting point for most analyses of time series. The results are quite adequate, but we can obtain better results by using approaches based on Neural Networks (NNs), especially the LSTM type, the latter being one of the most popular within the NNs-oriented community because of its highly accurate results, see e.g. [28]. These improvements are accompanied by a more complex computational structure and a higher data processing time.

In general, one could say that NNs-based results are often better than those obtained with an ARIMA model, cf. Table 8. This is because these networks allow us to obtain and store more information, even for more complex seasonal influences. It is also worth remembering that ARIMA works better in very regular situations. Both are valuable models, but in this case with this kind of data and context, LSTM can be a suitable tool not only for our probabilistic approach, but also for point estimation for short and medium term period forecasts, for more details see [27]. With the hybrid approach, Section 5.5, we have seen that we can balance our results and achieve some significant improvements with the accuracy of the point forecast for two weeks.

Let us finally emphasize that our results regarding the prediction of the (random) electricity price behavior (expressed in the form of a stochastic process) are based on a rather innovative probabilistic proposal, which gives us remarkable accuracy results. From a theoretical point of view, the results obtained are quite good and the idea is quite original for a probabilistic forecast.

Author Contributions: All authors contributed equally to this work.

Funding: This research received no external funding.

Acknowledgments: Emma Viviani would like to underline that the material developed within the present paper has been the result of her Erasmus+ experience she spent exploiting the E+ agreement between the University of Wuppertal and the Comp-Science Dept. of the University of Verona, collaborating with Prof. Matthias Ehrhardt and under the supervision of Prof. Luca Di Persio, both of them being her MSc-thesis coordinators.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACF	Autocorrelation Function
AIC	Akaike Information Criteria
BIC	Bayesian Information Criteria
CDF	cumulative distribution function
CRPS	continuous ranked probability score
DM	Diebold Mariano
EPEX	European Power Exchange
EPF	electricity price forecasting
FNN	feedforward neural network
GARCH	generalized autoregressive conditional heteroskedasticity
i.i.d.	independently identically distributed
LASSO	least absolute shrinkage and selection operator
LSTM	Long Short Term Memory
LTSC	Long-Term Seasonal Component
OLS	Ordinary Least Squares
PACF	Partial Autocorrelation Function
PDF	Probability Density Function
PI	prediction intervals
PIT	probability integral transform
PL	pinball loss
QRA	Quantile Regression Average
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SARIMA	Seasonal Autoregressive Integrated Moving Average
SDG	Stochastic Gradient Descent
STSC	Short-Term Seasonal Component
VAR	Value-at-Risk

References

1. Weron, R. Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting* **2014**, *30*, 1030 – 1081.
2. Ziel, F.; Steinert, R. Probabilistic mid-and long-term electricity price forecasting. *Renewable and Sustainable Energy Reviews* **2018**, *94*, 251–266.
3. Nowotarski, J.; Weron, R. Computing electricity spot price prediction intervals using quantile regression and forecast averaging. *Computational Statistics* **2015**, *30*, 791–803.
4. Marcjasz, G.; Serafin, T.; Weron, R. Selection of calibration windows for day-ahead electricity price forecasting. *Energies* **2018**, *11*, 2364.
5. Zhang, G.; Patuwo, B.E.; Hu, M.Y. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting* **1998**, *14*, 35–62.
6. Weron, R. *Modeling and forecasting electricity loads and prices: A statistical approach*; Vol. 403, Wiley, 2007.
7. Bundesnetzagentur SMARD Strommarktdaten. Electricity Generation in August and September 2019. Available at: https://www.smard.de/en/topic-article/5870/14626, 2020.
8. Westgaard, S.; Paraschiv, F.; Ekern, L.L.; Naustdal, I.; Roland, M. Forecasting price distributions in the German electricity market. *International Financial Markets* **2019**, *1*, 11.
9. Nowotarski, J.; Weron, R. Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews* **2018**, *81*, 1548–1568.
10. Gneiting, T.; Katzfuss, M. Probabilistic forecasting. *Annual Review of Statistics and Its Application* **2014**, *1*, 125–151.
11. Nowotarski, J.; Weron, R. On the importance of the long-term seasonal component in day-ahead electricity price forecasting. *Energy Economics* **2016**, *57*, 228–235.
12. Brockwell, P.J.; Davis, R.A. *Introduction to time series and forecasting*, third ed.; Springer, 2016.
13. Dickey, D.A.; Fuller, W.A. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association* **1979**, *74*, 427–431.
14. Bloomfield, P. *Fourier analysis of time series: an introduction*; Wiley, 2004.
15. Percival, D.B.; Walden, A.T. *Wavelet methods for time series analysis*; Vol. 4, Cambridge Univ. Press, 2000.
16. Haykin, S. *Neuronal Networks, A comprehensive foundation*; Mc Master University, 1999.
17. Graves, A. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*; Springer, 2012; pp. 5–13.
18. Werbos, P.J. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks* **1988**, *1*, 339–356.
19. Hammer, B. On the approximation capability of recurrent neural networks. *Neurocomputing* **2000**, *31*, 107–123.
20. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation* **1997**, *9*, 1735–1780.
21. Sagheer, A.; Kotb, M. Time series forecasting of petroleum production using deep LSTM recurrent networks. *Neurocomputing* **2019**, *323*, 203–213.
22. Hagfors, L.I.; Kamperud, H.H.; Paraschiv, F.; Prokopczuk, M.; Sator, A.; Westgaard, S. Prediction of extreme price occurrences in the German day-ahead electricity market. *Quantitative finance* **2016**, *16*, 1929–1948.
23. Gentle, J.E. *Random number generation and Monte Carlo methods*; Springer Science & Business Media, 2006.
24. Marsaglia, G.; Tsang, W.W.; others. The Ziggurat method for generating random variables. *Journal of statistical software* **2000**, *5*, 1–7.
25. Ljung, G.M.; Box, G.E. On a measure of lack of fit in time series models. *Biometrika* **1978**, *65*, 297–303.
26. Sheppard, K. MFE MATLAB function reference financial econometrics. *Oxford University, Oxford. Available at: http://www.kevinsheppard.com/images/9/95/MFE_Toolbox_Documentation.pdf* **2009**.
27. Maciejowska, K.; Nowotarski, J. A hybrid model for GEFCom2014 probabilistic electricity price forecasting. *International Journal of Forecasting* **2016**, *32*, 1051–1056.
28. Brownlee, J. *Deep learning for time series forecasting: Predict the future with MLPs, CNNs and LSTMs in Python*; Machine Learning Mastery, 2018.