



Norwegian University
of Life Sciences

Master's Thesis 2022 30 ECTS
Faculty of Science and Technology

A Study of Statistical and Machine Learning Methods for Power Price Prediction Based on Filling Levels of Hydropower Reservoirs

Joanna Szalas
MSc Data Science

Acknowledgements

I would like to express my sincere gratitude to my supervisors from the Norwegian University of Life Sciences (NMBU)—Hans Ekkehard Plesser and Stefan Schrunner—for their scientific advice, thorough draft reviews and the support I have received over several months of work on this thesis.

I would also like to thank Frano Cetinic and EDInsights AS for the idea of the thesis topic and introducing me to the world of power markets and hydropower production.

I am grateful to all my teachers and colleagues from NMBU, who contributed to my current Data Science knowledge and skills.

Finally, huge thanks to Ania, Kara, Kasia, Marcin (the creator of the nicest figure in this thesis), Mum & Dad and Zuzia (note the quasi-alphabetical order) for being my greatest motivation, and for your continuous support and understanding.

Joanna Szalas
Oslo, 14 July 2022

Abstract

The European power markets have become highly integrated over the past decade. The electrical grids of individual countries are increasingly well connected between each other, which allows for trading the electricity on the common markets and thus enhances the development of diverse electricity sources across the continent. With that comes an increasing volatility of the power prices. It is in the interest of all market players involved in generating, supplying, trading and consuming the electricity to find a way to forecast the power price as accurately as possible. This study investigates the potential of using filling level data from hydropower reservoirs and historical power price data—particularly, the Nordic system price—to forecast the future system price. For this purpose, three forecasting models for time series analysis were developed and evaluated—a statistical approach, as well as two artificial neural network architectures with different levels of complexity. The statistical approach is based on the autoregressive integrated moving-average model with exogenous inputs (ARIMAX), while the investigated neural networks include (a) a standard recurrent neural network (RNN), and (b) a combination of one-dimensional convolutional layers (1D CNNs) and a long short-term memory cell (LSTM). The experimental part of this work is based on data collected from 63 Norwegian hydropower reservoirs between 2015–2021. An extensive hyperparameter tuning was conducted on the machine learning models, including input data transformations, prediction time frames, network architecture parameters and the shape of the RNN/LSTM 3D input data tensor. The ARIMAX model outperformed the machine learning models for both most thoroughly tested prediction time frames of 14 and 28 days, achieving the R^2 score of 0.8 and the MAE of 5.40 EUR. After a qualitative assessment of the obtained results it has been concluded that the models show some promising potential, however, a number of aspects would have to be further investigated to develop a mature solution, ready for practical use in, e.g. power trading.

Contents

Abstract	ii
Contents	iii
List of Figures	v
List of Tables	vii
Abbreviations	viii
1 Introduction	1
1.1 Background	1
1.2 Industrial partner	6
1.3 Motivation	7
1.4 Data	8
1.5 Problem statement	9
2 Theoretical background	10
2.1 Physical and financial power markets	10
2.1.1 Physical power markets	11
2.1.1.1 Day-ahead market	12
2.1.1.2 Intraday market	15
2.1.1.3 Balancing market	17
2.1.2 Financial power markets	17
2.2 Hydropower production planning	18
2.3 Time series forecasting	21
2.3.1 Time series	21
2.3.2 Statistical approach	22
2.3.3 Deep learning approach	25
2.3.3.1 Deep learning as a branch of machine learning	25
2.3.3.2 Single-layer artificial neural network	27
2.3.3.3 Multilayer artificial neural network	30

2.3.3.4	Convolutional neural network	33
2.3.3.5	Recurrent neural network	35
2.3.3.6	Machine learning workflow	39
2.3.3.7	Activation functions	40
2.3.3.8	Loss functions and performance metrics in regression tasks	41
2.3.3.9	Model complexity and cross-validation	43
2.3.3.10	Inception module	45
2.3.3.11	Related work	46
3	Methods	47
3.1	General experiment outline	47
3.2	Exploratory data analysis and pre-processing	47
3.3	Statistical modelling using ARIMAX	53
3.4	Machine learning modelling using a basic RNN architecture	54
3.5	Machine learning modelling using a hybrid 1DCNN-LSTM architecture	59
3.6	Software and hardware	70
4	Results	71
4.1	Exploratory data analysis	71
4.2	ARIMAX model	75
4.3	Basic RNN model	76
4.4	Hybrid 1DCNN-LSTM model	81
4.5	Summary	88
5	Discussion	89
5.1	Interpretation of the results	89
5.2	Shortcomings of the current approach	92
5.3	Alternative approaches	92
6	Conclusion	94
	Bibliography	95

List of Figures

1.1	Share of different energy sources in the electricity production in EU and in Norway	2
1.2	Factors causing the change of water level in the hydropower reservoir	5
2.1	Power market types	12
2.2	Bidding zones in the Nordic and Baltic regions	13
2.3	Price formation in the electricity surplus and deficit areas	14
2.4	Electricity price formation according to the merit order model . . .	16
2.5	Hydropower production planning outcome	20
2.6	Difference between classical programming and machine learning workflows	26
2.7	Single-layer neural network architecture on the example of Adaline algorithm	28
2.8	Simplified gradient descent concept	29
2.9	Multilayer Perceptron network	31
2.10	2D convolution of the input and kernel matrices	34
2.11	1D convolution on a univariate time series	35
2.12	Comparison of a standard feedforward network and recurrent neural network	37
2.13	LSTM memory cell	38
2.14	Activation functions	41
2.15	Training and validation curves	44
2.16	Comparison of network configurations: linear stack of layers vs Inception module	45
3.1	Raw dataset with missing data	50
3.2	Visualisation of the raw input dataset	51
3.3	Despiking of the raw system price time series	52
3.4	Simple RNN model summary	58
3.5	Input data transformations—a comparison of the original, residual and differenced filling level time series	65

3.6	Input data transformation—various levels of smoothing applied on the SYS price time series	66
3.7	Outer layer tuning schedule in the 1DCNN-LSTM model	67
3.8	A sample 1DCNN-LSTM network architecture	68
3.9	A sample 1DCNN-LSTM model summary	69
4.1	Heatmap of correlations between the raw input features	72
4.2	Cross-correlations between filling level time series and the SYS price time series shifted by various time lags	74
4.3	Best results of ARIMAX modelling for 14-day and 28-day prediction time frames	76
4.4	Training and validation learning curves of the best performing simple RNN models	78
4.5	Predictions on the training, validation and testing intervals for the RNN model with the 14-day prediction time frame	79
4.6	Predictions on the training, validation and testing intervals for the RNN model with the 28-day prediction time frame	80
4.7	The results of the outer and inner layers of tuning of the 1DCNN-LSTM model	83
4.8	Training and validation learning curves of the best performing 1DCNN-LSTM models	84
4.9	Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 14-day prediction time frame	85
4.10	Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 28-day prediction time frame	86
4.11	Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 42-day prediction time frame	87
5.1	Analysis of ARIMAX prediction plots	91

List of Tables

- 3.1 The most important Python libraries used in the project 70
- 4.1 The best performing ARIMAX models 75
- 4.2 The best performing RNN models 77
- 4.3 The best performing 1DCNN-LSTM models 81
- 4.4 Summary of the best performing 14 days ahead models 88
- 4.5 Summary of the best performing 28 days ahead models 88

Abbreviations

Abbreviation	Meaning
ACF	Autocorrelation Function
Adaline	Adaptive Linear Neuron
ANN	Artificial Neural Network
AR(p)	Autoregressive [model] of order p
ARIMA	Autoregressive Integrated Moving Average [model]
ARIMAX	Autoregressive Integrated Moving Average [model] with exogenous variables
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EDA	Exploratory Data Analysis
EUPHEMIA	Pan-European Hybrid Electricity Market Integration Algorithm
LSTM	Long Short-Time Memory
MA(q)	Moving Average [model] of order q
ML	Machine Learning
MLP	Multilayer Perceptron
NEMO	Nominated Electricity Market Operation
NN	Neural Networks
NVE	The Norwegian Water Resources and Energy Directorate
PACF	Partial Autocorrelation Function
RNN	Recurrent Neural Network
SDAC	Single Day-ahead Coupling
SGD	Stochastic Gradient Descent
SYS	Nordic system price
TSO	Transmission System Operator

Chapter 1

Introduction

1.1 Background

The Statistical Office of the European Union (Eurostat¹) and The U.S. Energy Information Administration (EIA²) define renewable energy as energy from sources which replenish themselves naturally, thus are practically inexhaustible but flow-limited [1][2]. In other words, there is a limit on how much of energy from such sources is available per unit of time. The main types of renewables are:

- Water (further referred to as *hydropower*)
- Geothermal
- Wind
- Solar
- Biofuels

Majority of the renewable energy sources (excluding combustible biofuels), as well as the nuclear energy, are considered clean energy sources. This means that they can be used to generate electricity without emitting harmful byproducts (e.g. carbon dioxide) to the atmosphere, as it happens in the case of fossil fuels-based

¹https://ec.europa.eu/info/departments/eurostat-european-statistics_en

²<https://www.eia.gov/>

electricity production [3]. There are many initiatives and policies, both at the European and global levels, to decarbonise the energy system. An example is the European Green Deal, which is a set of policies approved by the European Commission in 2020, aiming for the EU to become climate neutral by 2050 [4].

Electricity production constitutes around 23% of the total energy consumed by end users³. According to Eurostat, in 2020, for the first time in history, renewable energy took the lead among energy sources utilised for power production in the European Union. It corresponded to 39% of the generated electricity, compared to 36% coming from fossil fuels and 25% from nuclear plants [5]. If we look closer at the renewable electricity mix, 14% of the total electricity was produced from wind, 13% from hydropower, 6% from biofuels and 5% from solar energy. These statistics have been averaged over individual data from all EU member states. The countries differ between each other quite significantly in that matter. Each country has its own challenges on the way to achieving climate targets, different starting positions and different geopolitical predispositions. Therefore, it does not seem appropriate to compare their achievements directly, but it is worth taking a closer look at the ones standing out in the electricity mix proportions. One such country is Norway—which is not an EU member, but its economy and power market are, respectively, closely connected to and integrated with those of the EU countries. Hence, Norway is an important element of the European energy landscape, bringing in over 98% renewable share in the country’s electricity production, with hydropower alone accounting for 92% of production [5], as shown in Figure 1.1.

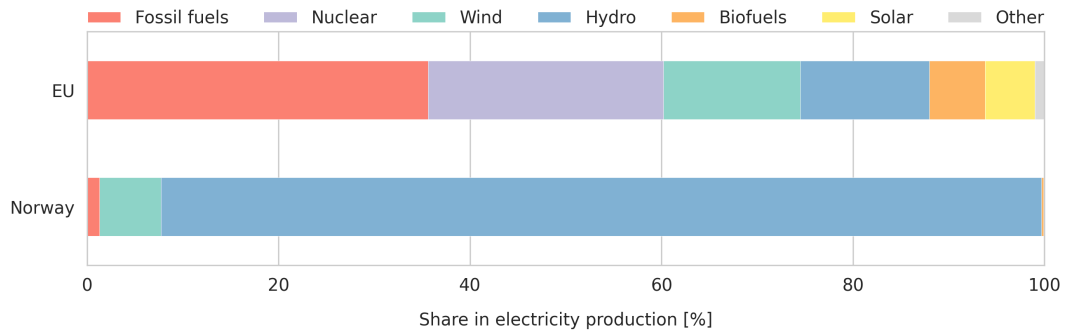


Figure 1.1: Share of different energy sources in the electricity production in EU and in Norway, based on the data from [5].

Norway pioneered the use of hydropower for industrial purposes in the late 19th century and continued through the Second Industrial Revolution fueled by clean,

³The remaining part is distributed among private households, industry, agriculture, transport and services.

renewable energy. Thanks to the advantageous geographical conditions, rich water resources and sustainable environmental governance, as of 2016 Norway was the largest hydropower producer in Europe and the 6th largest in the world [6].

The topic of the country’s hydro balance is vividly present in the public space in Norway and regularly makes its way to the national newspapers and TV. In this thesis, the hydro balance is understood as the total amount of water captured in a certain geographical unit at a given point in time. In the context of hydropower production, an interesting unit to calculate the hydro balance for is a hydropower plant located at a storage reservoir (further also referred to as *hydropower reservoir* or *regulated reservoir*; the plant uses the water from the reservoir to generate electricity). The hydro balance of the plant includes the amount of water that is already in the reservoir, as well as the amount accumulated in form of the snow or ice cover in the reservoir’s catchment area, assumed to eventually melt and end up in the reservoir. Adding up the calculated values for all hydropower plants in the region/country gives an overview of the hydropower potential at the regional/country level. The aforementioned public interest in the hydro balance topic is related to its historical correlation with the price which an average Norwegian had to pay for electricity delivered to their household. Generally speaking, in case of a “wet” year (high precipitation preceding and forecasted after a given point in time—high supply of the energy source), one would expect low electricity prices, and in a “dry” year—high prices. The prices are also driven by seasonal changes of market demand—more electricity is used in the winter due to society’s heating needs, and with the rising consumption the prices go up too [7].

Electricity production and trading have been deregulated in Norway in 1991 [8] and since then they function in a market-based system. Although this topic will be explained in more detail in Section 2.1, it is important to shed light already at this stage on how this fact impacts the hydropower sector. The main points to remember are:

- Electricity prices are determined by supply and demand in the power market [9].
- Norwegian power market is integrated with the markets of other Nordic countries and a large part of the European continent [10];
 - in terms of physical transmission capacity through a net of cross-border interconnectors (power cables),
 - in terms of electricity price dependencies through the common price coupling algorithm.

The process of market integration, especially the increase of the cross border transmission capacity, has changed the formation of electricity price paid by consumers in Norway dramatically over recent years. Electricity produced by companies operating within the integrated market comes from multiples energy sources—partly renewable, partly nuclear, partly conventional fossil fuels. The price for which the conventional producers are ready to sell their electricity on the market is typically much higher than the price viable for hydropower producers (hereinafter referred to as *hydro producers*) or wind power producers due to higher marginal cost of production (for more information see Section 2.1). Therefore, prices of the most expensive energy sources (oils, gas, coal) drive the final electricity prices in all regions of the integrated market, including Norway. Thus, the relationship between the power price and hydro balance in Norway has become more complex to model, due to an increasing number of influencing factors.

Hydro producers operating in such deregulated, highly competitive market need to carefully plan when to produce electricity. Their goal is to maximise profits, hence they want to produce when the market power price is high, and save the water in the reservoir for the future if the current price is low. For this purpose they need to develop reliable price forecasts, which are then fed into complex mathematical models created to optimise hydro production plans. The resulting production strategy is reflected in the reservoir's water level. The process of hydropower production planning is described in more detail in Section 2.2.

Positive and negative changes of the reservoir's water level are related to several factors. As illustrated in Figure 1.2, the water level increases as a result of direct precipitation, melting of snow from the surrounding higher-elevated terrain and inflow from the higher located hydropower plants. The water is caught in the reservoir by a dam which is integrated with the given reservoir's own hydroelectric facility. The water level drops when this plant turns its turbines on and starts generating electricity, using the flow of the water which has been for that purpose let into the system through the dam [11]. Intensity of the production is controlled by the operator and limited by the facility parameters. In general, a more intense production leads to a higher usage of water, which in turn empties the reservoir faster.

As a result of the described relation between produced electricity and water resources, the history of changes in the water level of a hydropower reservoirs is, to a high degree, a record of hydro producers' strategies. These, in turn, are based on the power price forecast. In this thesis we want to investigate whether it is possible to reconstruct trends of the said price forecast from the filling levels of the Norwegian reservoirs (filling level is a measure of how full the reservoir is,

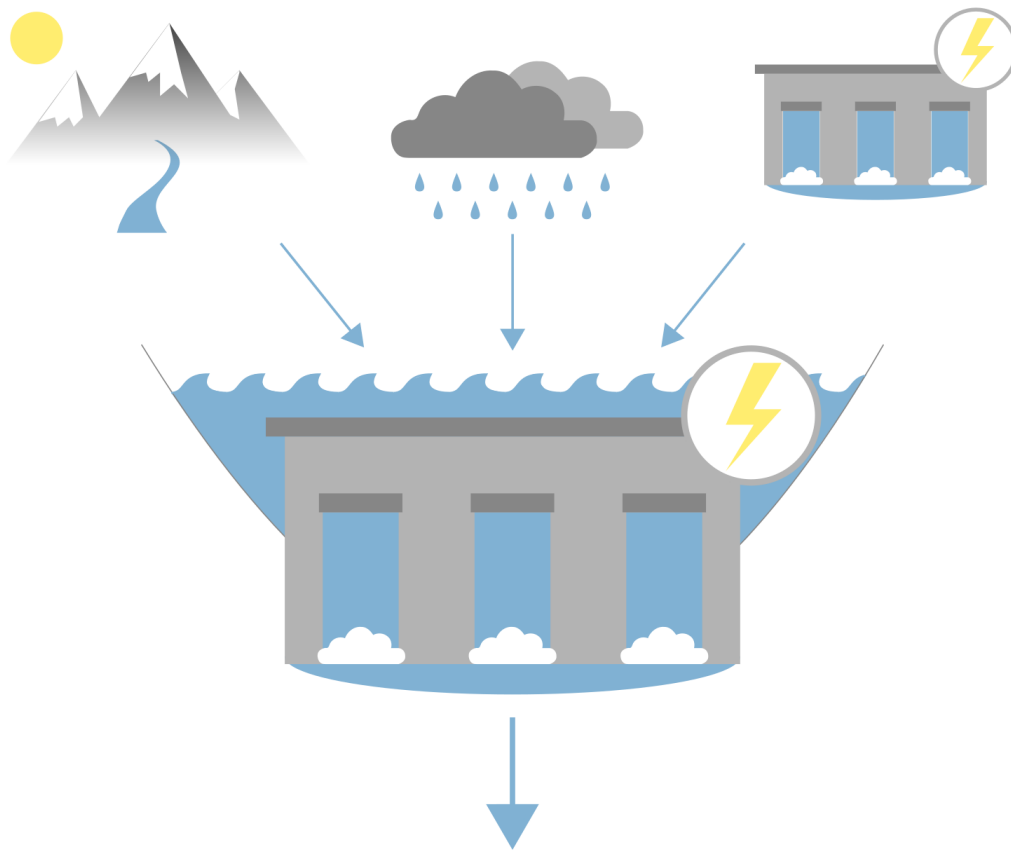


Figure 1.2: Factors causing the change of water level in the hydropower reservoir. Sources of the increase are shown in the upper row, from the left: melting snow accumulated in the surrounding higher-elevated terrain, direct rainfall and snowfall, inflow from the higher located hydropower plant. The lower part indicates the source of decreasing water level—the activity of the hydropower station located at the reservoir.

directly related to the water level; see Section 1.4). This would effectively allow for predicting the future power price, with the upper boundary of accuracy equal to the accuracy of the hydro producers' forecast. The data we can use for this purpose come in the shape of time series—vectors of values registered or measured on an arbitrary feature (like electricity price or water level of a reservoir), typically in regular time intervals [12].

It is important to remember, that each hydro producer operates multiple reservoirs, with different water storage and power generation capacities. Some reservoirs are used for long-term storage (can save water for drier seasons over multiple years), some have a much shorter filling/emptying cycle and are designed to respond to

short-term price signals. Power plants at each reservoir run according to their individual schedules, optimised jointly to maximise profits of the operating company. Hence, establishing relations between filling levels and future power price is not possible using short time intervals of data from a single reservoir. On the other hand, analysing several years of daily price records and several years of filling level data from tens of hydropower reservoirs is not a task that a human can handle, regardless of their expert domain knowledge or quantitative skills. However, in 2022 the obvious solution to the problem of finding patterns in the abundance of data is to employ artificial intelligence, and more specifically—deep learning.

Deep learning technology—a subdomain of machine learning methods based on artificial neural networks—is known to outperform traditional statistical methods in many tasks of time series forecasting, especially when the time series exhibit non-linear behaviour [13]. In this thesis, the statistical time series forecasting method ARIMAX is used to develop a baseline price prediction model. Next, we attempt to beat the ARIMAX prediction accuracy using a deep learning approach. More extensive theoretical background for the two time series forecasting methods can be found in Section 2.3.

1.2 Industrial partner

This master’s thesis has been developed in collaboration with EDInsights AS⁴.

Among other business lines, the company operates a pipeline that semi-automatically calculates water levels of the monitored reservoirs in the Nordics and south-eastern Europe, based on the satellite radar data provided by Sentinel-1 mission of the European Space Agency. The water levels are updated with a frequency between bi-weekly up to twice a week, depending on the Sentinel-1’s operationality and seasonal variation of radar image quality. EDInsights can then convert the water levels into water volumes (using parameters specific for each reservoir) and eventually calculate filling level values.

⁴<https://edinsights.no/>

1.3 Motivation

The motivation behind this work was to explore the landscape of potential novel applications of EDInsights' technology for monitoring water levels of hydropower reservoirs.

Information about the regional filling levels – an average state over a region – is publicly available in Norway. It is provided by NVE—The Norwegian Water Resources and Energy Directorate⁵. Filling levels on a single reservoir scale, in turn, are of strategical significance for the hydro producers and are restricted from public access for three months. EDInsights' technology offers more up-to-date information and therefore is of great interest to various hydropower producers, competing between each other. It enables them to track the entire hydropower system with greater granularity and thus optimise their production and increase revenue.

The hydro balance of the country where hydropower accounts for approximately 92% of total power production has also a large impact on the financial and physical power markets, both domestically and on the Nordic and European scale. Hence, market players such as traders are also vitally concerned with acquiring insights into hydropower producers' strategies as close to real-time as possible.

Having built the reservoir monitoring technology, EDInsights is looking into developing a new product which would leverage the unique information the company collects on a regular basis. Gathering feedback from current clients and acquiring opinions from market experts helped scope down the field of interest to what every person involved in the power production planning and trading business is interested in—the power price model.

The price model, albeit crucial, is not the sole factor taken into account in the complex mechanism of hydro production planning. Hence, we cannot expect that the filling levels will be a direct reflection of the price model, but we certainly can hope to recover trends of the price model that the companies were using in production planning (or rather its averaged version, as each company may have a slightly different model at their disposal). At this stage, two points are worth mentioning: firstly, production planners receive regular updates of the price model from the company's analysts. Secondly, even though the hydro producers have access to the best price forecasts existing on the market, extraordinary events may

⁵To keep control over the hydropower potential in the country NVE measures the water levels of nearly 500 most important reservoirs on a weekly basis [14].

happen which affect the global commodity prices in the short-, mid- or even long-term perspective. A war, pandemic, extreme weather, natural disaster or economic collapse are the examples of powerful circumstances that may completely disrupt even the most comprehensive price forecast. Should such circumstances occur, the hydro producer's model and the real (historical) price curve start diverging, despite the continuous model revision. There is a practical consequence of such divergence for the process of recovering the price model information from the filling levels. When training, validating and testing our model, we should be using the hydro producers' forecasts as ground truth values of the target variable. However, as third parties are not granted access to the hydro producers' forecasts, we are forced to use the real historical price data as the ground truth. As soon as it begins to significantly differ from the actual ground truth, it confuses the model we train. Despite this complication, it is still an interesting task to try and see what can be discovered by modern statistical and deep learning techniques from the data that are available.

1.4 Data

The input dataset consists of the price time series and multiple filling level time series.

There are a number of power price variants that are present on the financial and physical power markets (they are briefly described in Section 2.1). Since the objective of this work is to primarily utilise the filling levels information as input data (possibly supplementing it in the future with other data types), the price acting as target variable should have as direct a relationship with the filling levels as possible. As explained earlier, the filling level of the hydropower reservoir is the aftermath of the hydro producer pursuing their production strategy, which in turn is built upon a certain price model (more on this process in Section 2.2). For simplicity, in this work we assume that the Nordic system price (further also referred to as *SYS* or *SYS price*), is an average representative of multiple price components that drive the reservoir management optimisation. System price is given in EUR/MWh (the "/MWh" part is often assumed implicit and skipped when providing price values). The process of system price calculation and its role in the financial power market are described in Section 2.1.1.1. The historical data were obtained from Nord Pool [15], a pan-European power exchange.

Filling level denotes the degree to which the reservoir is full. Its values range from 0 to 1 (unitless), where 0 corresponds to the lowest regulated water level and 1

to the highest regulated water level⁶. EDInsights' pipeline has been monitoring the filling levels of a variable number of Norwegian hydropower reservoirs over different periods of time. For this master thesis, we decided to use filling level data calculated based on the historical ground truth water volume data provided by NVE. One reason for that is to be able to register the best possible performance of the developed model by excluding the effect of potential pipeline processing inaccuracies. Another is to possibly include information from the reservoirs which have not been so far monitored by EDInsights and which can potentially add value to the model. In the end, the delay in publishing the water volumes by NVE does not harm the process of developing the model, as it is not crucial to use the most up-to-date data for this purpose.

1.5 Problem statement

The main goal of this thesis is to evaluate the potential of deep learning methodology for recovering power price models from the filling levels of the hydropower reservoirs in Norway. In this process, we sought to provide answers to the following research questions:

- What magnitudes of forecasting performance can be achieved for various combinations of input data, modelling methods, forecasting time frames (short-/mid-/long-term) and hyperparameter setups?
- Does the deep learning approach, computationally more expensive than the traditional statistical method, provide a significantly better modelling outcome?

The accompanying goal is to determine best performing models for several prediction time frames; such outcome would enable EDInsights to further analyse their prospective applications and match them with potential users, such as business developers, hydro production planners or traders.

This master's project has an exploratory character—there was no a priori model to be outperformed, no established solution to be improved and no particular forecasting time frame indicated by market practitioners as of the highest importance.

⁶Every regulated reservoir has upper and lower limits of the water level. NVE is responsible for supervising the hydro producers and making sure they operate in accordance with their license [16].

Chapter 2

Theoretical background

2.1 Physical and financial power markets

The Norwegian power market is a free competition-based market (as opposed to state-run market type) and it has been maturing as such for over 30 years now [10]. Note that the deregulation from 1991 only pertains to electricity production and trading. Transmission and distribution of electricity function on a basis of natural monopoly¹ and are strictly regulated by the state to ensure rational management in the best interest of society [18]. Before we proceed to explaining different types of markets for power trading and how they are related to each other, there is an important physical aspect of the power system that needs to be mentioned. After electricity is generated and supplied to the grid by a power plant, it is no longer possible to distinguish between different deliveries flowing through the grid. It is not possible to track the physical “portion” of electricity the end user consumes to the plant that generated it. Electricity generated by all producers operating within the integrated pan-European market is sent to the grid which allows transmission between multiple countries. Producers are paid for the volume they deliver to this pool, and end users pay for the amount they consume [8]. The supply to and consumption from the pool need to be balanced at all times (every single second). Securing this balance is a responsibility of Transmission System

¹There are high fixed costs associated with the grid operations (development, maintenance) and it would simply be inefficient to allow for any redundancy in the electrical network. The reader can find more information about Norwegian electricity grid infrastructure in [17].

Operators (TSOs)² [17].

The main difference between physical and financial power markets is that clearing of the physical markets results in the physical delivery of electricity from the seller to the buyer, whereas trading on the financial markets is based on purely financial settlements [9]. Let us first describe the characteristics of the physical markets, as some of them will be useful to later discuss the financial markets.

2.1.1 Physical power markets

The physical power market can be divided with respect to the participants into:

- wholesale market—occupied by large players, such as: power producers and suppliers, brokers, energy companies and large industrial customers;
- end-user market—where power suppliers meet smaller end users, such as: private households, small and medium-sized businesses and industrial customers [8].

In this thesis we are only interested in the wholesale market, as this is the trading arena for hydropower producing companies. As shown in Figure 2.1, there are three wholesale physical power markets in Norway:

- day-ahead market (see Section 2.1.1.1)
- intraday market (see Section 2.1.1.2)
- balancing market (see Section 2.1.1.3)

While spot³ markets—the day-ahead and the intraday—are run by Nord Pool⁴, the balancing market is operated by the Norwegian TSO—Statnett⁵.

²There is typically one TSO per country, with few exceptions. See <https://www.entsoe.eu/about/inside-entsoe/members/> for European TSOs.

³Spot market is a market where delivery of product happens shortly after the trade has been closed—in case of electricity market the next day at the latest [19].

⁴Nord Pool AS (<https://www.nordpoolgroup.com/en/>) is a Nominated Electricity Market Operator (NEMO)—a power exchange. Originally served as power exchange for Nordic region, currently operates in several European countries.

⁵<https://www.statnett.no/en/>

FINANCIAL MARKETS	SPOT MARKETS	RESERVE MARKETS
Years/months/weeks ahead	Previous day	Delivery day
Financial	Day-ahead	Intraday
Financial settlement	Physical delivery	Physical delivery
NASDAQ	NORD POOL	TSO

Figure 2.1: Power market types. They are ordered along the timeline with respect to when the trading happens. Trading future or forward contracts on the financial markets (Section 2.1.2) starts years, months or weeks before physical delivery of electricity takes place, results in purely financial settlement between participants and in the Nordics it is hosted by NASDAQ. Trading on the spot markets—day-ahead (Section 2.1.1.1) and intraday (Section 2.1.1.2)—happens one day before and on the delivery day, respectively. It results in physical delivery of electricity and Nord Pool is the platform that provides the services for the Nordic region. Balancing market (Section 2.1.1.3) is also active on the physical delivery day (hence highlighted in green together with the intraday market) and it is operated by the TSOs.

2.1.1.1 Day-ahead market

Single Day-ahead Coupling (SDAC) is a mechanism that integrates European wholesale power markets with the aim of improving their efficiency, liquidity and transparency. It is supposed to ensure the optimal use of power infrastructure and electricity generation resources across Europe, improve security of supplies and reduce volatility of electricity prices. SDAC is based on the “Price Coupling of Regions” concept founded jointly by a group of European NEMOs. The main achievement of this collaboration is the development of the Pan-European Hybrid Electricity Market Integration Algorithm (EUPHEMIA) [20][21]. As stated in the EUPHEMIA Public Description document [21] issued by NEMO Committee⁶, the algorithm calculates “energy allocation and electricity prices across Europe, maximizing the overall welfare and increasing the transparency of the computation of prices and flows.” Price Coupling of Regions currently comprises most of the European countries [20].

⁶<https://www.nemo-committee.eu/nemo-committee>

The day-ahead market follows exactly the same routine every day in every country within the integrated pan-European market. Market activity during a given day pertains to physical delivery of electricity during the subsequent day (hence the name “day-ahead”). Between 08:00 and 12:00 sellers (e.g. hydro producers) register their offers (sell orders) and buyers register their bids (purchase orders) in the Nord Pool trading system. By 10:00 TSOs must publish transmission capacities for each bidding area in the market [8]. Bidding areas (also referred to as *bidding zones* or *price areas*) are determined by limitations in the transmission grid (often termed *bottlenecks* or *congestions*). If there is physically not enough transmission capacity to import and export electricity freely from one location to another, then these locations typically lie in different bidding zones—there must be no bottlenecks in the grid inside a bidding zone [9]. The shape of bidding zones is decided by local TSOs. Some countries constitute one zone, some are divided into several. As of June 2022, Norway has five bidding zones [22] (see Figure 2.2). Coming back to the daily cycle of the day-ahead market, the auction closes at

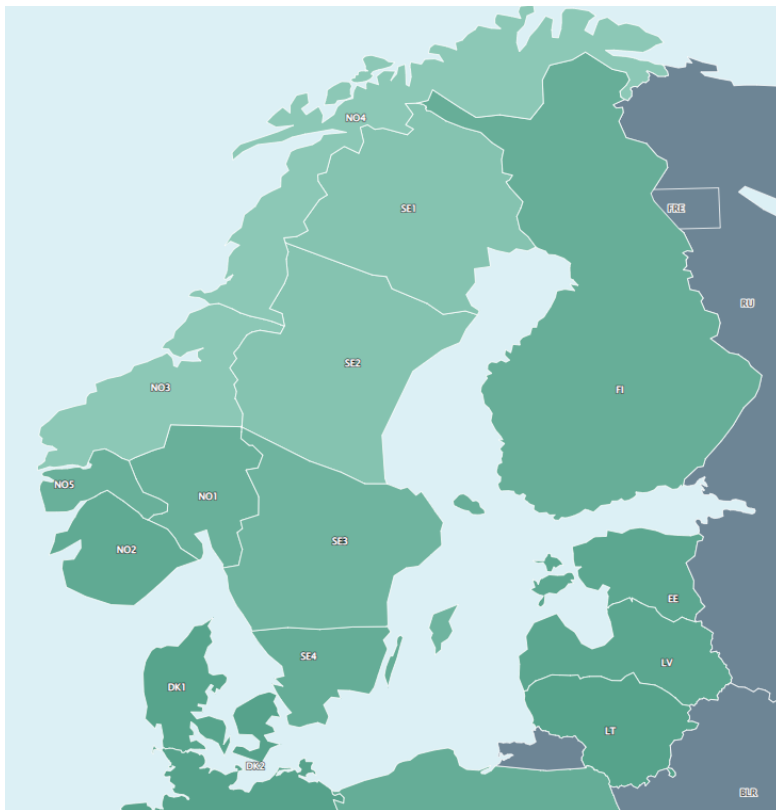


Figure 2.2: Bidding zones in the Nordic and Baltic regions, as of June 2022. Sourced at <https://www.nordpoolgroup.com/en/Market-data1/#/nordic/map> and zoomed in to the Nordic and Baltic regions, with permission from Nord Pool AS.

12:00. Between 12:00 and 13:00 all registered purchase and sell orders are used by EUPHEMIA to calculate prices for each hour of the following day, for each bidding zone. The prices are typically announced before 13:00 and are immediately available on the power exchanges' websites, e.g. [Nord Pool's market data](#). The trades are invoiced in the afternoon.

The bids and offers submitted in the auction are used to build supply and demand curves for each price area (for each delivery hour). The intersection of the two curves determines the prices for each area-delivery hour combination. As presented in Figure 2.3, in the areas with surplus of electricity the prices will typically be lower than in the ones with deficit. The electricity will flow from the cheaper area to the more expensive one. In general, the algorithm is trying to make the prices from different areas converge, utilising transmission capacity to the maximum. If the capacity set by the TSO is sufficient to balance the supply and demand in two areas, the prices will be identical in these areas, otherwise—they will differ [23].

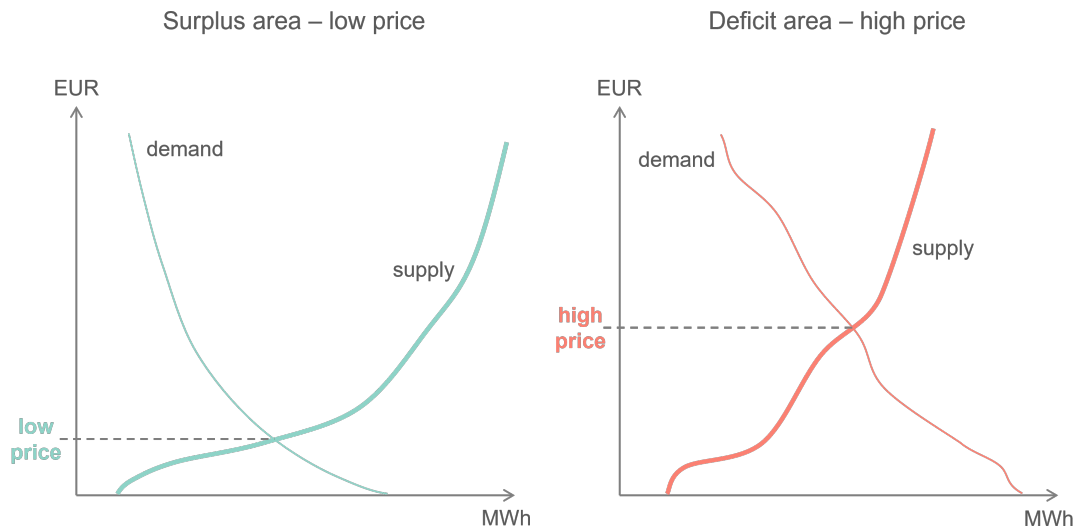


Figure 2.3: Price formation in the electricity surplus and deficit areas. Price is determined by the intersection of the supply and demand curves. It is lower in the surplus area, compared to the deficit area. Based on the original illustration from [23], redrawn with permission from Nord Pool AS.

The prices determined by the supply and demand curves ensure that market equilibrium has been reached at the lowest possible cost to society [8]. This is related to the fact that, in compliance with the *merit order* rule, the market operator orders supply bids in the ascending price order—the electricity from the cheapest source (lowest marginal cost of production) is sold first [24]. Hence, the price

that results in the market equilibrium is determined by the cost of production of the last unit of power that is necessary to satisfy the market demand—it usually corresponds to the offer from the plant producing electricity from fossil fuels (see Figure 2.4). It is worth noting that following the merit order is also beneficial from the environmental point of view, as the cheapest electricity is, in fact, the “clean” one (wind, solar, hydro, nuclear).

Among key factors forming the day-ahead price we should mention:

- those affecting the supply: fossil fuel prices, operationality of plants, hydro balance, wind and sun conditions, regulations, political situation;
- those affecting the demand: season of the year, weather conditions, industrial activity;
- those related to transmission capacity, e.g., availability of interconnectors, grid operationality [25].

Last but not least, the Nordic system price—particularly interesting from the perspective of this thesis—is also a product of the day-ahead market auction. Similarly to area prices, it is also calculated with use of EUPHEMIA, but locally at Nord Pool. The system price can be described as a theoretical price, as it is calculated based on the assumption that there is an infinite transmission capacity between bidding zones in the Nordic countries (Norway, Denmark, Sweden and Finland)—in fact, they are simply treated as one big bidding zone. The rest of the European bidding areas and the flows and restrictions between them remain the same as for the area price calculations. The system price is used as clearing reference price for the financial contracts traded in the Nordic region [23][8] (for more information about financial market see Section 2.1.2). As mentioned in Section 1.4, in this thesis the system price is treated as a generalised Nordic power price for a given point in time, as its value reflects the spread of area prices in the Nordic region.

2.1.1.2 Intraday market

The intraday market enables the participants to adjust their positions from the day-ahead market [8]. Let us imagine the following situation—after the day-ahead auction has been closed, a wind power producer gets an updated weather forecast. The next day is going to be more windy than what was assumed when submitting

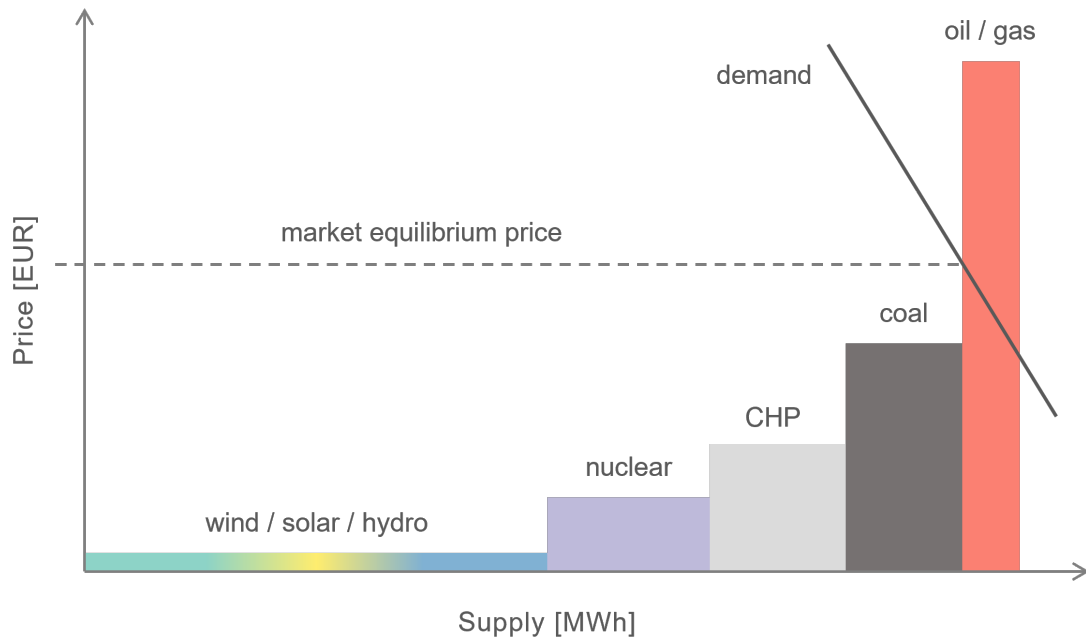


Figure 2.4: Electricity price formation according to the merit order model. Electricity produced from various sources forms a discrete supply curve, with the height of the bars corresponding to the marginal costs of producing electricity from the given sources. Electricity from the cheapest sources (renewables and nuclear) is produced and sold first, then the more expensive sources (CHP—Combined Heat and Power, oil, gas) step in to satisfy the demand. Market equilibrium is reached at the intersection of the supply and demand. Note that this is a rough visualisation and may not reflect the exact relations between the marginal production costs of different sources. Based on the original illustration from [25], redrawn with permission from Nord Pool AS.

the offer to the day-ahead market. Thus, the producer will be able to generate and sell more electricity. Another scenario involves a power supplier, who also received a weather forecast update—it is going to be colder, people will want to switch on the heaters. The supplier needs to buy more electricity from the producers to accommodate the additional demand from the end users side. The wind power producer from the first example and the power supplier from the second example can achieve their desired balance through intraday trading. In the intraday market trades are executed continuously, 24 hours per day, 7 days in the week. The length of contracts (the unit of time for which the purchased electricity will be delivered) and the time window during which the trade must be made depend on the market operator. The latter typically starts after the day-ahead market for the given delivery day has been cleared and ends 5-60 minutes before the physical delivery begins. The prices are set in a “pay-as-bid” process—prices for different

transactions are independent of each other (unlike in the day-ahead market where all aggregated bids and offers form a basis of price calculation) [26].

2.1.1.3 Balancing market

The balancing market is used as a last resort to ensure the physical balance between the supply and demand in the electrical grid. This market works on the basis of agreements between the TSOs and major power producers and consumers, who are willing to adjust their production or consumption instantaneously depending on what is needed to maintain the balance in the grid at a frequency of 50 Hz. The agreements are contracted a priori to ensure that the TSO can immediately activate the available reserves, should such need occur as a result of e.g. sudden failure of a transmission line or production unit. There are several tiers of reserves (primary, secondary, tertiary) which can be activated to secure the balance in the grid. The participants have no option to decide themselves when to sell or buy electricity in the balancing market—the reserves may or may not be activated at each time. However, the prices at the balancing market for those who offered their flexibility are typically more favourable than those one can trade for in the spot markets. Thus, provided a good imbalance forecast is in place, the risk and flexibility might be financially rewarded—at the expense of those who caused the imbalance in the grid [9][8][27].

2.1.2 Financial power markets

The electricity price on spot markets is driven by a series of weather, geopolitical, social and operational factors. There are instruments on the power markets that allow for managing the risk associated with prices volatility. One way of hedging the electricity prices available for large players (e.g. power producers, power suppliers or businesses with high demand for electricity) is participation in the financial power markets. It involves buying or selling long-term financial contracts which can be traded for up to six years ahead, split into daily, weekly, monthly, quarterly or yearly delivery periods. There are several types of financial contracts:

- forward and future contracts—result in cash settlement that reflects the difference between the price a buyer and seller agreed on (for a given power volume and delivery period) and the reference price from the spot market; as mentioned earlier, the Nordic market uses the system price as reference

for settling financial contracts. The difference between these two contracts is that forward contracts are only settled when they expire, and futures are settled both during trading and delivery periods [8].

- electricity price area differentials (EPAD)—traded to hedge the difference between the area prices and the system price (reference for forward and future contracts); they are getting more and more popular especially among players who are exposed to high area prices in the bidding zone NO1 in Norway [28].
- options—European options with forward contracts as the underlying product [9].

If a financial market participant is interested in receiving an actual physical delivery of electricity (as in the case of a power supplier, who needs to deliver it further to the end users), they still need to participate in the spot market and buy the required volumes there. The financial market settlement simply covers a difference between the price of the financial contract and the spot price. If the supplier bought the financial contract in advance for more than the actual spot price, they suffer loss. Otherwise, the hedge is successful.

Financial markets are also used for speculation. Speculators seek to profit from short-term price movements and close their positions before they expire. Although often negatively (and wrongly) associated with market disruptions, they in fact increase markets' liquidity by increasing the volumes of trades [29][30].

In the Nordics, most of the financial trading takes places on the Nasdaq OMX⁷ exchange.

In Section 2.2 we will take a closer look at some of the most important players on the power markets—hydropower producers.

2.2 Hydropower production planning

According to NVE, there are over 1600 hydropower plants operating in Norway. They are distributed across around 1400 regulated reservoirs.

⁷Nasdaq OMX Commodities AS—<http://www.nasdaqomxnordic.com/>.

Hydropower producers have an advantage over other renewable energy producers, originating from their unique capability to store water—their energy source. They are flexible in terms of deciding when to generate electricity, when to pause the production and when to relaunch it – and they can do it at a relatively low cost and short time [31]. In fact, the costs of turbine and generator start-up are often treated as negligible by scientific publications covering the topic of hydropower reservoir management [32]. This flexibility, however, carries a challenge of deciding how to schedule the production to maximise the profit.

Production planners are supported in their decision making process by complex models which have been developed for nearly a century now. EOPS (One-area Power-market Simulator) [33] created by SINTEF is one of the most popular commercial models in the Nordics, suited for long- and medium-term production planning. The models are founded on a number of assumptions, e.g. that a given plant’s activity is sufficiently small not to impact the overall market situation, or that the company only trades in the spot market [32]. Some algorithms consider participating in both spot and balancing markets in a sequential manner—ignoring the effect of the latter one on the former one and optimising balancing bids only after the spot market has been cleared. Yet another ones allow for coordinating trades in the two markets [34]. Mathematical and computational concepts most commonly used in such optimisation problems include Markov process, ARIMA process and stochastic dynamic programming.

Both [34] and [32] highlight the importance of forecasting multiple factors for the optimisation task and point out the complexity of dependencies between them. The most important factors are prices and power flows (for different time intervals, geographical locations and market types), as well as weather conditions and water inflow to the reservoir. Last but not least, there are restrictions specific to each reservoir to be accounted for when planning the production, such as lowest and highest regulated levels, dam and turbine parameters, seasonal regulations.

The end goal of the complex optimisation task is to produce electricity when the price is high. Note that the hydro producers cannot defer the production in order to reduce supply and consequently induce a price rise—this qualifies as market manipulation and it is illegal. There are regulations and guidelines in place to ensure it does not happen, even though the presence of a grey zone seems unavoidable.

In a workshop about physical and financial power markets organised by Nord Pool Academy in January 2022, a simplified illustration of the production planning

outcome was showcased by a representative from Statkraft⁸. Figure 2.5 (recreated based on [35]) illustrates a situation where the water resources available for a certain reservoir between April and October are predicted to suffice for 75 full days of production. This amount takes into account the reservoir content at the beginning of the planning period and the total expected inflow to the reservoir from April until the end of October. Another element of the sketched situation is the electricity price prognosis, predicted with a discrete step of one month (30 days). The 75 days of production can be freely distributed throughout the entire planning period. Hence, to optimise revenue the production is scheduled for 75 highest-priced days—30 days of October, 30 days of September and 15 days of April.

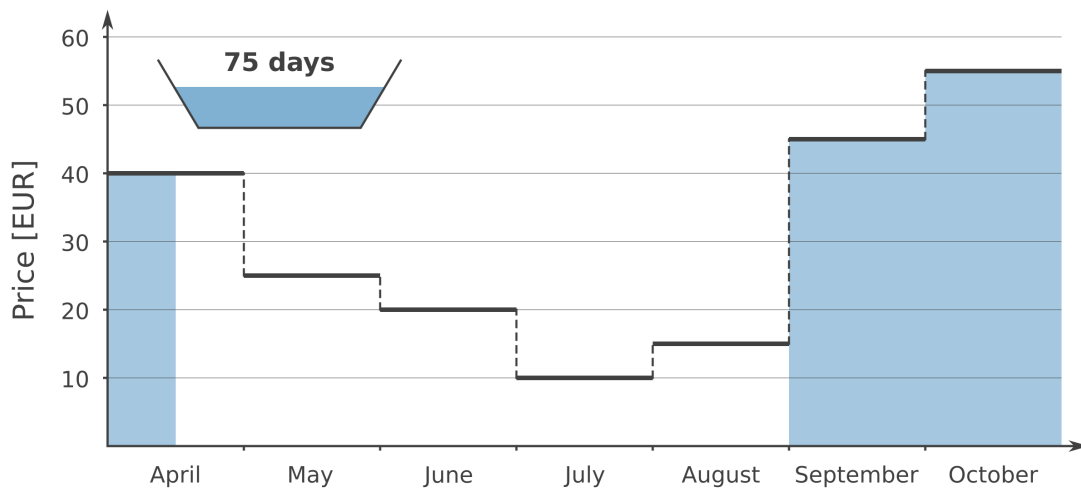


Figure 2.5: Hydropower production planning outcome. In the period between April and October the reservoir is estimated to have enough water for 75 full days of production. The production will be run when the forecasted power price is the highest, i.e. 30 days of October, 30 days of September and 15 days of April. Based on the original illustration from [35], redrawn with permission from Nord Pool AS.

⁸Statkraft AS (<https://www.statkraft.com/>) is a Norwegian state-owned hydropower company and the largest generator of renewable energy in Europe.

2.3 Time series forecasting

2.3.1 Time series

Time series data, such as stock prices, regular weather measurements or voice recordings, are a type of *sequential* data [36]. This means that subsequent observations in the dataset are ordered and dependent on each other—randomising the order of their appearance would result in the loss of information. In case of time series, the order is related to the time stamp of an observation. However, there are types of sequential data which do not have time dimension (e.g. DNA sequences). As far as non-sequential data are considered, the samples in the dataset are assumed *independent and identically distributed* (i.i.d). An example of non-sequential data can be a registry of students and their grades. The performance of one student does not, in principle, affect the performance of another one. In contrast, time series data are not considered independent—in fact, data measured at consecutive time points are often highly correlated.

Time series data can be classified as:

- univariate— $(x_t)_{t \in T} \in \mathbb{R}$
- or multivariate— $(x_t)_{t \in T} \in \mathbb{R}^n$,

where T is the time domain. In the first case, the time series consists of only one variable tracked over time—a sequence of measurements from one sensor, or a history of one company’s stock price. In the second case, we deal with a set of variables which might affect one another, in a linear or non-linear way. An example can be a record of weather data from a meteorological station, where multiple quantities like air temperature, atmospheric pressure, humidity, wind speed, etc. are measured simultaneously in regular time intervals over a period of time [37]. Note that although time series are assumed to be sampled in equidistant time intervals, this is often not the case for real-world data. Hence, interpolation between irregular time steps is a common pre-processing method.

Time series forecasting is a process of predicting future values of the series, based on its recent and current values. As pointed out in [38], historically, computer simulation methods were used in forecasting tasks. However, statistical and machine learning techniques tend to outperform the simulation methods in terms of both accuracy and time efficiency. They are based on the process of fitting a model to

the existing data and extrapolating it for predicting future values. In case of a multivariate time series, it is possible to fit the model using multiple input variables (*exogenous* variables $(x_t)_{t \in T}$) with a goal of predicting the future values of one of them (the output variable $(y_t)_{t \in T}$; note that both x_t and y_t are defined on the same time scale $t \in T$). This is the approach that will be explored in this thesis, as we attempt to forecast the future values of the system price variable using the recent and current values of the system price and filling level variables.

2.3.2 Statistical approach

In every prediction task, it is a good practise to first establish simple benchmark models. Benchmark models are the most intuitive ones, can be computed at a low computational cost, and are used as reference to assess the performance of the more complex models. Benchmark models for statistical forecasting methods include those computed with the average, naïve, seasonal naïve or drift methods [39, Chapter 3.1].

Taking one step towards more advanced forecasting methods—we have multiple regression models, in which the forecasted variable y is assumed to have a linear relationship with k independent predictor variables x_1, \dots, x_k at each time step t :

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \dots + \beta_k x_{k,t} + \varepsilon_t, \quad (2.1)$$

where ε_t is the random error (following an i.i.d Gaussian distribution with mean equal to 0) and β_0, \dots, β_k are the coefficients corresponding to the effects of the respective predictors on the forecasted variable [39, Chapter 5.1]. After fitting the regression model we can use it to predict the values of y for the set of predictor variables $x_{1,t}, \dots, x_{k,t}$ for t from the range that the model has been fitted on, or for the future time stamps [39, Chapter 5.6]. If the assumption of linear relationship between the forecasted and predictor variables is violated, there is also a way to model the non-linear relationship by transforming the forecasted and/or predictor variables with the natural logarithm function [39, Chapter 5.8].

Another, more advanced approach to the statistical time series forecasting is offered by a group of so-called autoregressive models, which build upon a concept of autocorrelation of a series [39, Chapter 8]. Autocorrelation is a measure of similarity between the series and its lagged (i.e. shifted along time axis) version [39, Chapter 2.8]. For lag k , the autocorrelation coefficient r_k quantifies the similarity between

y_t and y_{t-k} according to the formula:

$$r_k = \frac{\sum_{t=k+1}^{t_{max}} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^T (y_t - \bar{y})^2}, \quad (2.2)$$

where t_{max} denotes the length of the series.

Before delving into the topic of autoregressive models, some major concepts of time series analysis need to be introduced. In addition to the serial dependence between elements, described in Section 2.3.1 as the main property of sequential data, a time series may exhibit the following characteristics [12]:

- trend—a general, consistently upward or downward direction of changes;
- seasonality—periodical fluctuation;
- stationarity—statistical properties (e.g. mean, variance, autocorrelation) remaining constant over time.

Time series analysis is a process of decomposing an observed time series into the trend, seasonal and residual components. Residuals comprise the unexplained part of data variability, the part not accounted for by the trend and seasonal components [38]. If a time series exhibits trend or seasonality, it is considered non-stationary. In both cases it is possible to transform the series to stationary by *differencing* it [39, Chapter 8.1]. Differencing can be performed:

- to eliminate the trend—by computing differences between consecutive observations:

$$y'_t = y_t - y_{t-1} \quad (2.3)$$

In case the series is still not stationary after the first differencing, a second-order differencing can be applied:

$$y''_t = y'_t - y'_{t-1} \quad (2.4)$$

- to remove seasonality—by computing differences between observations lagged by m periods, where m is the number of seasons after which the seasonal cycle repeats itself (e.g. for monthly data $m = 12$):

$$y'_t = y_t - y_{t-m} \quad (2.5)$$

While in the multiple regression models the y variable is forecasted using a linear combination of predictor variables, basic autoregressive models use a linear combination of past values of the forecasted variable [39, Chapter 8.3] (past values of the predictor variables can also be used, in the extended variants of autoregressive models). Autoregression model of order p , also referred to as $AR(p)$ model, takes the form of:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t, \quad (2.6)$$

where c, ϕ_1, \dots, ϕ_p are the autoregression coefficients and ε_t is white noise. Another variant of a regression-like model is a moving average model, which is formulated as a linear combination of past forecast errors [39, Chapter 8.4]. Moving average model of order q , $MA(q)$, can be described by the following formula:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q}, \quad (2.7)$$

where $c, \theta_1, \dots, \theta_q$ are the moving average coefficients and ε_t is white noise. Combining the $AR(p)$ and $MA(q)$ models results in the $ARMA(p, q)$ model—Autoregressive Moving Average model of order p, q , which can be applied to model stationary time series [39, Chapter 8.5]. It is possible to model non-stationary time series with a modification of ARMA approach called ARIMA—Autoregressive Integrated Moving Average model, formulated as:

$$y'_t = c + \phi_1 y'_{t-1} + \dots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \quad (2.8)$$

where y'_t is the differenced series. ARIMA requires the differencing step to be performed beforehand, and an *integration* step—the reverse of differencing—to be performed after the model fitting. More specifically, the process of fitting an $ARIMA(p, d, q)$ model involves:

1. performing a d -fold differencing step (where d denotes the order of differencing),
2. fitting the $ARMA(p, q)$ model,
3. performing a d -fold integration step.

The optimal combination of p, q , and d parameters (where p is the number of AR terms, d is the number of non-seasonal differencing passes and q is the number of lagged forecast errors) can be found in the process of an automated parameter grid search [40]. However, initial indications regarding the optimal p and q values can sometimes be deduced from the Partial autocorrelation function (PACF) and

Autocorrelation function (ACF) plots, respectively. ACF is simply a set of autocorrelation values between the original series and the lagged series, for different time lags. PACF, on the other hand, consists of autocorrelation values for different time lags adjusted for the effects of all intermediate lags [39, Chapter 8.5].

The ARIMA model can be further extended to ARIMAX—Autoregressive Integrated Moving Average model with exogenous variables. ARIMAX incorporates additional input variables in the task of model fitting and forecasting of the target variable. Exploiting additional information sources which are known to influence the forecasted variable has a potential to increase the modelling capability and improve prediction accuracy [41].

Compared with more complex time series forecasting methods (such as those based on the machine learning concept, described in Section 2.3.3), the advantage of statistical methods originates from their limited number of parameters and relatively easily interpretable predictions [41]. However, estimating optimal parameters for such models is not trivial.

2.3.3 Deep learning approach

2.3.3.1 Deep learning as a branch of machine learning

Deep learning (DL) is a branch of *machine learning* (ML), which, in turn, is a subdomain of *artificial intelligence* (AI).

François Chollet—creator of the Keras deep learning library—points out in his book [37] the reasons behind deep learning not taking over the reins of the machine learning world until the second decade of the 21st century. Even though the key deep learning concepts used for computer vision or time series forecasting tasks were developed already in the 90's, certain bottlenecks (poor internet, lack of data, insufficient hardware) needed to be solved for this discipline to start advancing at the current speed.

To appreciate the importance of data for deep learning (or machine learning, in general), one should understand the difference between classical programming and machine learning. As sketched in Figure 2.6, the former is based on laying out a set of explicit rules for the computer to teach it how to perform certain tasks on the input data and return answers. The latter, requires feeding both the input data and the known answers into the computer, thus allowing the machine to decipher

and learn the rules on its own. In the next step, the computer should be able to apply the rules to the new input data and return answers. This part is analogous to the classical programming flow—the difference lies in the process of learning the rules by the computer, usually referred to as *training*. It is only possible when a sufficient amount of input data (including answers) is available. Note that when input data are available together with answers, we deal with a *supervised* machine learning. There also exist *unsupervised* machine learning algorithms which learn patterns from unlabeled input data (no answers available with the input dataset), as well as *reinforcement* learning algorithms, but these two types are not applicable in this thesis.

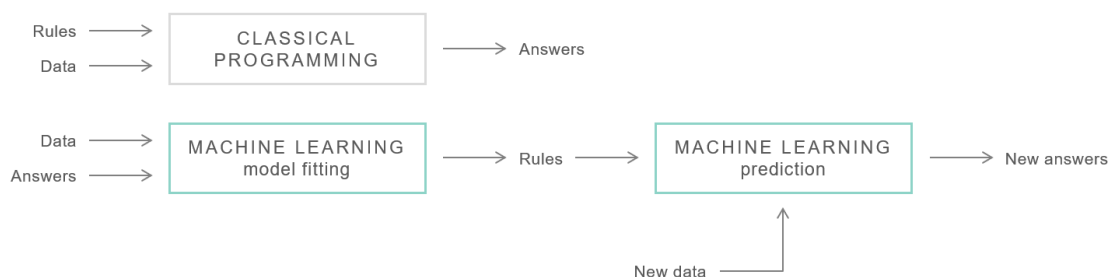


Figure 2.6: Difference between classical programming and machine learning workflows. Redrawn based on a figure from [37, Chapter 1.1.2].

Supervised machine learning problems can be further grouped into [36, Chapter 1]:

- classification tasks—where the expected outcome of prediction is a discrete class label (e.g. what animal species the input image represents, or whether a specific patient should be diagnosed with a disease or not, based on the symptoms)
- regression tasks—where the expected outcome is continuous (e.g. a price of a house predicted based on the factors such as location, area, public transport proximity, etc.)

The focus of this thesis—time series forecasting—can be classified as a supervised regression problem.

In the field of supervised machine learning, there is a wide spectrum of *shallow* ML methods, suited for solving both classification and regression problems. Popular

algorithms belonging to this group include single-layer neural networks (e.g. linear regression, logistic regression), kernel methods (e.g. support vector machines), decision trees or k-nearest neighbours models [36, Chapter 3 & Chapter 10]. The shallow ML methods *can* be found to perform better than deep learning methods in specific tasks, particularly when one does not have a high-performance computing hardware and an abundance of labelled training data at their disposal [42]. However, the major advantage of using deep learning over any shallow ML method is that it fully automates one of the most crucial steps in the ML workflow—feature engineering. Deep learning is considered an end-to-end approach, as it extracts and learns the important features by itself, from the the raw input data [37, Chapter 1]. Furthermore, there is empirical evidence that the DL methods tend to outperform shallow ML methods in the time series forecasting problems [43], [44], as well as in the computer vision and natural language processing tasks [37, Chapter 1].

Deep learning is based on the concept of *neural networks* (NN) built of multiple *layers*. The number of layers is the *depth* of a network. This is where *deep* learning takes a part of its name from and where the difference lies between the shallow and deep learning methods. An extensive explanation of how to understand the stacked layers of a neural network as increasingly meaningful *representations* of data is provided by Chollet in [37, Chapter 1]. There are different types of layers which can serve as building blocks for the deep learning architectures. They will be described in more details in the following sections, together with an outline of a typical machine learning workflow.

2.3.3.2 Single-layer artificial neural network

The concept of an *artificial neural network* (ANN) was first invented in the 1940s, in an attempt to emulate the human brain’s ability to solve complex problems. Over the years it has developed as a mathematical framework straying from its neurobiological inspiration, such that it is not correct to claim that the ANN learning mechanism mimics the one performed by human brain [37, Chapter 1].

When talking about ANNs, one usually has in mind a multilayer neural networks. However, to understand the multilayer variant, let us first explain a general single-layer NN architecture on the example of the *Adaptive Linear Neuron* (Adaline) algorithm, used for binary classification tasks (the output labels are either -1 or 1). The process of training the Adaline algorithm is illustrated by Figure 2.7 and can be summarised in the following steps [36, Chapter 2]:

1. *Weights* (trainable parameters of the model) in vector \mathbf{w} are initialised to 0

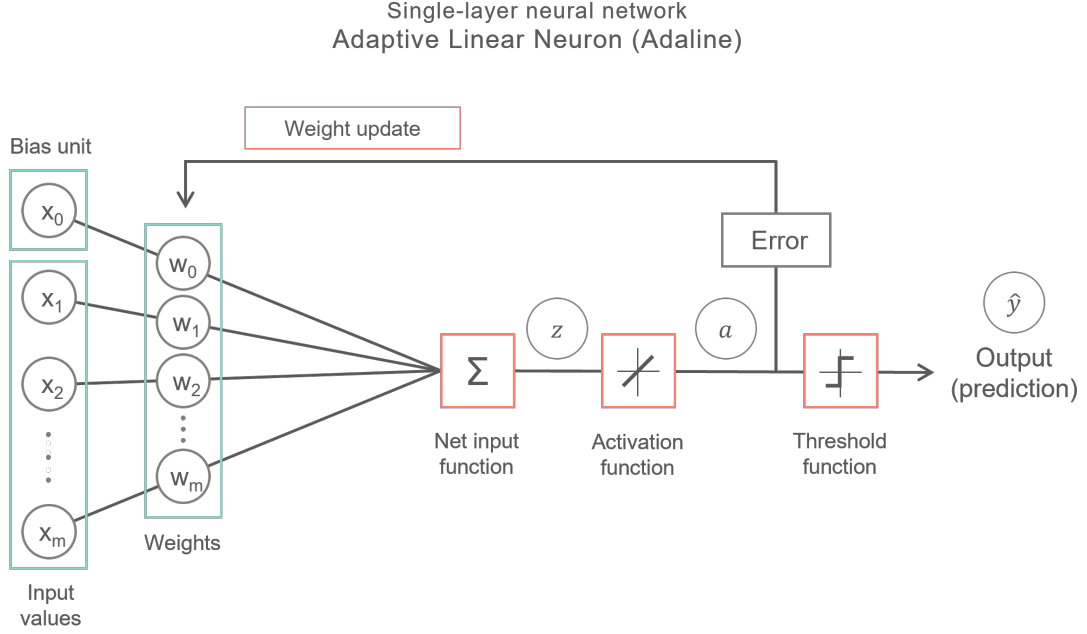


Figure 2.7: Single-layer neural network architecture on the example of the Adaptive Linear Neuron (Adaline) algorithm. Redrawn based on figures from [36, Chapter 2 & Chapter 12] under MIT license.

or small random numbers.

2. *Net input*, z , is calculated as a linear combination of \mathbf{w} and \mathbf{x} , where \mathbf{x} is a vector of input data (corresponds to one observation from the input dataset) consisting of m feature variables (x_1, x_2, \dots, x_m) and the *bias*⁹ unit x_0 :

$$z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \mathbf{w}^T \mathbf{x} \quad (2.9)$$

3. The net input is transformed by the *activation function* $\phi(z)$ into *activation* a :

$$a = \phi(z) = \phi(\mathbf{w}^T \mathbf{x}) \quad (2.10)$$

In case of Adaline algorithm, the activation function is a linear identity function of the net input: $\phi(z) = z$.

4. At this stage, the actual *learning* starts. There is an objective function that needs to be optimised—specifically, in the supervised ML it is called the *cost function*¹⁰ and it has to be minimised. This is done through the iterative

⁹Bias is an additional input element and its value is always set to 1 [36, Chapter 2].

¹⁰Cost function is also referred to as *loss function* or *error function*—for simplicity, in this thesis we disregard the subtle differences between these terms.

process of adjusting the weights. An update of the weight vector is defined as:

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad (2.11)$$

where:

$$\Delta\mathbf{w} = -\eta\nabla J(\mathbf{w}) \quad (2.12)$$

In Equation (2.12), η is the *learning rate* and $\nabla J(\mathbf{w})$ is the gradient of the cost function $J(\mathbf{w})$. A basic optimisation algorithm used to find the set of weights that minimise the cost function is called *gradient descent*. The underlying idea, visualised in Figure 2.8, is to take a step in the negative direction of the cost function's gradient, to find the cost function's minimum—preferably the global minimum, as a local minimum would limit the algorithm's prediction accuracy. The learning rate determines the size of the step—the goal is to strike a balance between the speed of learning and the risk of overshooting the global minimum. In case of Adaline, $J(\mathbf{w})$ is defined

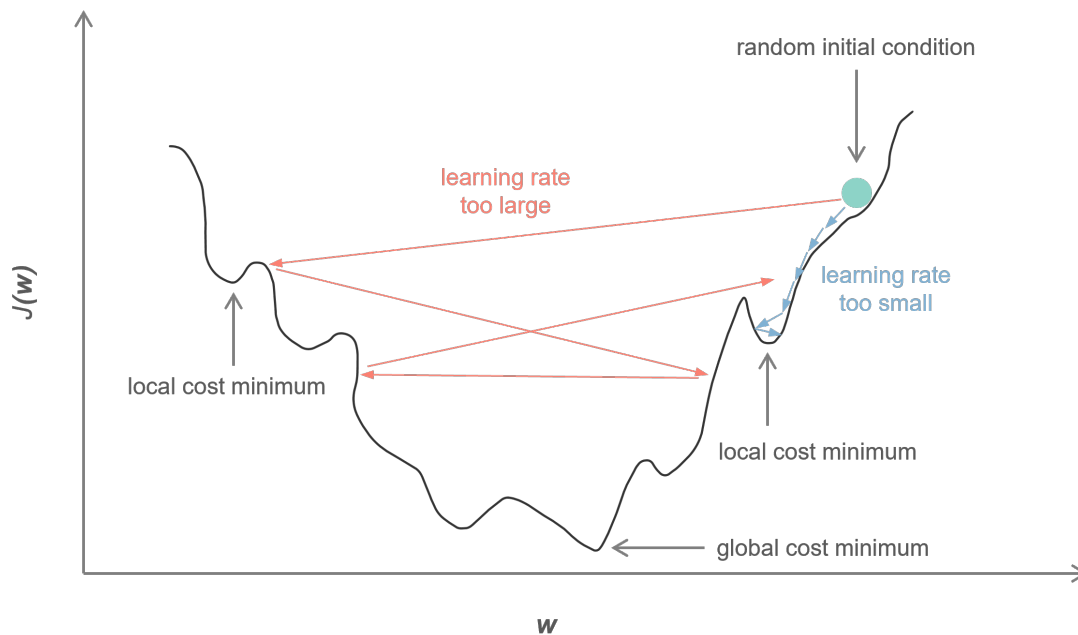


Figure 2.8: Simplified gradient descent concept. Starting from randomly initialised weights, the algorithm attempts to find the global minimum of the cost function. If the steps it takes (determined by the learning rate) are too small, it may get trapped in the local minimum. If too big, it may not find the global minimum either. Redrawn based on figures from [36, Chapter 2 & Chapter 12] under MIT license.

with respect to the weight parameters as a sum of squared errors (SSE)

between the true label (target variable y) and the activation ($a^{(i)} = \phi(z^{(i)})$), over i observations in the input dataset:

$$J(\mathbf{w}) = \frac{1}{2} \sum_i (y^{(i)} - a^{(i)})^2 \quad (2.13)$$

The gradient of the cost function $J(\mathbf{w})$ can be computed by taking its partial derivatives with respect to each weight, so that the final formula for the update of weight w_j is given by:

$$\Delta w_j = -\eta \frac{\partial J(\mathbf{w})}{\partial w_j} = \eta \sum_i (y^{(i)} - a^{(i)}) x_j^{(i)} \quad (2.14)$$

5. After each update of the weight vector, steps 1-4 are repeated with the new weights, until some criterion for stopping the training process is met—for instance, a maximum number of *epochs* (passes over the entire training dataset), that was arbitrarily set at launch, has been reached. Ideally, the training process stops when the algorithm has converged—the cost function’s minimum has been hit and all weight updates are evaluated to 0, for a given epoch.
6. After the end of the training process, the final (continuous) activations are passed to the *threshold function* which generates the binary predictions. For Adaline, the function assigns output label equal to 1 if the activation is greater or equal 0, and -1 otherwise.

The cost function minimisation step is crucial for the learning process, but might be computationally heavy, if the weight update is calculated based on the entire training dataset—as in the case of *batch gradient descent* used by the Adaline. It is possible to accelerate the process by utilising the *stochastic gradient descent* (SGD) algorithm instead, which updates the weights on a sample by sample basis, or the *mini-batch gradient descent* which takes in smaller subsets of data and performs the batch gradient descent on them. The last option is particularly of interest training on large datasets and with multilayer NNs [36, Chapter 2].

2.3.3.3 Multilayer artificial neural network

While a single-layer ANN has only one connection between input and output layer, multilayer architectures can be built of an arbitrary number of *hidden* layers in addition to the input and output layers. The layers can consist of multiple *units*

(also referred to as *neurons* or *nodes*). Multilayer NN architectures can be further divided into three main types:

- dense (fully connected) neural networks
- convolutional neural networks (CNNs)
- recurrent neural networks (RNNs)

This section describes the first type, based on the example of the *multilayer perceptron* (MLP) network—as it can be considered the most direct extension of the single-layer architecture. Figure 2.9 presents an example of MLP network with one hidden layer h . In this type of NN all units of one layer are connected to all units of the next layer through weights. If the network has more than one hidden layer, it is considered a *deep neural network* (DNN). In general, the learning capacity of the network increases with the number of layers and nodes within each layer, but along with it increases the risk of *overfitting* (see Section 2.3.3.9), due to an increasing number of model parameters. The MLP’s learning process can be

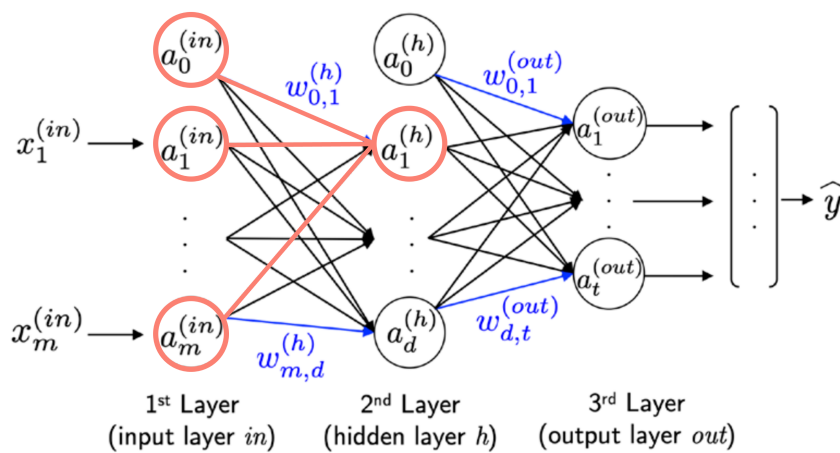


Figure 2.9: Multilayer Perceptron architecture, consisting of one hidden layer, in addition to the input and output layers. Marked in red, a part of the network that corresponds to the single-layer example illustrated by Figure 2.7—starting from the input values to the output from the activation function. Adapted from [36, Chapter 12] under MIT license.

summarised in three steps, repeated for multiple epochs [36, Chapter 12]:

1. Forward propagation

At this step, subsequent activations are calculated from the left (input layer) to the right (output layer) of the network sketched in Figure 2.9. Assuming this is performed on a batch of samples, activations of the hidden and output layers, respectively, can be written in the matrix notation as follows:

$$\mathbf{A}^{(h)} = \phi(\mathbf{Z}^{(h)}) = \phi(\mathbf{A}^{(in)}\mathbf{W}^{(h)}), \quad (2.15)$$

$$\mathbf{A}^{(out)} = \phi(\mathbf{Z}^{(out)}) = \phi(\mathbf{A}^{(h)}\mathbf{W}^{(out)}), \quad (2.16)$$

where superscripts (*in*), (*h*) and (*out*) correspond to the input, hidden and output layers, respectively. Note that the linear activation function used in Adaline is not capable of solving complex, non-linear problems. A range of more sophisticated, differentiable activation functions is described in Section 2.3.3.7.

2. Error calculation

The cost function is calculated based on the values of activations from the output layer and the true labels. There are various cost functions that can be used in more complex ANNs (instead of a simple SSE function, as in Adaline), some of which are suited for classification, some for regression tasks¹¹. Similarly, there is a wide range of optimisers to choose from¹²—the best one, for a given problem, can be found in the hyperparameter tuning process.

3. Backpropagation

Backpropagation is an extremely important algorithm that sends the feedback about the error through multiple layers, from the end of the network (output) to its beginning, in an efficient manner. It employs the mathematical *chain rule* to compute partial derivatives of complex, nested cost functions. The derivatives are used by the optimising algorithm (e.g. gradient descent) to adjust the weights across the network, aiming at lowering the loss from epoch to epoch.

Another important aspect of increasing the number of layers in the network is a phenomenon known as *vanishing or exploding gradients*. It manifests itself in the weight updates being extremely small or large, respectively, which in turn makes it impossible for the optimisation algorithm to converge. It is related to repeated

¹¹An interested reader may check the full list of cost functions available in Keras library at <https://keras.io/api/losses/>.

¹²A list of optimisers implemented in Keras: <https://keras.io/api/optimizers/>.

differentiation operations throughout the backpropagation process and can be mitigated via avoiding certain activation functions (see Section 2.3.3.7, implementing *residual connections* (see [37, p. 7]) or RNN-specific techniques described in [36, Chapter 16].

2.3.3.4 Convolutional neural network

This section opens with a quick introduction of the mathematical operation of convolution. More specifically, a one-dimensional *discrete convolution* between two vectors, \mathbf{x} (input signal) and \mathbf{w} (kernel), which have n and m elements, respectively, with $m \leq n$. For practical reasons we assume that vector \mathbf{x} is padded with an arbitrary number (denoted as p) of zeroes on each side (resulting in \mathbf{x}^p , and thus has size of $n + 2p$). A formula for computing such convolution operator $*$ is as follows:

$$\mathbf{y} = \mathbf{x} * \mathbf{w} \rightarrow y[i] = \sum_{k=0}^{m-1} \mathbf{x}^p[i + m - k]w[k], \quad (2.17)$$

where index i runs through each element of the output vector \mathbf{y} . For practical computation advice and information about a two-dimensional convolution formula, an interested reader can check [36, Chapter 15].

Convolutional neural networks (also referred to as *convnets*) are a kind of networks that have convolution layers as their main building blocks, but these can be supplemented by layers of any other type. Two-dimensional CNNs are widely used in computer vision tasks, such as image classification. One-dimensional convnets, on the other hand, are often combined with RNN blocks in the modelling of sequential data. Let us first describe 2D convnets, as the visual process of recognising patterns in images appears more intuitive than recognising 1D patterns in sequential data.

In the 2D CNN architecture, the early layers are responsible for recognising simple shapes, e.g. detecting edges, straight lines, blobs. Deeper layers built on the knowledge from the previous layers, combining the detected low-level shapes into more complex patterns and objects, such as buildings, cars or animal species. This process of identifying increasingly complicated and meaningful shapes can be thought of as the built-in feature engineering ability of the neural networks mentioned in Section 2.3.3.1. A single step of the forward propagation part of the CNN layer's training process is illustrated by Figure 2.10 and can be described as follows: a *kernel* (alternatively called a *filter*) is a matrix of trainable parameters (weights); it slides around an image channel performing discrete 2D convolutions

between the filter elements and image pixels. There is one kernel for each of the k channels of the image. Convolution results from a particular location in the input channel are then summed over the k channels and projected onto the corresponding location in the output feature map [37, Chapter 5].

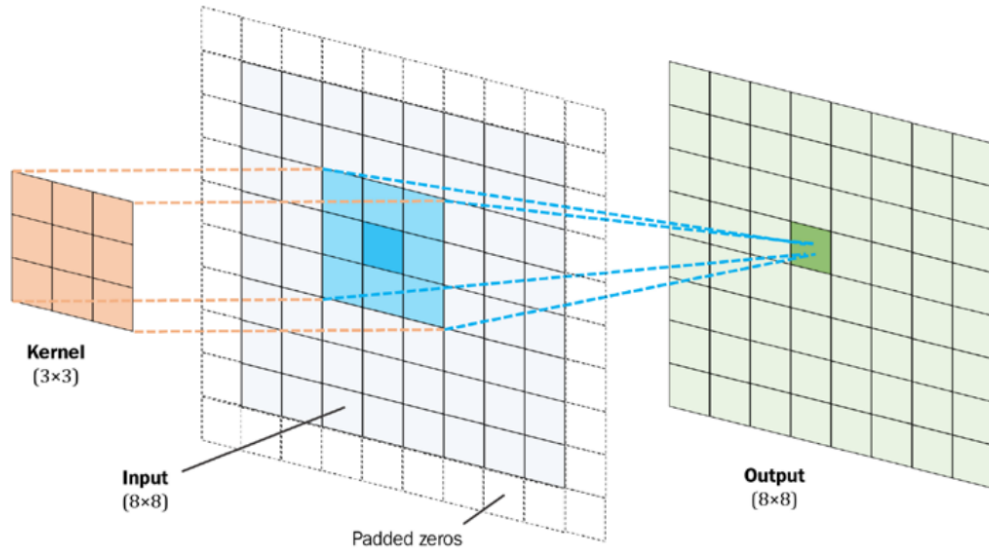


Figure 2.10: 2D convolution of a one-channel 8x8-element input matrix and 3x3-element kernel matrix. The size of the resulting feature map is 8x8, provided that the kernel slides by one element at a time and the input matrix is padded with one row of zeroes on each side. Reproduced from [36, Chapter 15] under MIT license.

Correspondingly, 1D convnets are used to detect one-dimensional patterns in the sequential data, e.g. temporal patterns in time series data. As shown in Equation (2.17), if a 1D kernel slides over a sequence with a stride¹³ higher than 1, it can also significantly shorten the length of the sequence, which can then be processed faster by the subsequent (often RNN) layer [37, Chapter 6].

Convolutional layers are often combined with subsampling (*pooling*) layers and followed by fully connected layers which use the CNN-derived features to predict the target values. Pooling layers do not have any trainable parameters and their primary function is to reduce the size of the feature maps [36, Chapter 15].

Another interesting concept widely used within the CNN architectures (but not exclusively there) is a *dropout*. It is a *regularisation*¹⁴ technique, which is based on

¹³Stride parameter determines the number of elements that is being “jumped over” when moving a convolution filter to the next position over the input vector or matrix.

¹⁴Regularisation is one of the methods used to fight overfitting.

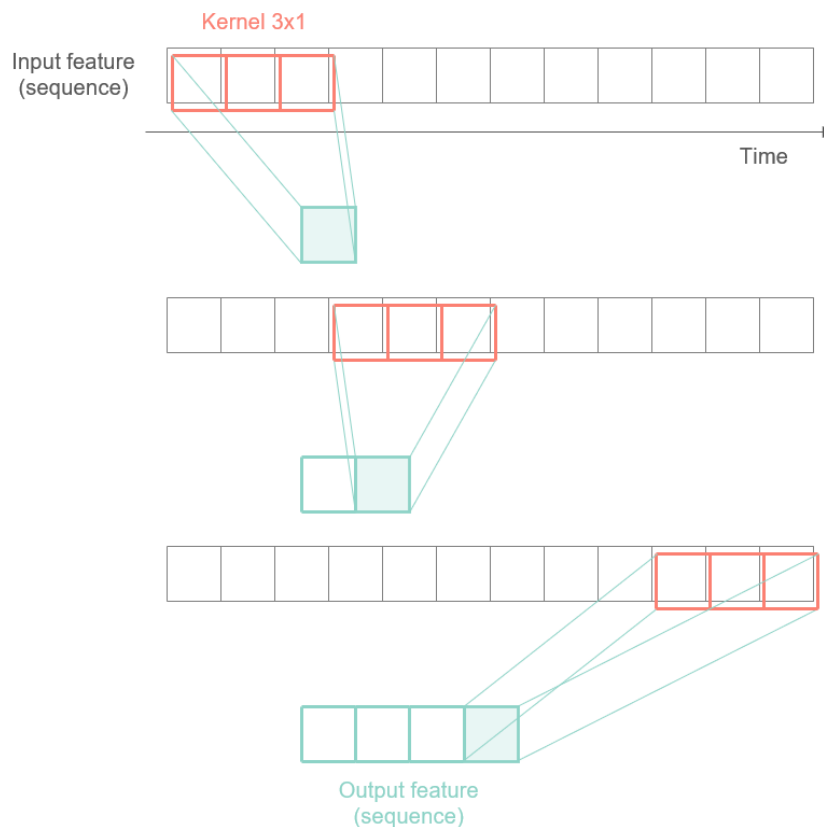


Figure 2.11: 1D convolution of a 12-element time series input feature and 3-element kernel, with stride equal to 3. The output sequence has 4 elements—it is significantly shorter than the input sequence.

deactivating a number of randomly selected nodes in a layer (setting their outputs to zero in the forward propagation stage of the training process). Dropout is set *on* a layer and can be parametrised by a *dropout rate*, which determines the fraction of the layer’s nodes that should be switched off [37, Chapter 4].

2.3.3.5 Recurrent neural network

Neural network architectures described in the sections above share an important property: they assume that the input data samples (observations) are independent of each other. Hence, the order in which they are fed into the network can be random—the network updates the weights in the forward-backpropagation pass over a sample or a batch of samples and does not keep any information in memory

about the processed samples. This type of networks can be classified as *feedforward networks* [37, Chapter 6].

Sequential data, such as time series, violate the independence assumption. Therefore, it is crucial to process them in the sorted order (e.g. sorted along time axis, in case of time series data) and leverage the information learnt from the preceding samples. An architecture capable of processing the sequence and retaining the information from the past training examples is known as a *recurrent neural network*. The difference between a feedforward network and a recurrent network comes down to the fact that in an RNN, the hidden layer at time step t receives inputs not only from the preceding layer, but also from its own unit activated at previous time step $t - 1$. This is visualised for a simple, one-hidden-layer case in Figure 2.12. The input, hidden and output layers are denoted by \mathbf{x} , \mathbf{h} and \mathbf{o} , respectively. Matrices of weights \mathbf{W} mark connections between the layers indicated by subscripts. The weight matrix \mathbf{W}_{hh} in the RNN section is associated with a so-called *recurrent edge* (recurrent connection between the subsequent time steps). The RNN structure is first presented in the compact form (with the round arrow) and then in the unfolded way. Typically, an RNN layer can either output a sequence of values (for multiple time steps) or just one output from the last time step. The activations of the hidden layer for a unit at time step t can be computed in the following way [36, Chapter 16]:

$$\mathbf{h}^{(t)} = \phi_h \left(\mathbf{z}_h^{(t)} \right) = \phi_h \left(\mathbf{W}_{xh} \mathbf{x}^{(t)} + \mathbf{W}_{hh} \mathbf{h}^{(t-1)} + \mathbf{b}_h \right), \quad (2.18)$$

where ϕ_h is the activation function and \mathbf{b}_h is the bias in the hidden layer h .

The backpropagation in RNNs is slightly different than in the feedforward networks, as the error is propagated both across the layers and the time steps. For more information on the *backpropagation through time* process (BPTT) the reader is referred to [36, Chapter 16]. BPTT is not free from challenges, such as the vanishing and exploding gradients problem mentioned in Section 2.3.3.3. One way of tackling these problems is to implement a more sophisticated RNN algorithm called the *long short-term memory* (LSTM).

The key concept that makes LSTM capable of modelling long-range dependencies in the data is a *memory cell*, which substitutes a basic hidden layer in the RNN network architecture. The memory cell's structure is schematically presented in Figure 2.13. Compared to the regular flow of information between subsequent time steps in an RNN, there is an additional flow that carries information across many time steps without being repeatedly multiplied with weights. It is called a *cell state*, $\mathbf{C}^{(t)}$ and it is marked in the figure with the red rectangle. It preserves the information from the early steps until the end of a long sequence is processed.

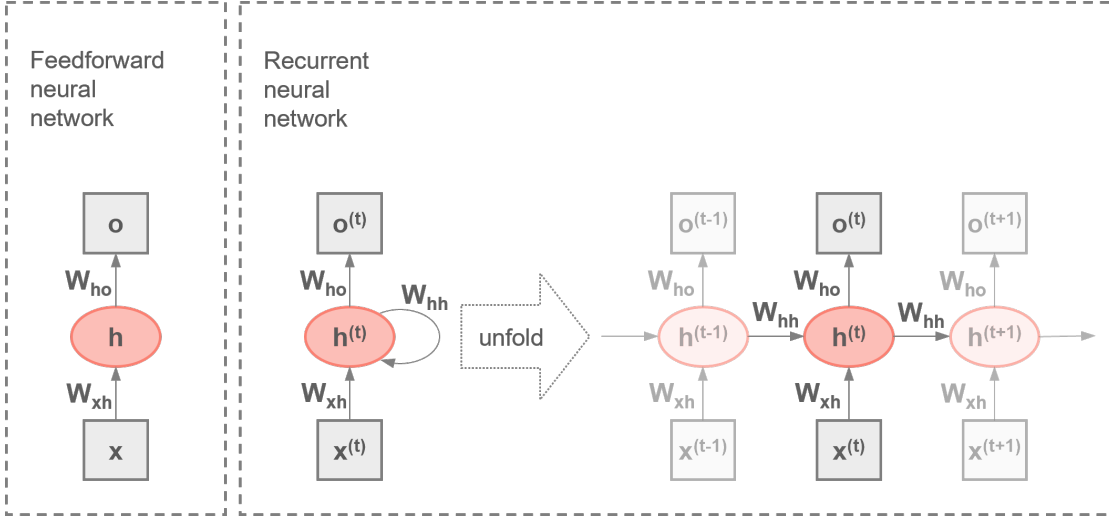


Figure 2.12: Comparison of a standard feedforward network and recurrent neural network. Input, hidden and output layers are denoted by \mathbf{x} , \mathbf{h} and \mathbf{o} , respectively. Matrices of weights \mathbf{W} mark connections between the layers. Weight matrix \mathbf{W}_{hh} is associated with the recurrent edge. Redrawn based on figures from [36, Chapter 16] under MIT license.

In addition to the cell state component, the memory cell is built of several *gates*, coloured in yellow, which have different tasks¹⁵ assigned with regard to processing the information that is fed into the cell from the input layer (\mathbf{x}^t) and from the previous time step of the cell ($\mathbf{h}^{(t-1)}$):

- *forget gate* \mathbf{f}_t —decides which information in the cell state flow is irrelevant and should be forgotten; computed as follows, with σ denoting a sigmoid activation function:

$$\mathbf{f}_t = \sigma (\mathbf{W}_{xf}\mathbf{x}^{(t)} + \mathbf{W}_{hf}\mathbf{h}^{(t-1)} + \mathbf{b}_f) \quad (2.19)$$

- *input gate* \mathbf{i}_t and *candidate value* $\tilde{\mathbf{C}}_t$ —contribute to the cell state update:

$$\mathbf{i}_t = \sigma (\mathbf{W}_{xi}\mathbf{x}^{(t)} + \mathbf{W}_{hi}\mathbf{h}^{(t-1)} + \mathbf{b}_i) \quad (2.20)$$

$$\tilde{\mathbf{C}}_t = \tanh (\mathbf{W}_{xc}\mathbf{x}^{(t)} + \mathbf{W}_{hc}\mathbf{h}^{(t-1)} + \mathbf{b}_c) \quad (2.21)$$

¹⁵The *tasks* are merely an attempt to interpret how different sets of weights affect the learning process and should rather be considered as its constraints, rather than literal interpretations [37, Chapter 6].

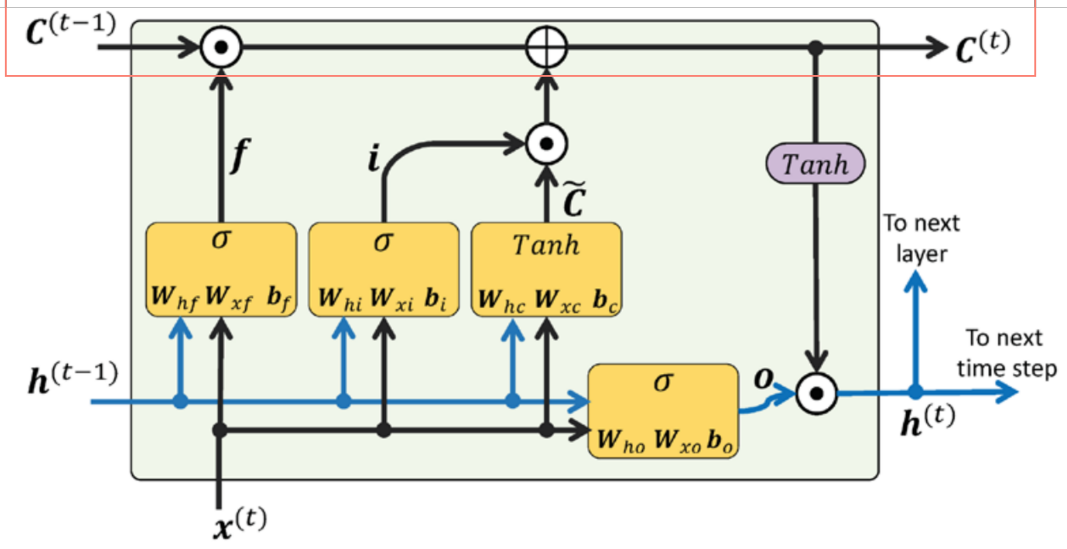


Figure 2.13: LSTM memory cell structure. Cell state information flow is marked by a red rectangle. Forget, input, candidate value and output gates with corresponding activation functions and weights are represented by yellow boxes. Adapted from [36, Chapter 16] under MIT license.

- *output gate* \mathbf{o}_t —affects the update of the hidden units' activations

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}^{(t)} + \mathbf{W}_{ho}\mathbf{h}^{(t-1)} + \mathbf{b}_o) \quad (2.22)$$

The above equations can be used to compute the cell state at time step t according to:

$$\mathbf{C}^{(t)} = (\mathbf{C}^{(t-1)} \odot \mathbf{f}_t) \oplus (\mathbf{i}_t \odot \tilde{\mathbf{C}}_t), \quad (2.23)$$

where \odot and \oplus denote element-wise multiplication and element-wise addition operations, respectively. Eventually, the final formula for a hidden unit's activation at time step t can be written as:

$$\mathbf{h}^{(t)} = \mathbf{o}_t \odot \tanh(\mathbf{C}^{(t)}) \quad (2.24)$$

Note that at the first time step both the cell state and the activation corresponding to the previous step are initialised to zeroes or small random values [37, Chapter 6].

The last couple of sections summarised the main types of NN building blocks (dense, CNN, RNN layers) that will be used in the experimental part of this thesis. The following sections give an outline of a standard machine learning workflow and provide more detailed descriptions of selected machine learning aspects.

2.3.3.6 Machine learning workflow

A universal workflow for conducting a machine learning project includes the following steps [45]:

1. Defining the problem, collecting the data and choosing the measure of success;

At this stage, an expert domain knowledge should be used to determine:

- what type of problem is there to solve: classification, regression, clustering?
- what kind of data are available? are the data labeled? are the labels reliable or noisy? is the dataset balanced?
- is there a baseline model to beat?
- what kind of performance metrics can be used to compare the obtained models?
- how to evaluate the training process—what type of validation protocol to use? [36, Chapter 6]

2. Exploring, visualising and pre-processing the data;

As part of the exploratory data analysis (EDA) one should plot the data and/or their distributions, as well as identify and handle missing data, outliers and dataset imbalance. Data pre-processing can include multiple steps, e.g.:

- dimensionality reduction through feature selection or feature extraction
- data standardisation—data are transformed so that each feature in the dataset is centered (the mean is shifted to 0) and scaled (its standard deviation is equal to 1); it helps the optimising algorithm converge faster at the global or local minimum of the cost function
- feature encoding

An obligatory step is to split the available data into training and test sets. The model should be trained and validated on the train set. Afterwards, the model is tested on the part of the data that has been left out—typically, between 10-30% of the dataset. Thus, it is possible to check if the model generalises well to the unseen data [36, Chapter 4 & Chapter 5].

3. Model development;

This stage can be performed in parallel for multiple algorithms or architectures, to eventually choose the best performing variant. Within a single ML model there are also multiple hyperparameters which can be tuned to boost the model’s predictive power. Among commonly tuned hyperparameters there are: activation function, optimising algorithm, learning rate and loss function. The model architecture elements, such as the number of nodes in the layers or presence of different types of layers, can be themselves tuned as hyperparameters.

Each model (each architecture and hyperparameter combination) is regularly validated throughout the training process. The learning that the model undergoes within this stage is described in more details in Section 2.3.3.2 on the example of the single-layer ANN.

The training strategy is usually to develop a too complex model that overfits¹⁶ and subsequently regularise it. The final assessment of the model’s performance is done by performing predictions on the test set and comparing them to ground truth values of the target variable using performance metrics chosen in the first phase of the project.

Note that this workflow stops at the stage of finalising the model and does not include the steps which need to be taken to deploy the model in the production mode.

2.3.3.7 Activation functions

To capture non-linear dependencies in data, one needs to use non-linear activation functions—otherwise the network will only perform linear operations, such as dot product and addition (used for computing net inputs) [37, Chapter 3]. A set of widely used activation functions is presented in Figure 2.14. Different functions are preferred for different problems. For instance, the linear variant is used in the output layer in regression problems, *sigmoid* in the output layer in binary classification problems. *Hyperbolic tangent (tanh)*, belonging together with the sigmoid function to the group of *s-shaped* functions, is preferred over the sigmoid function in the multilayer NNs, as it has a broader range of output values ((-1, 1) compared to the sigmoid’s (0, 1)). Consequently, its derivative can take larger values, allows for larger weight updates and the optimising algorithm converges faster[46]. Eventually, the *Rectified linear unit (ReLU)* and its multiple strains are

¹⁶A model *overfits* when it performs well on the training data, but predicts poorly on the new, unseen data.

extremely popular in the deep NNs. Due to ReLU's derivative being always 1 for positive net input values, it tackles the vanishing gradients problem [36, Chapter 13].

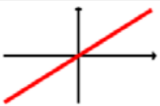
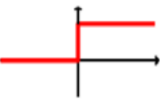


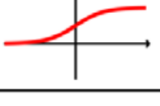

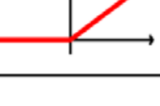
Activation function	Equation	Example	1D graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit step (Heaviside function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, multilayer NN	
Hyperbolic tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Figure 2.14: Selection of widely used activation functions. Reproduced from [36, Chapter 15] under MIT license.

2.3.3.8 Loss functions and performance metrics in regression tasks

A loss function is an objective function that is being optimised during the learning process [47]. Performance metric, on the other hand, is used to evaluate the performance of the model after a certain training period and is not used in the error backpropagation process [48]. Any loss function *can* be used as a metric, in fact, it is often the case in the regression tasks. Both loss and metric functions

indicate in various ways how far the model predictions are from the actual values of the target variable. The functions used in this project are described below, based on [49], [36, Chapter 10] and [50]:

- mean squared error (MSE)—equal to the averaged value of the SSE cost introduced in Section 2.3.3.2, defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2, \quad (2.25)$$

where n denotes the number of observations for which the predicted output value \hat{y} and the true output value are compared. Due to the squaring operation preceding the averaging, large errors are given higher weights in the average.

- root mean squared error (RMSE)—can be thought of as MSE transformed back to the original unit of predictions:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2}, \quad (2.26)$$

- mean absolute error (MAE)—averages absolute values of the prediction errors, thus all errors are weighted equally:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|, \quad (2.27)$$

- mean absolute percentage error (MAPE)—the difference between the predicted and true values is normalised by the true value for each observation:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}} \right|, \quad (2.28)$$

- coefficient of determination R^2 —reflects the accuracy of predictions:

$$R^2 = \frac{\sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^n (y^{(i)} - \bar{y})^2} \quad (2.29)$$

$R^2 = 1$ for predictions perfectly matching the true values ($MSE = 0$) and can be negative if the overall difference between the predictions and the true values is higher than between the true values and their mean value \bar{y} .

Out of the four error functions, only the RMSE function is not implemented in Keras as both a loss and a metric function. It is only offered as a performance metrics, because as a loss function it would deliver exactly the same optimised network weights as MSE.

2.3.3.9 Model complexity and cross-validation

An important aspect of training any machine learning model is to make sure that it captures general trends in the data, rather than memorises very specific patterns and noise present in the training dataset. Should the latter happen, the model will not be capable of making correct predictions on the unseen data—such a situation can be diagnosed as overfitting and such model is referred to as one having a *high variance*. The opposite situation would involve the model not being able to capture the general patterns in the training dataset, which also results in the poor performance the unseen data. In such case, however, the model exhibits a *high bias* and the situation is classified as *underfitting* [36, Chapter 3].

Cross-validation is an important technique that allows for estimating model’s performance on the new data during the training process. In the search for the best performing hyperparameter combination, multiple models are being fitted on the train set—every such model needs to be evaluated on the unseen data (validation set) during the training process, in order to select the best performing one. In the final step, the most credible and unbiased evaluation of the model’s generalisation ability is performed on the test set.

Standard cross-validation methods include:

- holdout validation—requires splitting the input dataset into train, test and validation sets; as long as test set remains the “unseen” data until the end of the training process, the validation set is used to regularly evaluate model’s performance *during* the training process;
- k-fold cross-validation—the train set is randomly split into k subsets, and in each epoch a different subset is used for performance evaluation;
- leave-one-out-cross-validation—a single sample is left out for validating the model’s performance during each training epoch.

The loss and the performance score of a given model can be recorded for both training and validation sets at each step of the training (for each epoch). These

values can then be appended and plotted as *learning* (or *training*) and *validation curves*, with respect to the specific parameter that is being tuned—e.g. epoch number, when trying to find the optimum number of iterations that maximise the score, before the model starts to overfit. If one can imagine a single parameter that can describe model’s complexity, the training and validation curve would approximately look like the ones presented in Figure 2.15. Note that the validation score will always be lower than the training score (which tends to always increase with the increasing model complexity) and that the best model is assumed to be the one with the optimal validation score (lowest, if the used metric is an error metric, such as MSE, or highest, if the R^2 is used). The moment the validation score crosses the inflection point, the model starts overfitting and its performance on the unseen data must be expected to drop.

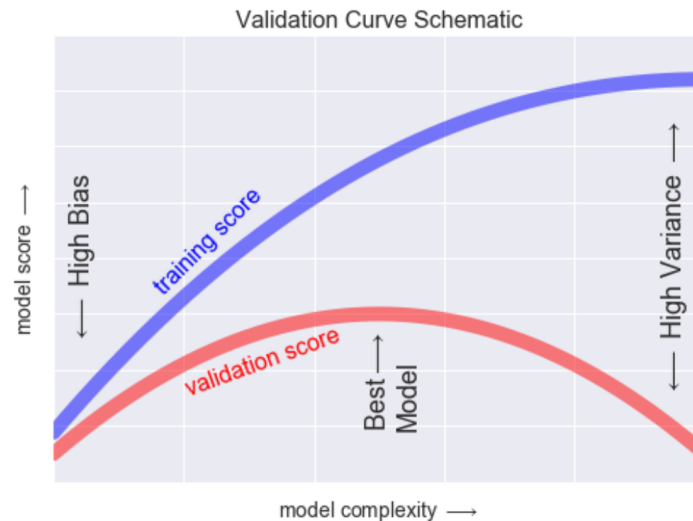


Figure 2.15: Training and validation curves schematic. The optimal model complexity corresponds to the inflection point of the validation curve. To the left from it, the model underfits (has a high bias), to the right it overfits (has a high variance). Reproduced from [36, Chapter 15] under CC-BY-NC-ND license.

Tuning a model’s complexity is the key to achieving a good trade-off between bias and variance. To fight overfitting, one can for instance:

- reduce the number of layers and/or nodes in the network,
- apply L1 and/or L2 regularisation—penalising large values of weight coefficients [36, Chapter 3],

- regularise the network by adding dropout [36, Chapter 15].

2.3.3.10 Inception module

In a standard configuration, a network consists of layers that are stacked on top of each other, so that the output from the first layer is an input to the second layer, the output from the second layer is an input to the third layer, and so forth. However, there are architectures that exhibit more flexible approach—in which the layers form a graph, rather than a linear stack. *Inception module* is one example of such architecture. It is a concept developed in 2015 by Szegedy et al. at Google, which significantly improved the utilisation of the computing resources inside the CNN architecture for image classification and detection [51]. Its underlying idea is that several convolutional layers operate on the same level—instead of being stacked sequentially. They act as branches—they are processed in parallel and their outputs are concatenated into a single tensor on the subsequent level, as demonstrated in Figure 2.16.

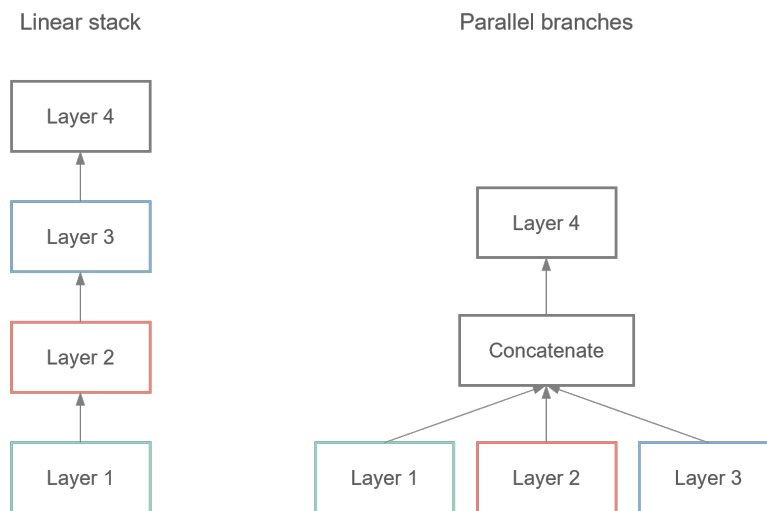


Figure 2.16: Comparison of network configurations: linear stack of layers vs Inception module consisting of three parallel branches concatenated on top.

Inception modules can be implemented in Keras using a *functional API*—which, as opposed to Keras *sequential API*, allows for designing more complex networks with e.g. multiple inputs/outputs to/from a layer [52].

2.3.3.11 Related work

One of the model architectures tested out in this thesis has been inspired by the MRC-LSTM solution proposed in [53]. Guo et al. combined a multi-scale residual convolutional neural network (MRC) and the LSTM concepts for the purpose of Bitcoin closing price prediction. The MRC block has been designed to extract highly expressive features that would represent input data on different time scales. For this purpose, they implemented an inception-like block which included three parallel 1D CNN layers with different kernel sizes (responsible for extracting patterns on different time scales) and ReLU activation functions. In the next step, the three branches were concatenated together with an identity mapping of the original input, and further fed into a 2D CNN layer. Finally, an LSTM layer and a fully connected layer were used to learn the patterns and predict the prices.

What will be adopted in this thesis is the idea of combining multiple (parallel) 1D CNN layers to extract the features on different time scales with subsequent LSTM and dense layers. The number of layers on each level, as well as their respective numbers of units (and kernel sizes, in the CNN case) will be subject to hyperparameter tuning.

Chapter 3

Methods

3.1 General experiment outline

The experimental part of this project can be divided into three modules, all of which were expected to generate a system price prediction model. The outcomes of the three modules can be summarised as:

1. a baseline model produced with the statistical method ARIMAX (see Section 3.3)
2. a simple machine learning model based on a standard RNN network (see Section 3.4)
3. a more sophisticated machine learning model based on a 1DCNN-LSTM network (see Section 3.5)

The third module was designed as the core part of the study, hence it involves the most extensive parameter search and tuning.

3.2 Exploratory data analysis and pre-processing

This section summarises the exploratory data analysis performed on the input dataset prior to the model development stage. The goal of this part of the project

was to get familiar with the data and potentially discover which prediction frame (how many days or weeks ahead) might be promising.

The raw input dataset consisted of time series representing the daily filling levels from 63 Norwegian hydropower reservoirs (their official ID numbers assigned by the NVE served as columns names) and the SYS price history, covering the period between 01/01/2015 and 30/09/2021—2465 data points. The data were checked for missing values. Based on the visual assessment (Figure 3.1) only four reservoirs seemed to have an outstandingly poor coverage, namely IDs 690, 703, 2196 and 2197. The corresponding columns were dropped by setting the threshold to filter out reservoirs with more than 10% of data points missing. The missing values in the remaining columns were filled with the value of the nearest preceding or succeeding cell which had a valid numerical data point. The resulting dataframe consisted of 59 columns (58 reservoir filling level time series and the SYS price time series) and 2465 rows (time points), with all the data points formatted as floating point numbers.

In the next step, the data were plotted—see Figure 3.2. All filling level time series are plotted together, to inspect general characteristics of this type of data. A yearly seasonal behaviour can be observed in the majority of reservoirs. Typically, filling level values are contained in the 0–1 interval, with a few exceptions—these are most likely related to the failed volume to filling level conversion, less likely to the abnormal water levels exceeding the highest regulating levels. The bottom plot represents the SYS price history. No seasonality can be observed and there is no monotonous trend that would continue over the entire presented time period.

As a part of EDA, a cross-correlation between all features from the input dataset was calculated and plotted in the form of heatmaps. Analogous to the autocorrelation mechanism described in Section 2.3.2, a cross-correlation between two time series is a measure of their similarity. Calculating and plotting the cross-correlations was expected to yield insights into the importance of individual reservoirs in the price modelling process. The relationships between the filling levels of the reservoirs and the SYS price, calculated for various time lags, were expected to shed light on promising prediction time frames. However, it is important to keep in mind that cross-correlation can only depict linear relationships.

Further, a pre-processing step of despiking the SYS price time series was conducted, to reduce the influence of ultra short-term (one data point) price fluctuations on the model training process. Such fluctuations are assumed to be related to some extreme events on the market and should rather be treated as outliers. The rule had been set that every data point at time t , whose differences against the

values at time steps $t - 1$ and $t + 1$ were larger than 5, was to be replaced with the mean of the values from time steps $t - 1$ and $t + 1$. The threshold value of 5 was determined based on the visual assessment of multiple tested options. The result is presented in Figure 3.3.

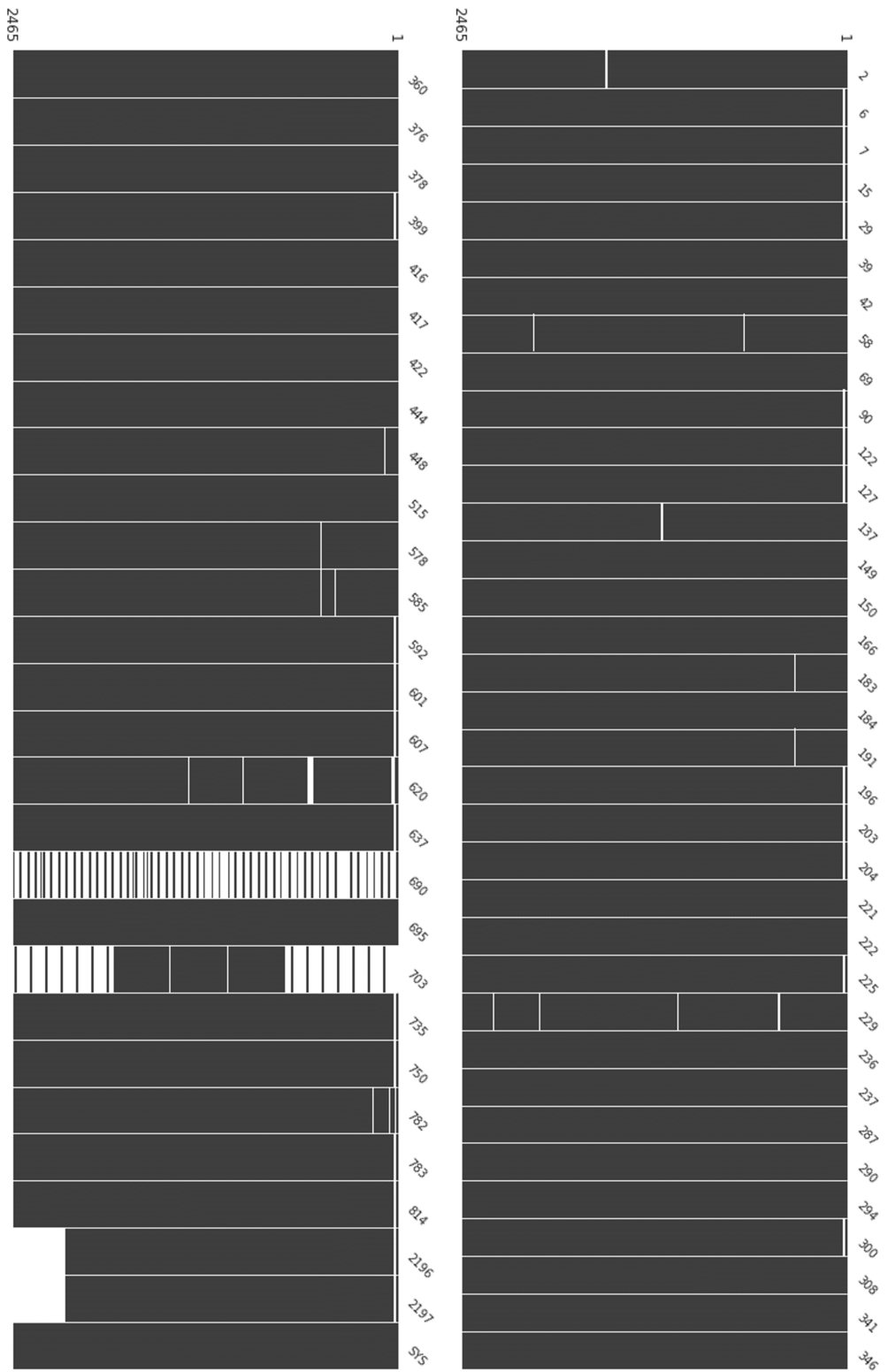


Figure 3.1: Raw dataset with missing data. White gaps in the columns correspond to missing data points for respective reservoirs or SYS price time series (last column) along the vertical time axis.

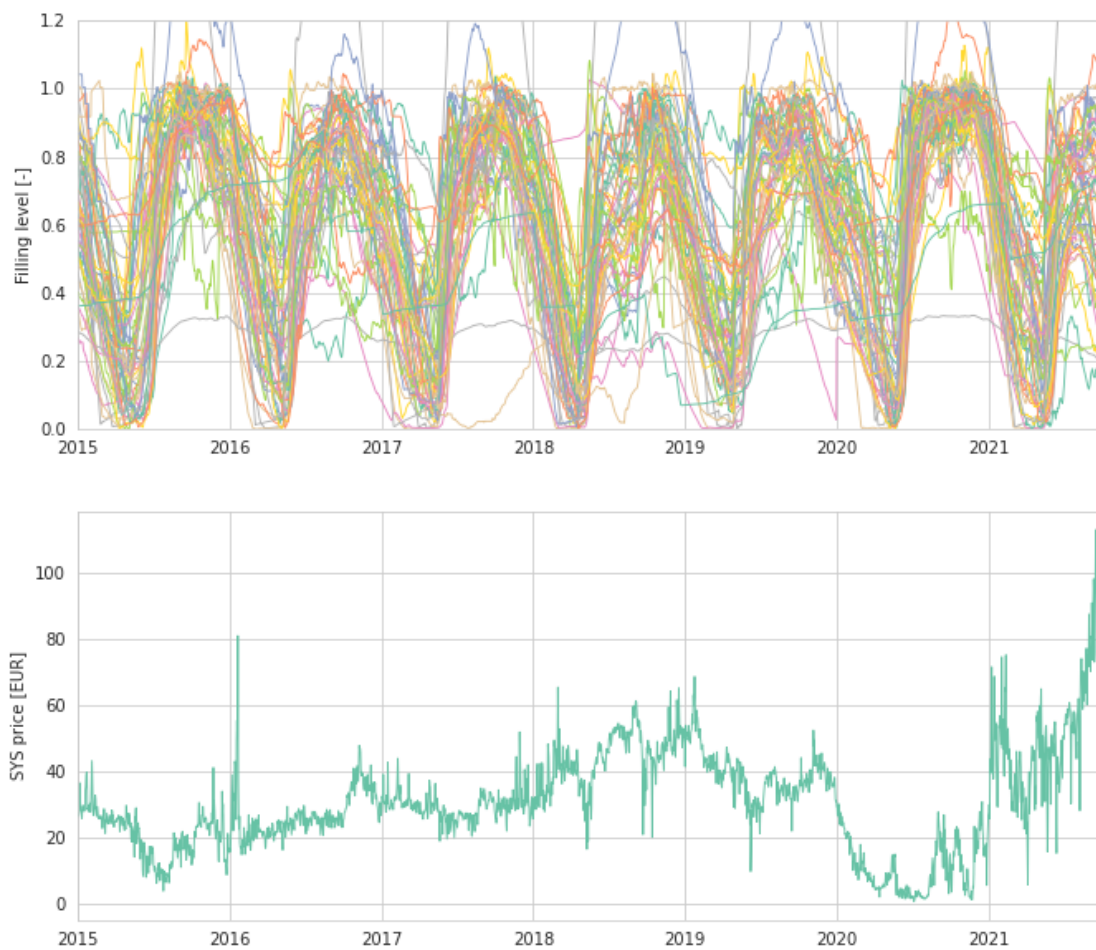


Figure 3.2: Visualisation of the raw input dataset. In the upper plot, all filling level time series are plotted. System price is plotted separately in the lower plot.

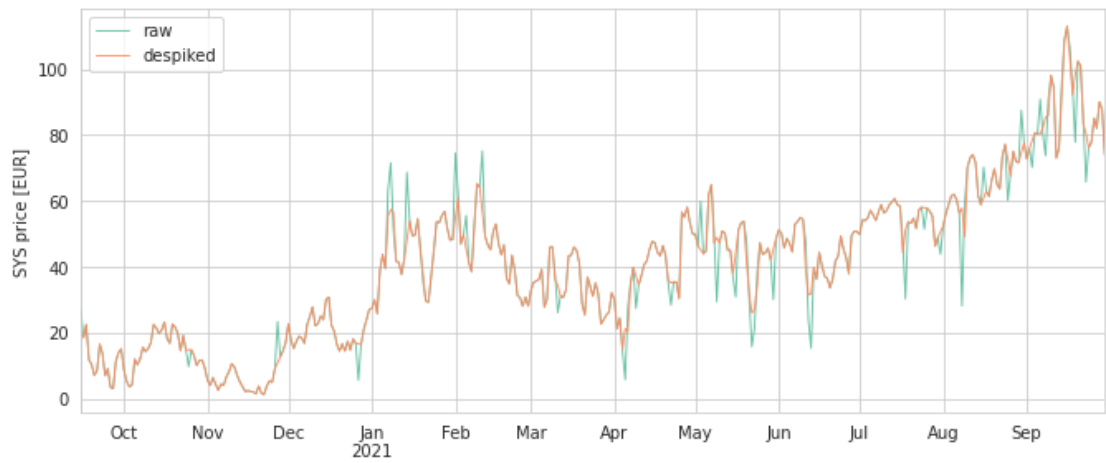


Figure 3.3: Despiking of the raw system price time series.

3.3 Statistical modelling using ARIMAX

ARIMAX—a statistical method for modelling time series—was used to create a baseline model. Although ARIMAX does not require training and testing the model on separate datasets, the train-test split was performed anyway and the model’s performance was evaluated on the test set. This was to ensure that the results are comparable with the results of machine learning models presented later in the thesis.

The ARIMAX model has been developed using a *rolling forecast approach* described by [54] and [55]. The main idea underlying this approach is to re-fit the model every time a new observation is received, to avoid accumulating the error when forecasting for multiple time steps ahead. The looping workflow was as follows: at each time step t in the test set, the model was fitted on p AR and q MA steps, the forecast was calculated a number of steps into the future (an arbitrary prediction time frame—a parameter called *delay*), the last value was saved as prediction for $t + \text{delay}$ time step and afterwards the entire process was repeated to obtain a prediction for $t + 1 + \text{delay}$ time step, $t + 2 + \text{delay}$ time step, and so forth, throughout the entire test set.

Inside every loop, the model was fitted using SARIMAX class from statsmodels library (see the code in Example 3.3.1). SARIMAX parameters include the *endogenous* variable `endog` (the target times series—SYS price), the *exogenous* variables `exog` (filling level time series from the selected reservoirs) and the `order` that specifies the (p, d, q) order of the model. The p , q and d values were tuned manually as hyperparameters. Thus parametrised SARIMAX, performs in fact the ARIMAX modelling. The letter “S” in the name of the class extends its applicability to SARIMAX modelling (another version of ARIMA-like models, that accounts for seasonality in the data), however, it would require specifying additional parameters for the model to perform as SARIMAX.

Example 3.3.1.

```

1 model = SARIMAX(endog=df_training['SYS'],
2                 order=(p, d, q),
3                 exog=df_training[['290', '300', '360', '378']])
4 model_SYS = model.fit()
```

A steep increase of training time was observed for increasing p and q , hence the highest value tested in both cases was equal to 5. The manual parameter tuning was based on the performance metrics (RMSE, MAE, MAPE and R^2) calculated

for each generated model, as well as visualisation of the original and predicted time series for the time interval corresponding to the ML test set range.

3.4 Machine learning modelling using a basic RNN architecture

This section covers the methodology used to develop a basic machine learning model—consisting of one RNN layer and one fully connected (also referred to as *dense*) output layer.

After the initial pre-processing described in Section 3.2, the input dataset was split into training and test sets, the former encompassing the first 90% of the data, the latter consisting of the remaining 10%. The split was performed chronologically, without shuffling the observations, as it is crucial to preserve the original order of consecutive data points when processing and analysing time series data. The split at this stage was performed solely to fit a standard scaler function on the training set and subsequently use it to standardise the entire dataset. Note that in this project the SYS price time series both belongs to the input features set (together with the filling level time series) and serves as a target variable.

While feedforward ANNs take two-dimensional matrices of data as input (*observations* \times *features*)¹, the RNN layer processes sequences, hence it requires 3D tensors (*observations* \times *steps* \times *features*) on input. To structure the data into 3D tensors, a custom generator function proposed by Chollet in [37, Chapter 6] was used². The generator function requires specifying the following parameters which determine the shape of the 3D data tensor:

- delay—prediction time frame—the number of time steps into the future that the target prediction should be made for,
- lookback—the number of previous time steps that an RNN layer takes into account when calculating activations at a given time step,

¹An observation is simply a row in the input dataset, consisting of the values of different features at a particular time point t .

²The code from the book is available under MIT license at https://github.com/fchollet/deep-learning-with-python-notebooks/tree/master/first_edition. In the second edition of the book, generator function has been replaced by Keras built-in function `timeseries_dataset_from_array`.

- batch size—the number of observations fed into the network at once,
- step—the number of time steps between consecutive samples in the lookback sequence.

The function yields tuples which consist of a batch of input features and the corresponding target array. Separate generators were instantiated to yield batches of data from the training, validation and testing sets—extracted in the chronological order from the standardised input dataset in proportions 70%, 20% and 10%, respectively.

After preparing the data generation tool, a framework for tuning machine learning models was set up using KerasTuner³ API. The main goal of tuning is to find a hyperparameter combination that yields the best model performance, following the metrics specified upon tuner’s instantiation. A `BayesianOptimization` tuner was chosen over other available tuners (e.g. `RandomSearch`). This variant is supposed to learn from previous iterations—investigates the hyperparameter space in the areas that had been found promising earlier in the training process, instead of randomly choosing hyperparameter combinations that are to be tried out next [56]. The code used to define the `BayesianOptimization` tuner is presented by Example 3.4.1. Among parameters that can be specified when creating a tuner instance, there are: the objective function (`objective`—set in this project to the loss calculated on the validation set), the maximum number of hyperparameter combinations to draw from the hyperparameter space (`max_trials`), the number of executions per hyperparameter combination (`executions_per_trial`).

Example 3.4.1.

```

1 from keras_tuner.tuners import BayesianOptimization
2 from keras_tuner.engine.hyperparameters import HyperParameters
3
4 tuner = BayesianOptimization(
5     hypermodel=build_model,
6     objective='val_loss',
7     max_trials=120,
8     executions_per_trial=2,
9     directory=TUNER_LOG_DIR)

```

The most important part, however, is to specify under `hypermodel` the model instance that should be tuned. In this project, the model was returned from a

³https://keras.io/api/keras_tuner/

separate function, `build_model`—Example 3.4.2, which took an instance of Keras HyperParameters [57] class (`hp`) as an argument.

Example 3.4.2.

```

1 def build_model(hp):
2     """
3     This function defines a parameter search space for tuning and
4     builds a model using an external function.
5     """
6     rnn_units = hp.Choice('rnn_units',
7                           [4, 8, 16, 32, 64])
8     activation_hidden = hp.Choice("activation_hidden",
9                                   ["relu", "tanh"])
10    dropout = hp.Choice('dr',
11                        [0.0, 0.1, 0.2, 0.5])
12    recurrent_dropout = hp.Choice('rec_dr',
13                                  [0.0, 0.1, 0.2, 0.5])
14    optimizer = hp.Choice('optimizer',
15                          ['adam', 'rmsprop', 'adadelat'])
16    lr = hp.Float('lr',
17                 min_value=1e-5, max_value=1e-2, sampling
18                 ="log")
19
20    # call an external model architecture specified within
21    design_model function
22    model = design_model(
23        rnn_units=rnn_units,
24        activation_hidden=activation_hidden,
25        dropout=dropout,
26        recurrent_dropout=recurrent_dropout,
27        optimizer=optimizer,
28        lr=lr
29    )
30    return model

```

The HyperParameters class allows for defining a *search space*—a grid of hyperparameter values that the tuner should draw their combinations from. As shown in the code snippet above, tuning included various parameters of the RNN layer: the number of units (`rnn_units`), the activation function (`activation_hidden`), the dropout and recurrent dropout rates (`dropout` and `recurrent_dropout`). Additionally, for the optimising algorithm (`optimizer`) there were three options available within the search space, and the learning rate (`lr`) values were sampled from the specified range. The model architecture itself was defined in a function `design_model`, presented by Example 3.4.3:

Example 3.4.3.

```

1  from tensorflow.keras.layers import *
2  from tensorflow.keras.models import Model
3  from tensorflow.keras.metrics import MeanAbsoluteError,
   RootMeanSquaredError, MeanAbsolutePercentageError
4  from tensorflow.keras.optimizers import Adam, RMSprop, Adadelta
5  from tensorflow.keras.losses import MeanSquaredError
6
7  def design_model(rnn_units, activation_hidden, dropout,
8                  recurrent_dropout, optimizer, lr):
9      """
10     This function builds a network architecture according to the
11     hyperparameter combination passed as an argument. Returns a
12     compiled model.
13     """
14     input_layer = Input(shape=(int(lookback / step), num_features)
15                          )
16
17     rnn = SimpleRNN(rnn_units,
18                    return_sequences=False,
19                    activation=activation_hidden,
20                    dropout=dropout,
21                    recurrent_dropout=recurrent_dropout)(input_layer
22                                                         )
23
24     output_layer = Dense(1)(rnn)
25
26     model = Model(inputs=input_layer,
27                  outputs=output_layer)
28
29     if optimizer == 'adam':
30         optimizer = Adam
31     elif optimizer == 'rmsprop':
32         optimizer = RMSprop
33     elif optimizer == 'adadelta':
34         optimizer = Adadelta
35
36     model.compile(loss=MeanSquaredError(),
37                  optimizer=optimizer(learning_rate=lr),
38                  metrics=[MeanAbsoluteError(),
39                           RootMeanSquaredError(),
40                           MeanAbsolutePercentageError()])
41
42     return model

```

The resulting model summary is shown in Figure 3.4.

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 60, 5)]	0
simple_rnn (SimpleRNN)	(None, 32)	1216
dense (Dense)	(None, 1)	33

```

=====
Total params: 1,249
Trainable params: 1,249
Non-trainable params: 0
=====

```

Figure 3.4: Simple RNN model summary.

Before running the search, two Keras *callback* objects⁴ were initialised: `EarlyStopping` and `TensorBoard`. The former is responsible for stopping the training process after a specified number of epochs without an improvement in the monitored metric [58] (validation loss, in this case), whereas the latter logs the metrics at each step of the training and can be further used to visualise the learning and validation curves [59]. The section of the code containing the callbacks' instantiation, together with the tuner's search call is presented in Example 3.4.4.

Example 3.4.4.

```

1 from tensorflow.keras.callbacks import EarlyStopping,
   TensorBoard
2
3 early_stopping = EarlyStopping(monitor='val_loss', patience=8,
   mode='min')
4
5 TB_LOG_DIR = f'{dir}/tboard_{datetime.datetime.now().strftime("%
   Y%m%d-%H%M%S")}'
6 tensorboard = TensorBoard(TB_LOG_DIR, histogram_freq=0,
   write_graph=True, write_images=True)
7
8 # Run the search
9 tuner.search(train_generator,
10             epochs=50,
11             validation_data=valid_generator,
12             shuffle=False,

```

⁴A callback is an object that can perform certain action at different stages of the training process, e.g. after every epoch.

```

13         callbacks=[early_stopping, tensorboard],
14         steps_per_epoch=train_steps,
15         validation_steps=val_steps)

```

The `search` method takes as arguments the arguments that should, in a standard workflow, be passed to the model fitting function, i.e. the training and validation data (here yielded by the generators), the number of epochs to run the training for, the list of callbacks, and the number of steps per epoch (how many times the generator should be called, in both the training and validation cases).

The tuner returns the best hyperparameters combination (the one that yielded the lowest validation loss score), as well as the best model itself—a set of model’s weights (parameters). The model is output at its best performing epoch [60], and can be saved in the `.hdf5` format—thus it is possible to load it and re-use for predictions in the future.

The best model was further used to make predictions on the full range of data, including the training, validation and test sets. For a given observation at time t , a single value of SYS price was predicted, that corresponded to the time step $t + \textit{delay}$ —hence, in order to be appended to the input dataset as a *prediction*, it had to be shifted forward by the number of steps equal to the `delay` parameter. Inverse scaling was applied, to revert the effect of standardisation and restore the predicted SYS price values to the original scale. A set of performance metrics, including RMSE, MAE, MAPE and R^2 , was calculated using the ground truth and predicted values for the observations belonging to the test set. Eventually, the results were plotted, to visually assess the match between the real and predicted SYS price time series.

3.5 Machine learning modelling using a hybrid 1DCNN-LSTM architecture

The network architecture presented in this section has been inspired by the approach implemented in [53], described in Section 2.3.3.11. The architecture combines the one-dimensional CNN layers and the LSTM variant of recurrent NNs.

The workflow for this approach follows the one of the simple RNN model in such aspects as: data standardisation, the generator function, the use of `BayesianOptimization` tuner, as well as post-processing stages, including prediction on the full range of

data, inverse scaling, calculating performance metrics on the test set and plotting the results. What differs is the hyperparameter tuning, which was extended beyond the parameters related to the model architecture and optimisation.

In the search for the best performing model, a two-level tuning was conducted. The outer layer of tuning was concentrated on the data pre-processing step. It included various combinations of input data transformations, such as:

- transformation of filling level time series into residual filling levels (presented in the middle plot in Figure 3.5); a result of the low-frequency smooth signal (dashed curves in the top plot), modelled as a rolling mean, being subtracted from the original time series.
- transformation of filling level time series into differenced filling levels (presented in the bottom plot in Figure 3.5); differencing was performed at each time step t by subtracting the value from time step $t - 1$.
- transformation of the (despiked) SYS price time series into smoothed SYS price—smoothing windows of 5 and 21 samples were tested, as shown in Figure 3.6.

Additionally, tuning at this level involved parameters determining the shape of the 3D data tensor taken as input by the LSTM layer, such as: delay, lookback, batch size and step. After initial manual parameter testing, lookback, batch size and step were fixed at 60, 32 and 1, respectively. Delay values that were further tested with different combinations of other parameters included 14, 28 and 42 days. The outer layer tuning was conducted manually, in separate Jupyter notebooks, according to the schedule shown in Figure 3.7. Fifteen combinations were tested (marked in red), inside of each the inner layer tuning was performed. The three remaining tests on the differenced filling levels were dropped, as this input variant had not yielded promising results.

The inner layer tuning was focused on the design and parametrisation of the network itself. An extensive hyperparameter grid search was automated using the KerasTuner framework, same as in the simple RNN case. The corresponding `build_model` and `design_model` functions are presented in Example 3.5.1 and Example 3.5.2, respectively.

Example 3.5.1.

```
1 def build_model(hp):
2     """
```

```

3  This function defines a parameter search space for tuning and
4  builds a model using an external function.
5  """
6  conv_layers      = hp.Int('conv_layers',
7                          min_value=1, max_value=3, step=1)
8  conv_units      = hp.Int('conv_units',
9                          min_value=4, max_value=16, step=4)
10 lstm1_units     = hp.Choice('lstm1_units',
11                             [4, 8, 16, 32])
12 lstm2           = hp.Boolean('lstm2')
13 lstm2_units     = hp.Choice('lstm2_units',
14                             [4, 8, 16, 32])
15 activation_hidden = hp.Choice("activation_hidden",
16                               ["relu", "tanh"])
17 pool           = hp.Boolean('pool')
18 dropout        = hp.Choice('dr',
19                             [0.0, 0.1, 0.2, 0.5])
20 recurrent_dropout = hp.Choice('rec_dr',
21                               [0.0, 0.1, 0.2, 0.5])
22 optimizer      = hp.Choice('optimizer',
23                             ['adam', 'rmsprop', 'adadelat'])
24 lr             = hp.Float('lr',
25                             min_value=1e-5, max_value=1e-2, sampling
26                             ="log")
27
28 # call an external model architecture specified within
29 design_model function
30 model = design_model(
31     conv_layers=conv_layers,
32     conv_units=conv_units,
33     lstm1_units=lstm1_units,
34     lstm2=lstm2,
35     lstm2_units=lstm2_units,
36     activation_hidden=activation_hidden,
37     dropout=dropout,
38     recurrent_dropout=recurrent_dropout,
39     pool=pool,
40     optimizer=optimizer,
41     lr=lr
42 )
43 return model

```

Example 3.5.2.

```

1  def design_model(conv_layers, conv_units, lstm1_units, lstm2,
2                    lstm2_units, activation_hidden, dropout, recurrent_dropout,
3                    pool, optimizer, lr):

```

```

2  """
3  This function builds a network architecture according to the
4  hyperparameter combination passed as an argument. Returns a
5  compiled model.
6  """
7  input_layer = Input(shape=(int(lookback / step), num_features)
8  )
9  conv_layers_list = []
10 for i in range(conv_layers + 1):
11     conv1 = Conv1D(filters=conv_units,
12                  kernel_size=3*i,
13                  strides=1,
14                  padding='same',
15                  activation=activation_hidden)(input_layer)
16     conv_layers_list.append(conv1)
17
18 concat = Concatenate()(conv_layers_list)
19
20 if pool:
21     pool1 = AveragePooling1D(pool_size=2)(concat)
22     lstm1 = LSTM(lstm1_units,
23                 return_sequences=lstm2,
24                 activation=activation_hidden,
25                 dropout=dropout,
26                 recurrent_dropout=recurrent_dropout)(pool1)
27 else:
28     lstm1 = LSTM(lstm1_units,
29                 return_sequences=lstm2,
30                 activation=activation_hidden,
31                 dropout=dropout,
32                 recurrent_dropout=recurrent_dropout)(concat)
33
34 if lstm2:
35     lstm2 = LSTM(lstm2_units,
36                 activation=activation_hidden,
37                 dropout=dropout,
38                 recurrent_dropout=recurrent_dropout)(lstm1)
39     output_layer = Dense(1)(lstm2)
40 else:
41     output_layer = Dense(1)(lstm1)
42
43 model = Model(inputs=input_layer,
44               outputs=output_layer)
45
46 if optimizer == 'adam':
47     optimizer = Adam
48 elif optimizer == 'rmsprop':

```

```

47     optimizer = RMSprop
48     elif optimizer == 'adadelata':
49         optimizer = Adadelata
50
51     model.compile(loss=MeanSquaredError(),
52                 optimizer=optimizer(learning_rate=lr),
53                 metrics=[MeanAbsoluteError(),
54                         RootMeanSquaredError(),
55                         MeanAbsolutePercentageError()])
54
55     return model

```

As shown in the examples above, multiple network architecture elements and model optimisation hyperparameters were tuned, including:

- **conv_layers**—the number of 1D convolutional layers (performing feature extraction), determining at the same time the kernel size in each layer
- **conv_units**—the number of units in each 1D CNN layer
- **lstms1_units**—the number of units in the first LSTM layer
- **lstm2**—the presence and potentially the number of units in the second LSTM layer
- **activation_hidden**—the activation function for hidden (i.e. CNN and LSTM) layers
- **dr** and **rec_dr**—the dropout and recurrent dropout rates for the LSTM layers
- **pool**—the presence of a 1D average pooling layer
- **optimizer**—the optimising algorithm
- **lr**—the learning rate used by the optimising algorithm

The values that different hyperparameters could take are specified in Example 3.5.1, while the network architecture setup is defined in Example 3.5.2. A visualisation of a sample 1DCNN-LSTM network is provided by Figure 3.8. In this example, the input layer feeds the data into four 1D convolutional layers, with kernel sizes determined by the successive powers of three⁵, the number of kernels in each

⁵Kernels of different sizes are supposed to detect patterns (in the data) of different temporal lengths—see Section 2.3.3.11.

layer specified by hyperparameter `conv_units` and activation functions determined by `activation_hidden`. Notice the `padding` and `strides` parameters in Example 3.5.2 were fixed at “same” and “1”, respectively—such combination ensured the equal length of all output sequences (from all 1D CNN layers), which was a precondition for concatenation procedure. Next, the activations from 1D CNNs are concatenated and fed into the pooling layer, which reduces the length of sequences (the size of the time dimension) by half. Subsequently, the pooling layer output is sent to the LSTM layer with `lstm1_units`. In this example, the `lstm2` hyperparameter must have been set to `False`, as there is no second LSTM layer. The last step is a fully connected layer with one node and a default (linear) activation function. The corresponding model summary is shown in Figure 3.9.

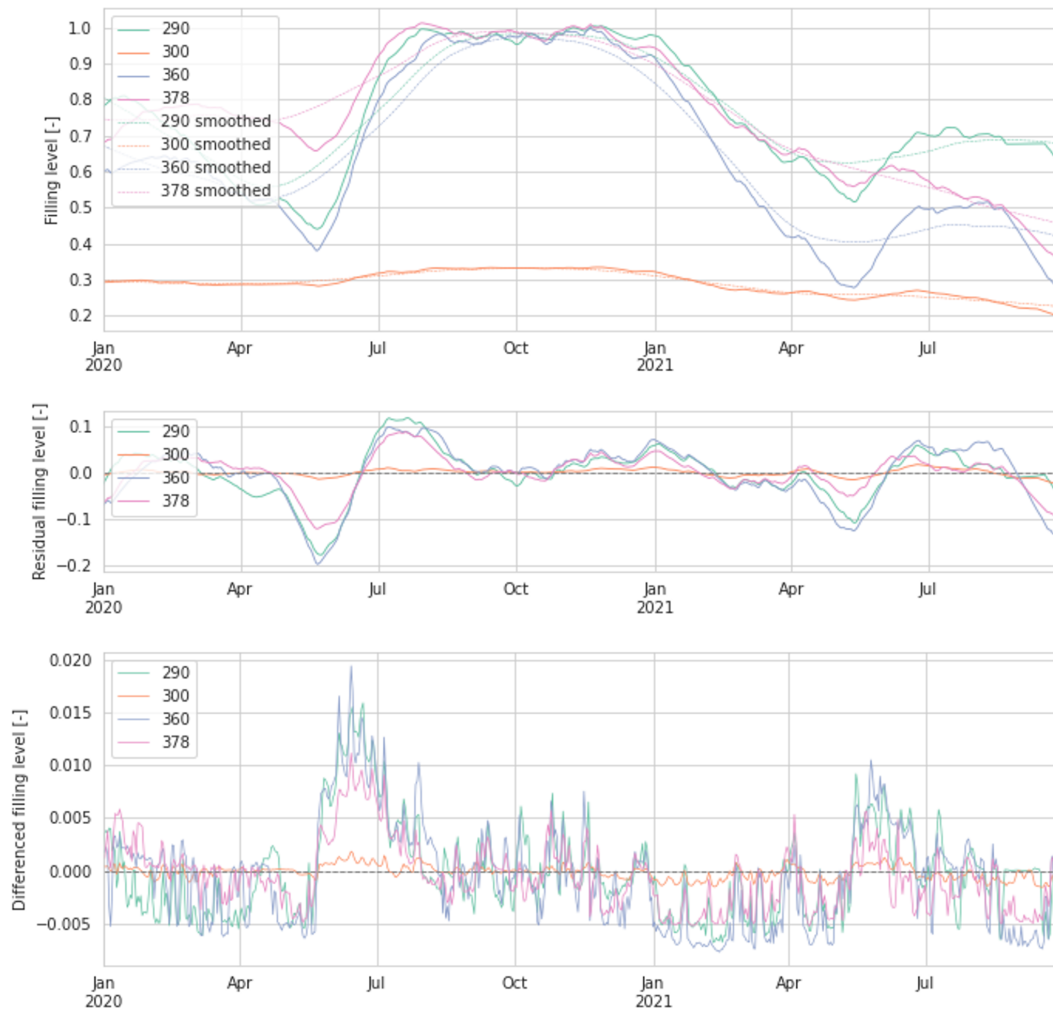


Figure 3.5: Input data transformations—a comparison of the original, residual and differenced filling level time series.



Figure 3.6: Input data transformation—various levels of smoothing applied on the SYS price time series. The smoothed curve in the upper plot corresponds to the smoothing window of 5 samples, in the lower plot—the window of 21 samples.

test #	filling_levels	smooth_SYS	delay	
Test01	original	5	14	Red bar indicating conducted tests
Test02	original	5	28	
Test03	original	5	42	
Test04	original	20	14	
Test05	original	21	28	
Test06	original	21	42	
Test07	residual	5	14	
Test08	residual	5	28	
Test09	residual	5	42	
Test10	residual	21	14	
Test11	residual	21	28	
Test12	residual	21	42	
Test13	differenced	5	14	
Test14	differenced	5	28	
Test15	differenced	5	42	
Test16	differenced	21	14	
Test17	differenced	21	28	
Test18	differenced	21	42	

Figure 3.7: Outer layer tuning schedule in the 1DCNN-LSTM model. The red colour on the right marks the conducted tests—the remaining ones were dropped, as the preceding tests that used the variant of differenced filling levels had not yielded promising results.



Figure 3.8: A sample 1DCNN-LSTM network architecture.

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 60, 5)]	0	[]
conv1d (Conv1D)	(None, 60, 4)	24	['input_1[0][0]']
conv1d_1 (Conv1D)	(None, 60, 4)	64	['input_1[0][0]']
conv1d_2 (Conv1D)	(None, 60, 4)	184	['input_1[0][0]']
conv1d_3 (Conv1D)	(None, 60, 4)	544	['input_1[0][0]']
concatenate (Concatenate)	(None, 60, 16)	0	['conv1d[0][0]', 'conv1d_1[0][0]', 'conv1d_2[0][0]', 'conv1d_3[0][0]']
average_pooling1d (AveragePooling1D)	(None, 30, 16)	0	['concatenate[0][0]']
lstm (LSTM)	(None, 8)	800	['average_pooling1d[0][0]']
dense (Dense)	(None, 1)	9	['lstm[0][0]']

=====
Total params: 1,625
Trainable params: 1,625
Non-trainable params: 0
=====

Figure 3.9: A sample 1DCNN-LSTM model summary.

3.6 Software and hardware

The experimental part of this thesis was conducted using Python 3.7.13 in the free version of Google Colaboratory⁶ (“Colab”) environment. Colab hosts a Jupyter notebook service with access to free computing resources, including standard CPU, as well as accelerated GPU and TPU runtimes. It is connected to the Google Drive data storage service, which ensures convenient input and output files exchange.

The most crucial Python libraries, from the thesis’ point of view, are listed in Table 3.1, including the information about their versions used in the project.

Table 3.1: The most important Python libraries used in the project

Library	Version
TensorFlow	2.8.2
Keras	2.8.0
scikit-learn	1.0.2
pandas	1.3.5
NumPy	1.21.6
statsmodels	0.10.2
matplotlib	3.2.2
seaborn	0.11.2

⁶<https://colab.research.google.com/>

Chapter 4

Results

This chapter presents the findings from the EDA, as well as the prediction results obtained with the three modelling approaches: ARIMAX, simple RNN and 1DCNN-LSTM. They are first described separately and later arranged together for comparison purposes.

4.1 Exploratory data analysis

The relationships between input dataset features are visualised via a correlation heatmap in Figure 4.1. Note that, in this thesis, “correlation” always refers to the Pearson correlation coefficient. The heatmap indicates that the filling level time series from the majority of reservoirs are highly positively correlated between each other, as expected. Furthermore, a general negative correlation between the amount of water in the storage reservoirs and the SYS price is confirmed by a noticeably different colour of the very bottom row of the heatmap, corresponding to the correlations of filling levels with the SYS price. Judging by the colour intensity of the cells in that row, it can be observed that the absolute correlation coefficients for the reservoirs with IDs between 287 and 378 are on average higher than for the rest of the reservoirs. This observation was considered in the process of limiting the number of reservoirs taken further into modelling. Additionally, a greedy manual feature selection was conducted using a basic RNN network with fixed parameters, to find out which reservoirs included in the input dataset yield best performance scores. The findings were supplemented by the results of backward feature selection tests—aiming at eliminating the reservoirs whose removal from the input dataset

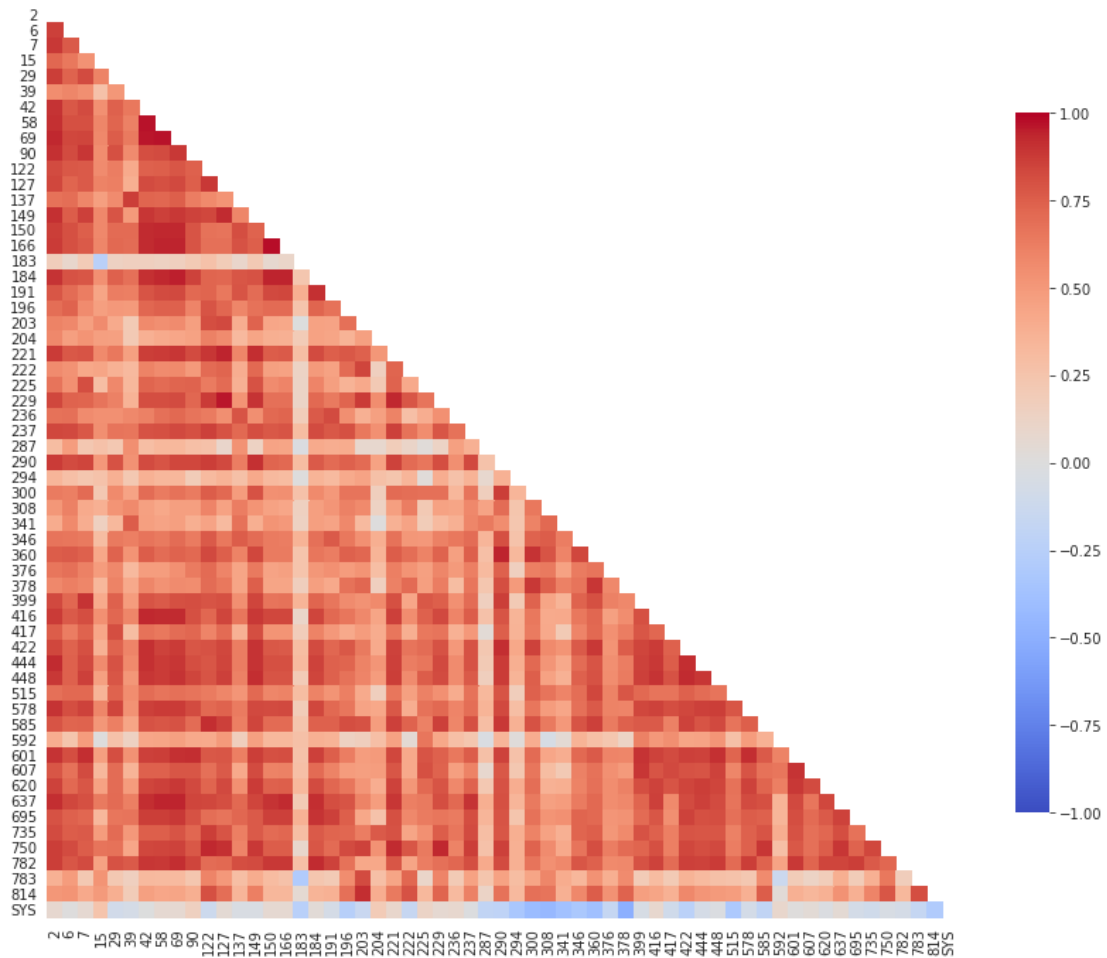


Figure 4.1: Heatmap of correlations between the raw input features.

yielded a positive or neutral model performance change. Finally, a practical aspect of choosing reservoirs that are currently monitored by EDInsights (or possible to monitor in the future—with the reservoir size being the limiting factor) was taken into account. Based on all the factors mentioned above, the input dataset has been limited to four filling level time series (reservoirs IDs: 290, 300, 360, 378) and the SYS price time series. Such dimensionality reduction allowed for conducting a more thorough hyperparameter tuning at the later stages.

An alternative technique which could be used to address the multicollinearity of the original set of filling level time series is a Principal Component Analysis (PCA). It is a common data dimensionality reduction tool which, in contrast to the feature selection process performed in this study, does not preserve the original features, but handles the data redundancy problem better.

The correlations illustrated in Figure 4.1 were calculated based on non-shifted time series. To investigate the predictive potential of filling level data, cross-correlations were calculated for each reservoir between the filling level and price time series for a set of shifts (lags), as presented in Figure 4.2. The column name *SYS_1w_ahead* means that the price time series was shifted 7 days in the negative direction—so that the filling levels from time step t were, after the shift, in the same row with the price from the original time step $t + 7$. In the resulting heatmap, the rows refer to subsequent reservoirs and are tagged with the reservoirs' IDs. The first column in Figure 4.2 corresponds to the last row of Figure 4.1. Further columns correspond to various SYS price time shifts. Focusing on the four reservoirs selected at the earlier step (290, 300, 360, 378), it can be observed that their noticeable correlation with the non-shifted SYS price drops significantly towards increasing time shifts. It suggests that the predictions are most likely to get more difficult for increasing time frames—which is rather intuitive. The reservoir no. 15 sticks out with its high positive cross-correlation values for the longest analysed time frames, but it probably constitutes an example of a spurious correlation. Looking at the heatmap as a whole, there is no apparent time shift that would stand out as one for which the price time series is highly correlated with the filling levels of a significant number of reservoirs. Hence, it did not lead to a conclusion about which time shift might potentially be promising as a prediction time frame. Thus, the prediction time frame was one of the hyperparameters tuned in the modelling process, with an upper limit restricted to six weeks—as the cross-correlations seem to fade significantly around and after that length.

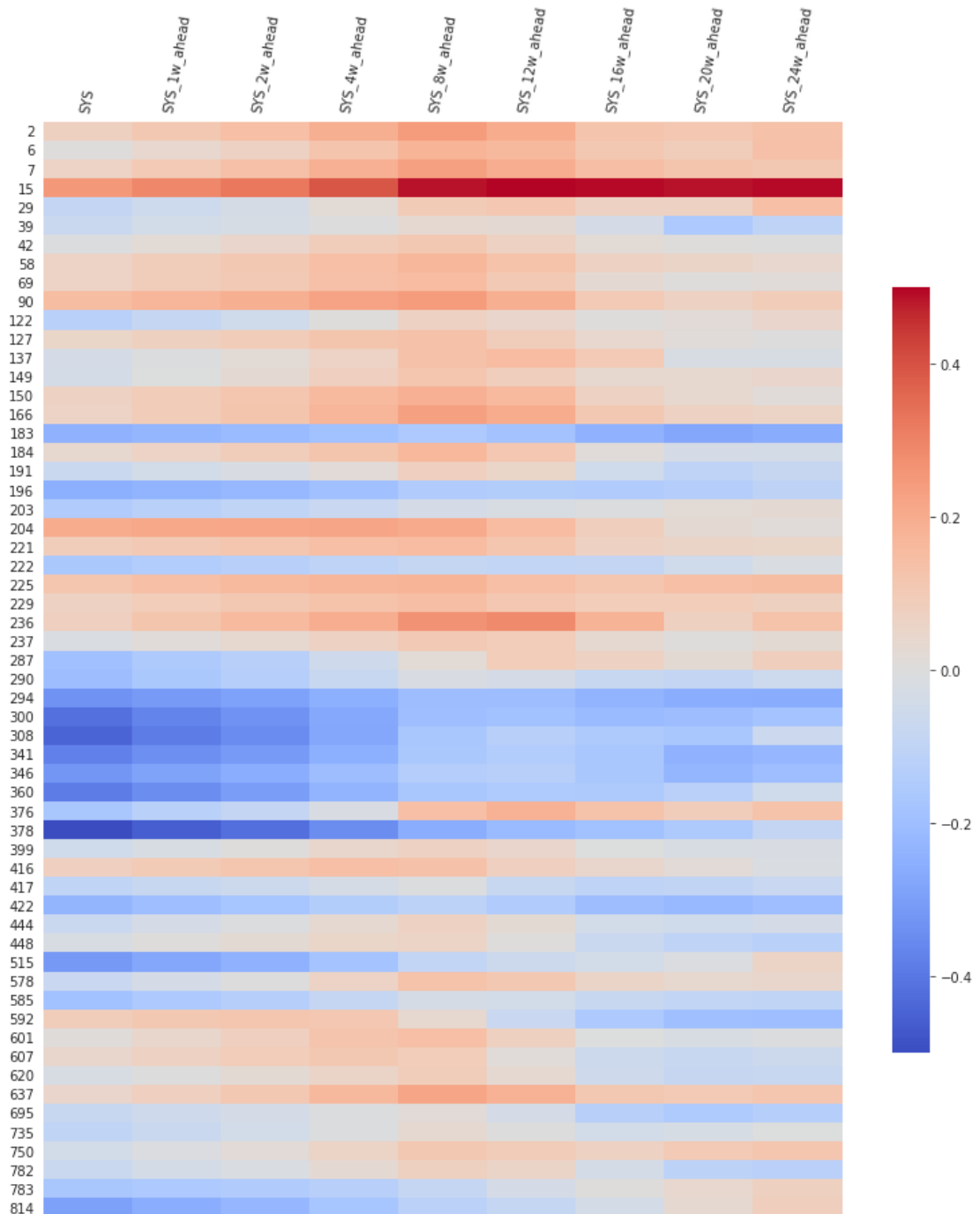


Figure 4.2: Cross-correlations between filling level time series and the SYS price time series shifted by various time lags. The rows correspond to different reservoirs and are tagged with the reservoir IDs.

4.2 ARIMAX model

Table 4.1 and Figure 4.3 summarise the results of the ARIMAX model. The former lists the performance scores and the latter plots the predictions made using the models which achieved the highest R^2 scores for the 14-day and 28-day prediction time frames. These particular time frames were found to be the most promising in the machine learning modelling, hence it is relevant to focus on the performance of the baseline ARIMAX model in these two variants. The input SYS price time series that yielded these scores was smoothed with the window of 21 samples and the optimal set of (p, d, q) parameters was found to be equal to $(1, 1, 1)$, in both cases.

Table 4.1: The best performing ARIMAX models

Model	RMSE	MAE	MAPE	R^2
ARIMAX 14 days ahead	6.61	5.40	9.96	0.80
ARIMAX 28 days ahead	11.32	9.29	15.57	0.39

Performance of the model with the shorter prediction time frame is better than for the longer prediction time frame. This is an expected effect, as the forecasting error accumulates when making a prediction for time step $t + 1$ based on the prediction for time step t . Looking at the plots, one can also observe that the shorter prediction time frame, the more the predicted time series resembles the original time series. That means that the autoregressive component of the model plays an important part in forecasting the future values, and that the exogenous variables are of little significance—which is by no means a desired effect.

Finally, it is important to remember that the ARIMAX model does not take the history of the exogenous variables (the filling levels) into account, but just the current filling levels for a given time step t . The autoregressive part of the model only looks at the history of the endogenous variable—the SYS price. The machine learning models, in contrast, will consider the historical component of all the input variables.

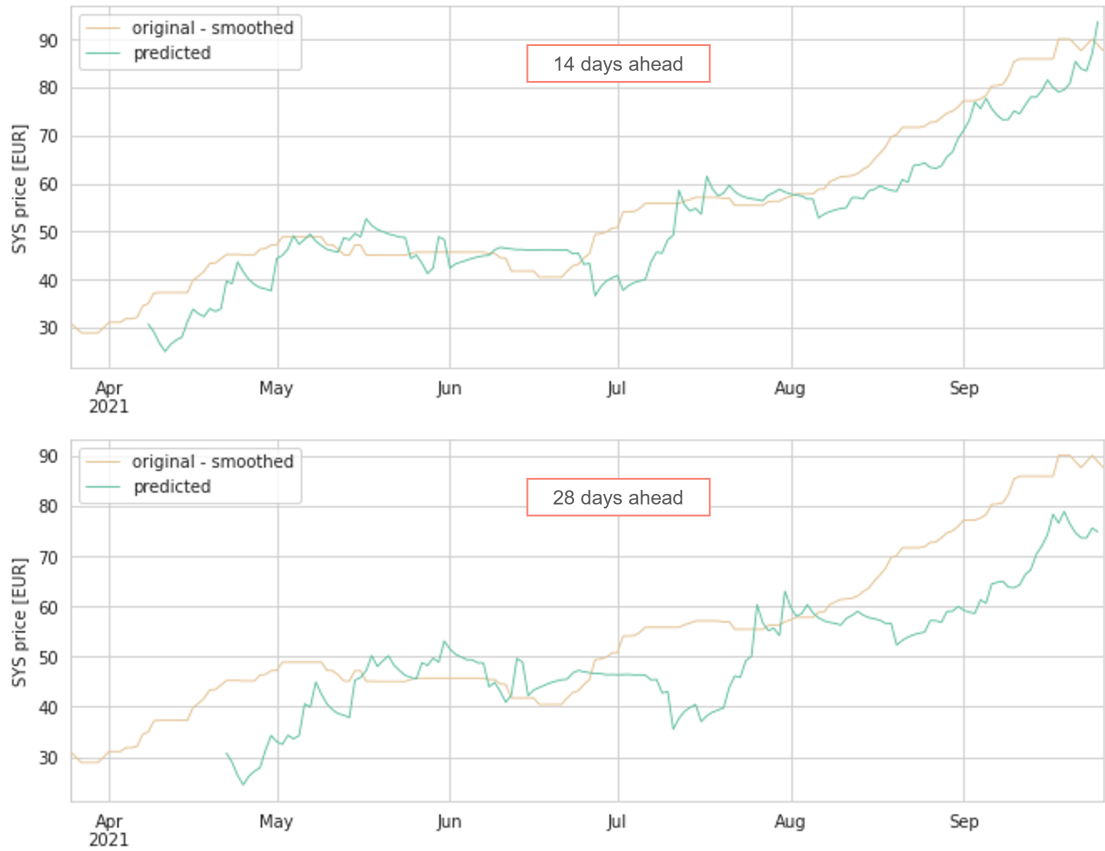


Figure 4.3: Best results of ARIMAX modelling for two prediction time frames—14 days ahead in the upper graph and 28 days ahead in the lower graph.

4.3 Basic RNN model

Two RNN models—one for the 14-day and one for the 28-day prediction time frame—which achieved the minimum validation loss scores in the hyperparameter tuning process are presented in Table 4.2, together with their test set performance scores. They were trained on the input dataset with the SYS price time series smoothed using a 21-sample window and the original filling level time series. The best combination of hyperparameters for both prediction time frames included the RNN layer with 32 units, tanh activation function and dropout rate of 0.0, as well as “Adam” optimiser and learning rate of the order of 0.001. The recurrent dropout rate was 0.5 in the winning 14 days ahead model and 0.0 in the 28 days ahead model.

Table 4.2: The best performing RNN models

Model	RMSE	MAE	MAPE	R ²
RNN 14 days ahead	8.89	6.94	11.71	0.64
RNN 28 days ahead	17.01	14.39	23.61	-0.35

Figure 4.4 illustrates training and validation learning curves for the two models, documented using the `TensorBoard` callback. As expected, the training losses (left column) are generally lower than validation losses (note the different scales of the y axes) and decrease monotonically with an increasing number of training epochs. The two graphs in the lower row correspond to the training and validation curves aggregated over consecutive epochs for the 28 days ahead model. One can notice that the model was only trained for 9 epochs and must have been stopped by the `EarlyStopping` callback due to insufficient improvement rate. The lowest validation loss was registered after the first epoch of training and the weights corresponding to this stage are the ones that the model was saved with. The upper row contains the corresponding curves for the 14 days ahead model, only this time two executions of model training with the same combination of hyperparameters were run. Repeating the training process multiple times with the same hyperparameter combination can reduce results variance and allows for a more accurate assessment of the model’s performance [60]. The execution in which the lowest validation loss was achieved yielded the model whose performance scores are discussed in this section.

Predictions made on the training, validation and testing intervals are visualised in Figure 4.5 and Figure 4.6, for the 14-day-delay and 28-day-delay models, respectively. The top graph in each figure shows the smoothed ground truth SYS price time series, as well as the predicted time series divided into training/validation and testing sets. The middle graph zooms into the smoothed and predicted price in the testing period, and the bottom one plots the residuals for the testing period (a difference between the observed and predicted data). It can be observed that the prediction quality drops significantly towards the end of the testing period, when the real SYS price values increased to 60–90 EUR—a level not registered at any time in the training or validation dataset. Intervals without predictions, at the beginning of the training/validation and testing sets, originate from the fact that a number of samples (equal to `lookback` parameter) before time step t is needed to calculate prediction for time step $t + \text{delay}$ (where `delay` is equal to the prediction time frame). Thus, the first prediction is made at time step number equal to the sum of `lookback` and `delay`, counting from the beginning of the dataset.

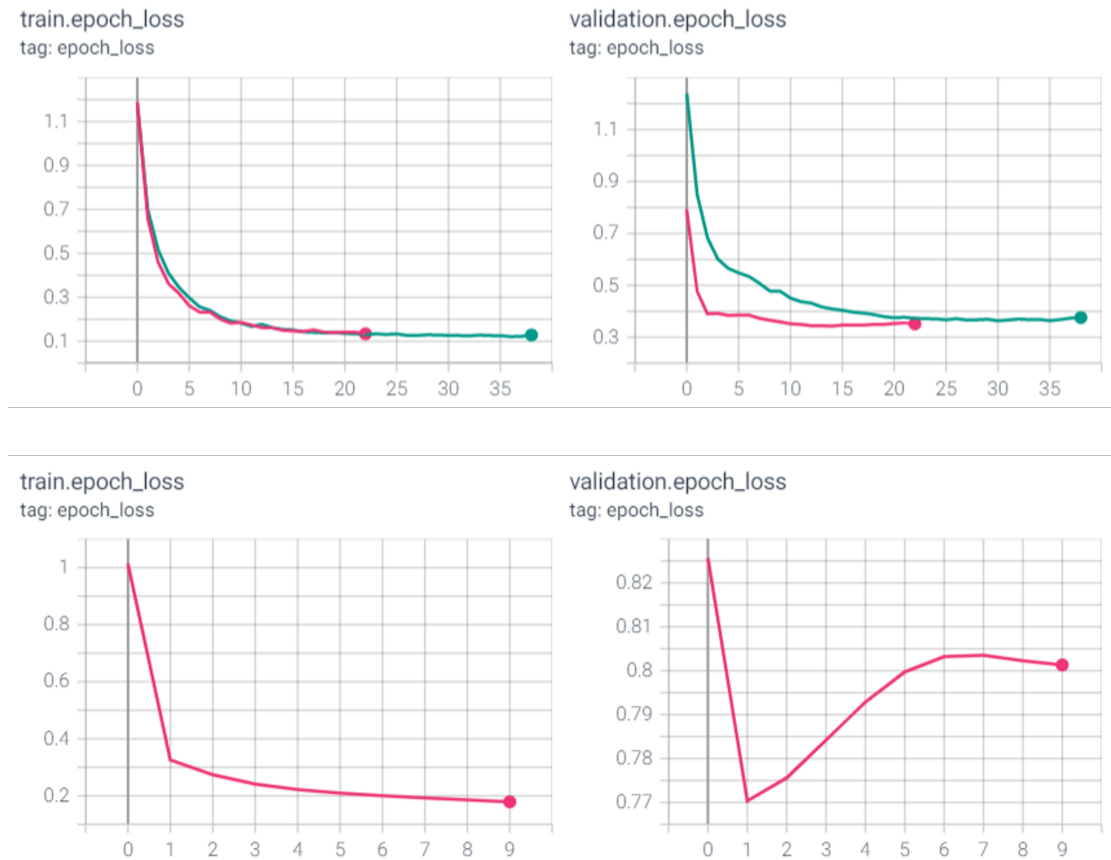


Figure 4.4: Training and validation learning curves of the best performing simple RNN models. Plots in the upper and lower row correspond to the 14-day and 28-day prediction time frame models, respectively. In the 14-day case, two executions of each individual hyperparameter combination were run.

The plots confirm that the 14-day prediction time frame yields a lower prediction error than the 28-day one. Same as in the ARIMAX case, similarities can be noticed between the ground truth and prediction curves, which suggest that the SYS price variable influences the predictions stronger than the filling level variables.

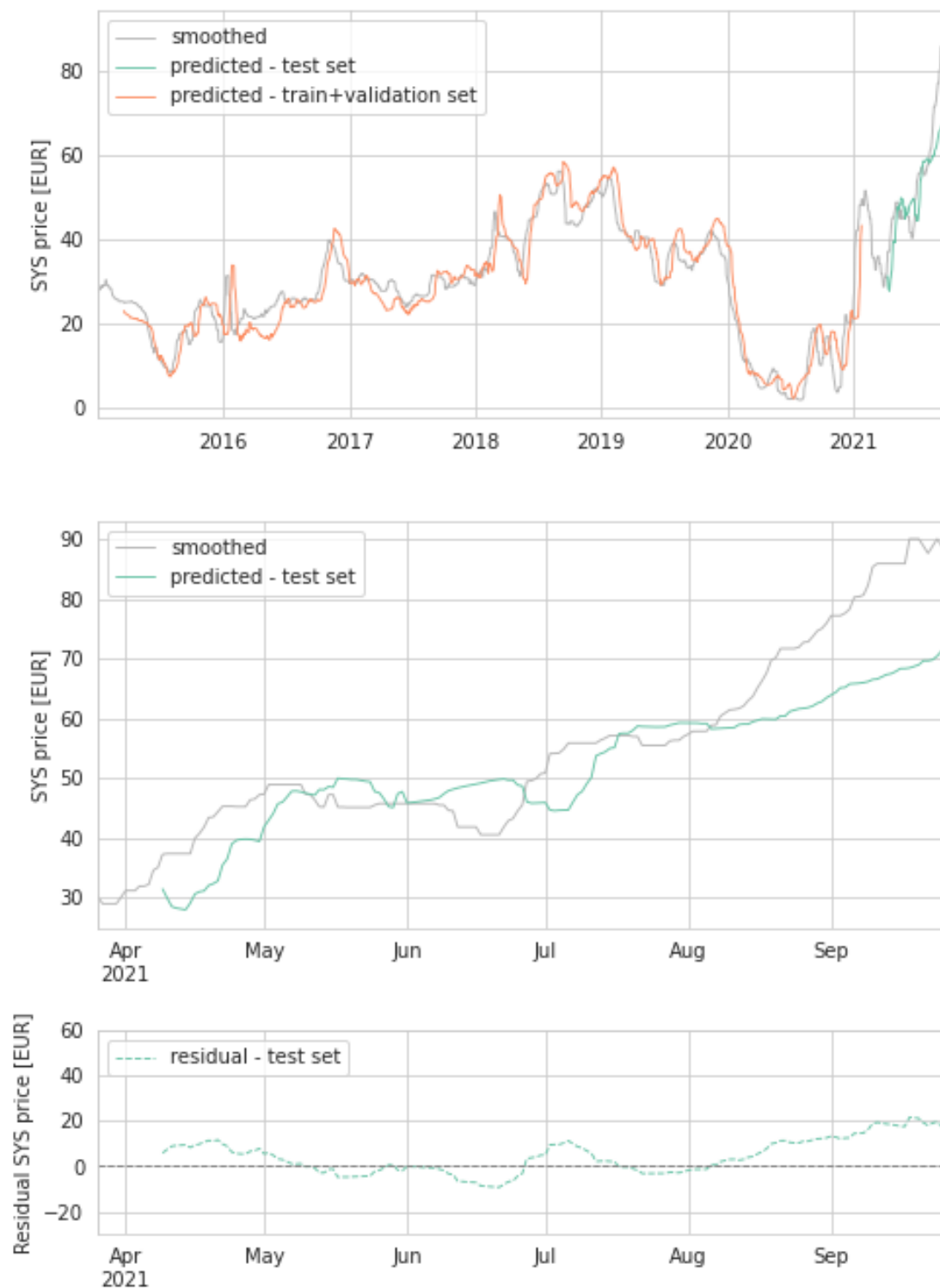


Figure 4.5: Predictions on the training, validation and testing intervals for the RNN model with the 14-day prediction time frame. In the top graph, the smoothed ground truth SYS price is plotted, together with the predicted SYS price divided into training/validation and testing sets. The middle graph is a zoom into the predictions on the test set, while the bottom one plots the test set residuals.

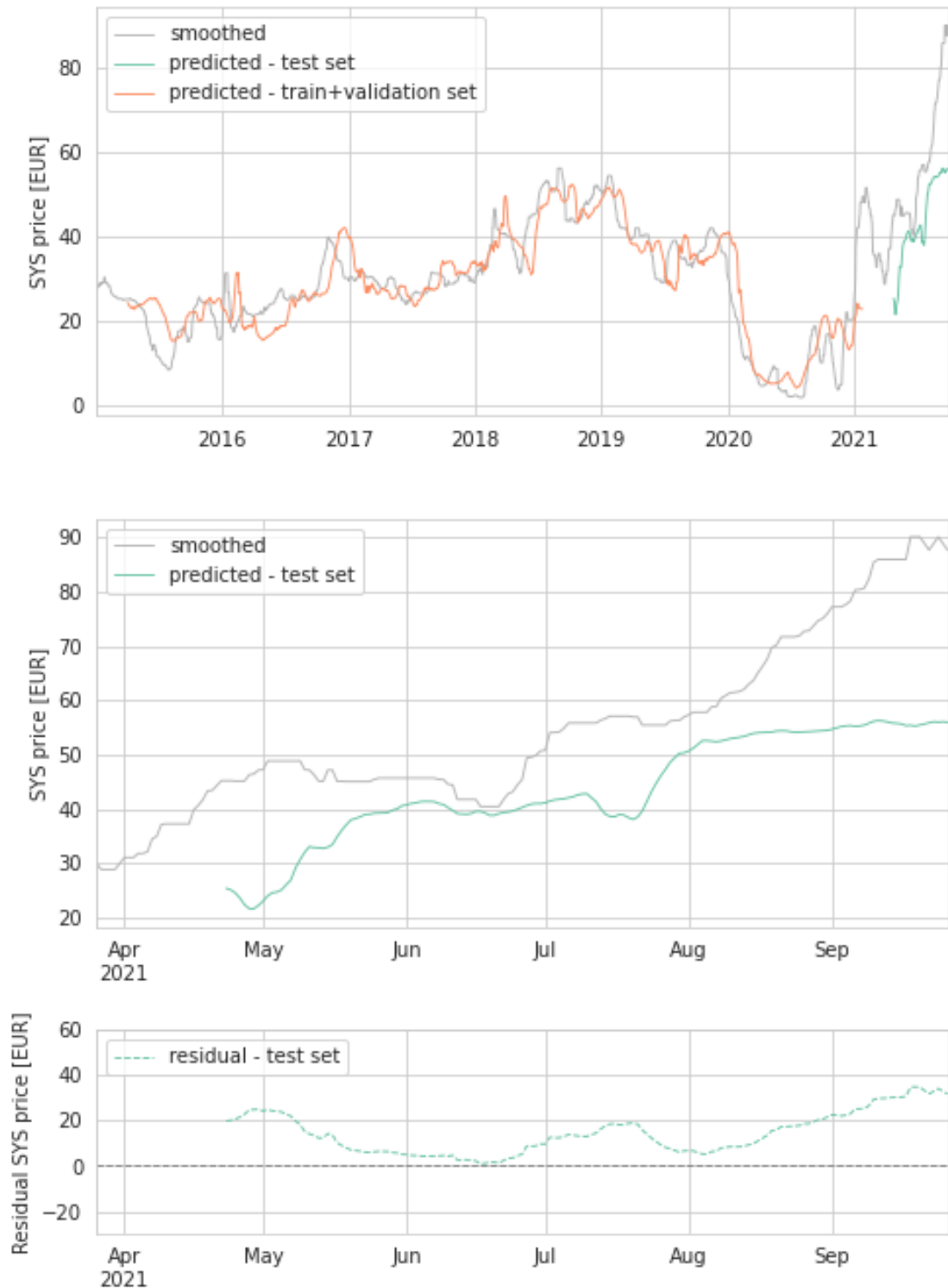


Figure 4.6: Predictions on the training, validation and testing intervals for the RNN model with the 28-day prediction time frame. In the top graph, the smoothed ground truth SYS price is plotted, together with the predicted SYS price divided into training/validation and testing sets. The middle graph is a zoom into the predictions on the test set, while the bottom one plots the test set residuals.

4.4 Hybrid 1DCNN-LSTM model

The results of the hyperparameter tuning of the 1DCNN-LSTM model are presented in Figure 4.7. The rows in the table correspond to the tests described in Figure 3.7. For a given test, the hyperparameters from the outer layer of tuning (i.e., input data transformations, LSTM input tensor shape) were fixed, while the hyperparameters under the *inner layer tuning* category correspond to the model that achieved the lowest validation loss score in the inner layer tuning process. The first four columns present a model’s predictive performance on the test set.

There are no obvious indications of any particular combination of hyperparameters dominating among the best performing models. The only hyperparameter that is common for all models listed in Figure 4.7 is the Adam optimiser. Another characteristic that is common for all models except one, is that they only had one LSTM layer—the `lstm2` hyperparameter was set to `False`. Apart from that, the rest of the hyperparameters took various values available within the search space.

The table in Figure 4.7 is divided into sections corresponding to the three tested prediction time frames—14-day, 28-day and 42-day. The models from each section that performed best on the test set are highlighted in green, together with their corresponding hyperparameter combinations. In two cases, the model with the lowest prediction errors is not the one with the highest R^2 score. In this study, it is preferred for the predictions to reflect the correct directions of the price change, rather than minimise the absolute errors (potentially at the cost of losing the change direction). Hence, it was decided that for the purpose of comparison with the models produced with the ARIMAX and simple RNN approaches, the best models will be selected according to the R^2 criterion. An exception was made for the prediction time frame of 42 days—even though the two models competing for the best model’s prize both performed noticeably poorly, the model with slightly a lower R^2 score is more stable (due to the larger input price smoothing window). Thus selected models, generated under the names of *test10*, *test08* and *test12*, are summarised in Table 4.3.

Table 4.3: The best performing 1DCNN-LSTM models

Model	RMSE	MAE	MAPE	R^2
1DCNN-LSTM 14 days ahead	10.66	8.02	12.92	0.49
1DCNN-LSTM 28 days ahead	16.68	13.26	21.03	0.02
1DCNN-LSTM 42 days ahead	19.88	17.45	27.78	-0.79

The learning curves of the three models were registered by `TensorBoard` and are shown in Figure 4.8. It can be observed that in all three cases the training was stopped well before reaching the maximum iterations limit set to 50. The curves exhibit short-term spikes, particularly in the early stages of training. Eventually, however, they all converge towards stable solutions.

The predictions on the training, validation and testing sets are visualised in Figure 4.9, Figure 4.10 and Figure 4.11, for the 14-day, 28-day and 42-day prediction time frames, respectively. Similarly as in the case of the ARIMAX and RNN models, prediction errors increase with the increasing prediction time frame. The 14-day and 42-day models were trained on the SYS price smoothed with a window of 21 samples, hence the predicted curve seems more stable than in the case of the 28-day model (where the SYS price smoothing window was set to 5 samples). In neither case was the model able to predict the steep increase of the price in the end of the training period.

	Performance on the test set					Outer layer tuning					Inner layer tuning							
	RMSE [EUR]	MAE [EUR]	MAPE [%]	R ²	filling_levels	SYS_smooth [window_len]	delay [days]	conv_layers	conv_units	lstm1_units	lstm2	lstm2_units	activation_hidden	pool	dr	rec_dr	optimizer	lr
14 days ahead																		
test01	14.8	11.7	19.41	0.24	original	5	14	1	16	4	FALSE	32	relu	FALSE	0.2	0.1	adam	0.010
test04	12.98	10.94	18.14	0.26	original	21	14	1	16	4	FALSE	8	relu	TRUE	0.5	0.1	adam	0.010
test07	18.33	14.71	23.37	-0.17	residual	5	14	1	4	32	FALSE	4	relu	TRUE	0.0	0.0	adam	0.010
test10	10.66	8.02	12.92	0.49	residual	21	14	1	4	16	FALSE	4	relu	FALSE	0.0	0.0	adam	0.010
test13	19.78	16.86	27.25	-0.36	differenced	5	14	1	12	32	FALSE	4	relu	FALSE	0.5	0.0	adam	0.002
28 days ahead																		
test02	19.72	15.05	22.75	-0.37	original	5	28	3	4	32	FALSE	4	tanh	FALSE	0.5	0.5	adam	0.010
test05	16.11	12	18.11	-0.21	original	21	28	3	12	16	TRUE	8	tanh	FALSE	0.0	0.0	adam	0.010
test08	16.68	13.26	21.03	0.02	residual	5	28	2	16	32	FALSE	4	relu	FALSE	0.0	0.5	adam	0.010
test11	18.18	14.84	23.38	-0.55	residual	21	28	2	16	32	FALSE	16	relu	TRUE	0.0	0.1	adam	0.010
test14	21.46	17.3	26.6	-0.62	differenced	5	28	2	4	32	FALSE	32	tanh	FALSE	0.0	0.1	adam	0.002
test17	17.12	14.98	24.33	-0.37	differenced	21	28	3	4	32	FALSE	4	relu	FALSE	0.0	0.5	adam	0.010
42 days ahead																		
test03	27.85	23.8	36.2	-1.6	original	5	42	3	4	8	FALSE	4	relu	TRUE	0.5	0.5	adam	0.010
test06	22.54	20.04	31.89	-1.3	original	21	42	2	8	32	FALSE	32	relu	TRUE	0.1	0.5	adam	0.010
test09	23.02	20.56	33.11	-0.78	residual	5	42	3	12	4	FALSE	32	relu	TRUE	0.0	0.0	adam	0.010
test12	19.88	17.45	27.78	-0.79	residual	21	42	1	16	32	FALSE	8	relu	FALSE	0.0	0.2	adam	0.010

Figure 4.7: The results of the outer and inner layers of tuning of the 1DCNN-LSTM model.

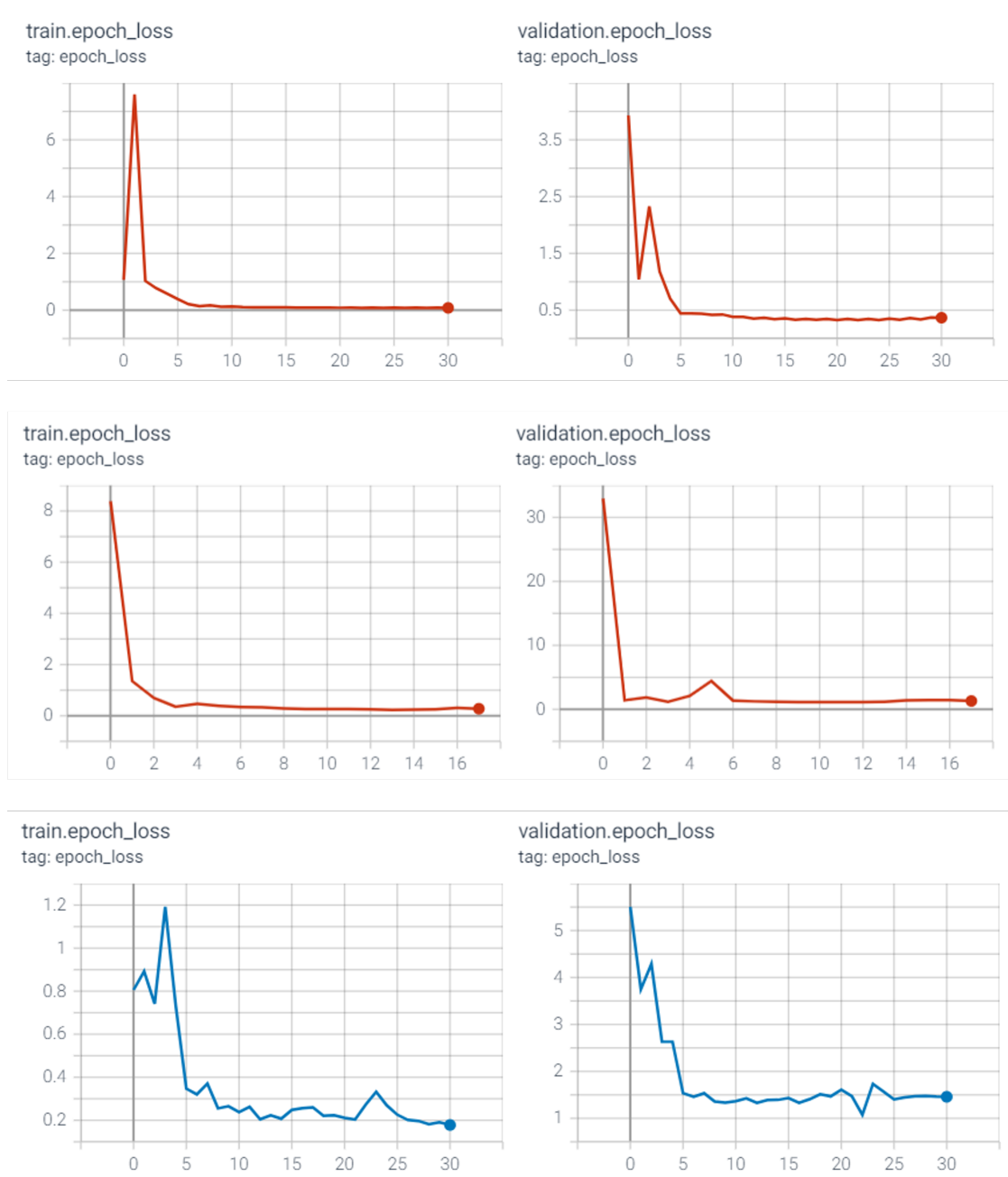


Figure 4.8: Training and validation learning curves of the best performing 1DCNN-LSTM models. Starting from the top row, the plots correspond to the 14-day, 28-day and 42-day prediction time frame models, respectively.

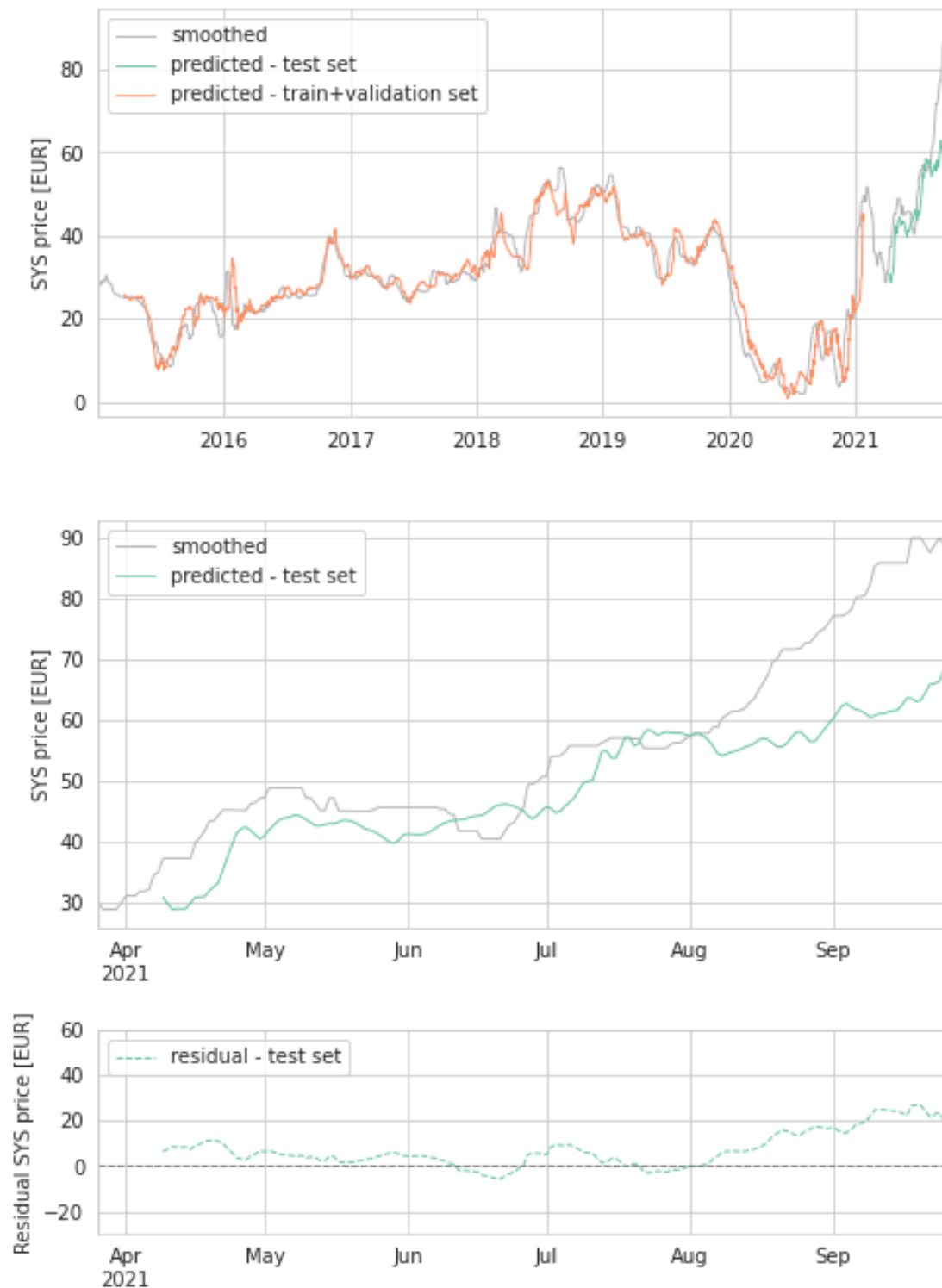


Figure 4.9: Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 14-day prediction time frame. In the top graph, the smoothed ground truth SYS price is plotted, together with the predicted SYS price divided into training/validation and testing sets. The middle graph is a zoom into the predictions on the test set, while the bottom one plots the test set residuals.

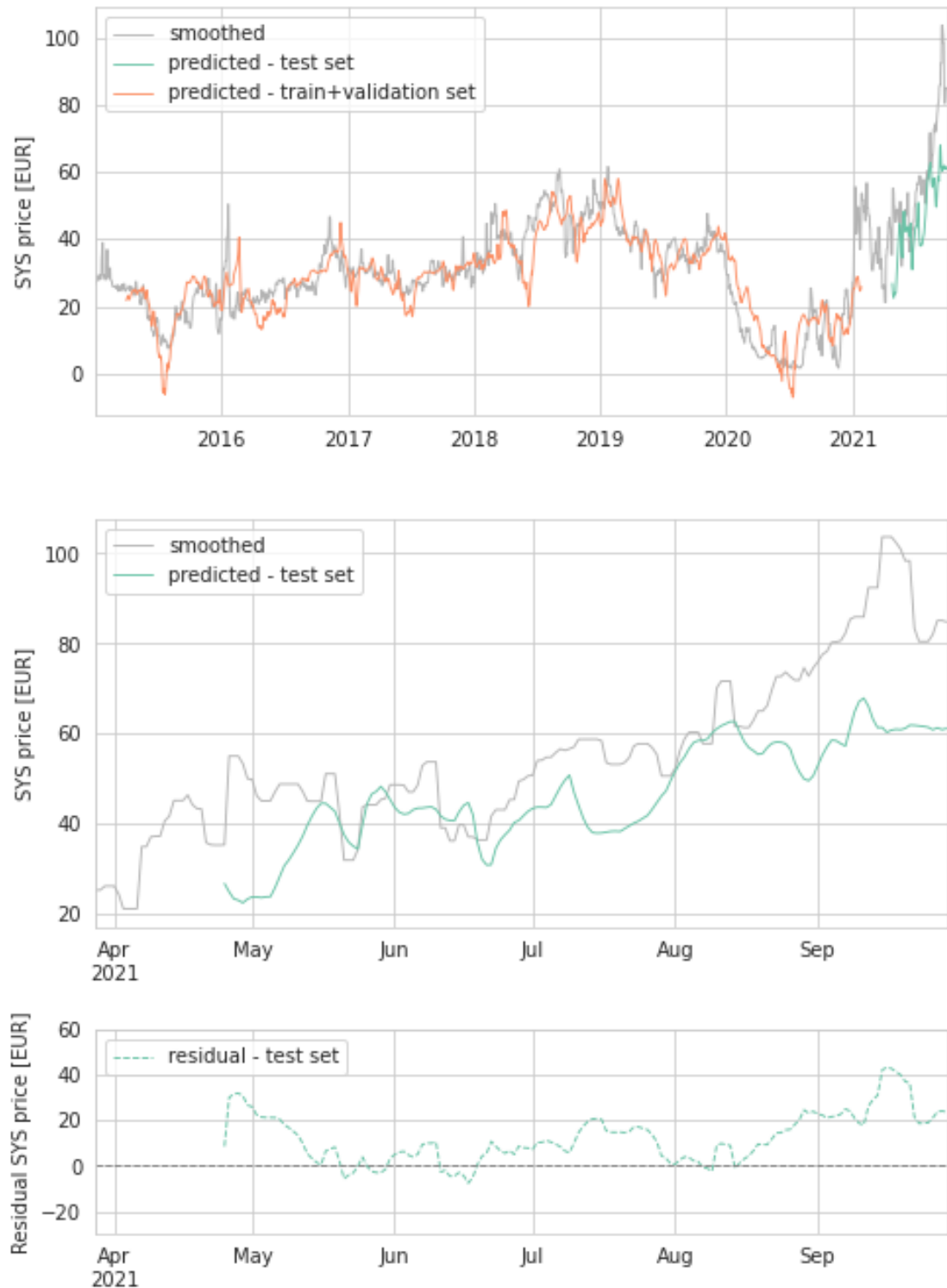


Figure 4.10: Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 28-day prediction time frame. In the top graph, the smoothed ground truth SYS price is plotted, together with the predicted SYS price divided into training/validation and testing sets. The middle graph is a zoom into the predictions on the test set, while the bottom one plots the test set residuals.

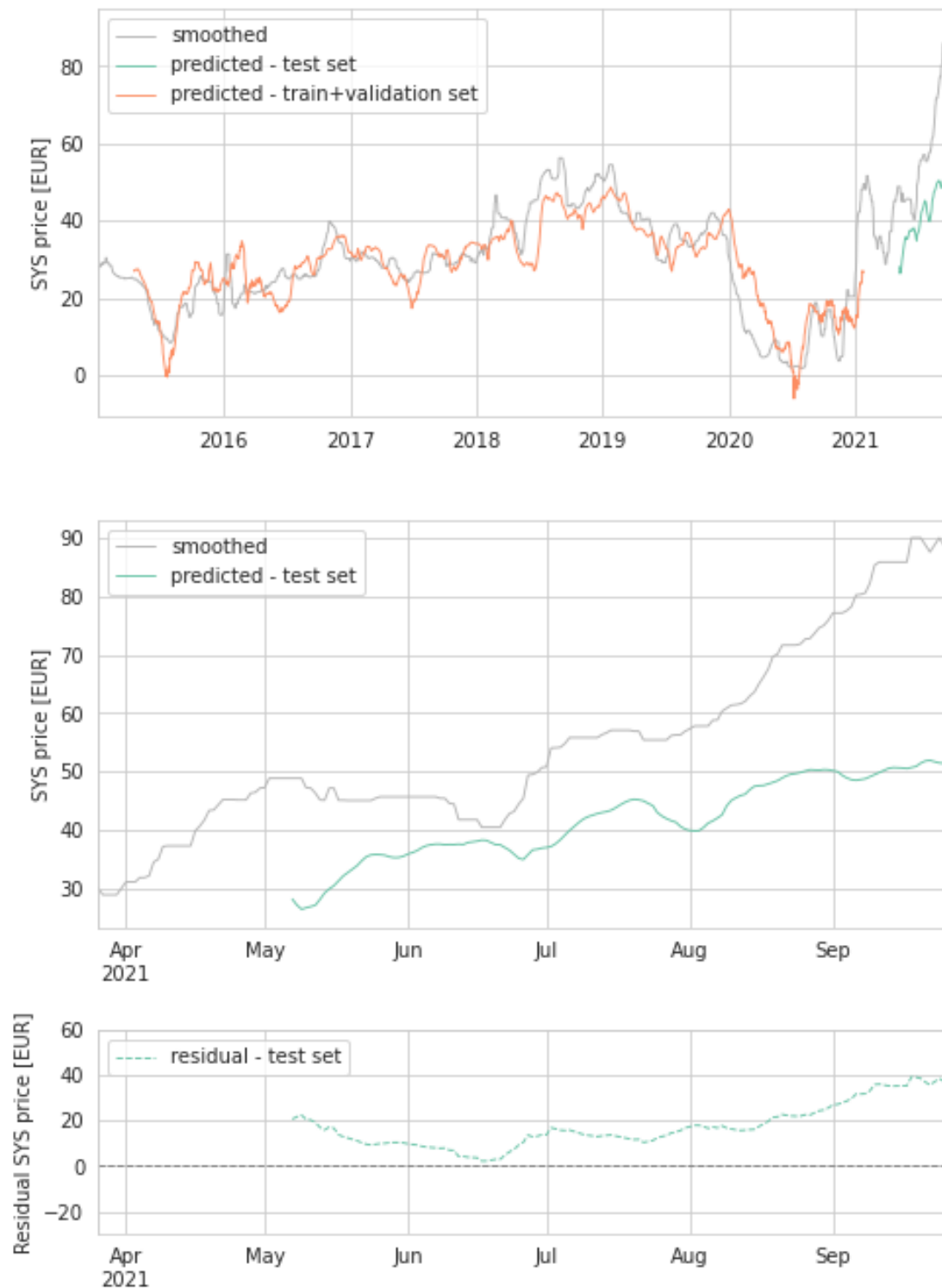


Figure 4.11: Predictions on the training, validation and testing intervals for the 1DCNN-LSTM model with the 42-day prediction time frame. In the top graph, the smoothed ground truth SYS price is plotted, together with the predicted SYS price divided into training/validation and testing sets. The middle graph is a zoom into the predictions on the test set, while the bottom one plots the test set residuals.

4.5 Summary

Table 4.4 and Table 4.5 summarise the best performing models of each category (ARIMAX, RNN, 1DCNN-LSTM) for two prediction time frames, 14-day and 28-day, respectively. In both cases it was the ARIMAX model which yielded the best predictions on the test set, achieving the highest R^2 scores equal to 0.8 (14-day) and 0.39 (28-day) and the lowest predictions errors. The machine learning models performed worse, which must to a large extent be related to their inability to catch the steep upward price trend by the end of the test period. The RNN 28-day variant achieved a negative R^2 score, which translates to the model yielding a worse match than a straight line would yield.

Table 4.4: Summary of the best performing 14 days ahead models

Model	RMSE	MAE	MAPE	R^2
ARIMAX 14 days ahead	6.61	5.40	9.96	0.80
RNN 14 days ahead	8.89	6.94	11.71	0.64
1DCNN-LSTM 14 days ahead	10.66	8.02	12.92	0.49

Table 4.5: Summary of the best performing 28 days ahead models

Model	RMSE	MAE	MAPE	R^2
ARIMAX 28 days ahead	11.32	9.29	15.57	0.39
RNN 28 days ahead	17.01	14.39	23.61	-0.35
1DCNN-LSTM 28 days ahead	16.68	13.26	21.03	0.02

Chapter 5

Discussion

As demonstrated in the previous chapter, the statistical ARIMAX models outperformed the machine learning models. This proves that it should not be taken for granted that the more computationally expensive approaches always yield better results.

In the previous chapter, the performance of various developed models was presented using quantitative performance metrics, such as error functions (RMSE, MAE, MAPE) and the R^2 score. However, this way of assessing the performance and applicability of the model in a real setting is not sufficient when there is no a priori baseline model that we could refer to. One needs to be aware that the quantitative performance scores reflect how the predictions match the observed values over the entire period of time that predictions were calculated for, e.g. the length of the test set. However, to understand the quality of the models developed in this thesis we need to consider how they can be used in practise. Instead of an averaged performance over the entire prediction set, we need to take a closer look at how the models perform within the range of the prediction time frame, at individual points in time.

5.1 Interpretation of the results

Let us think about how the information available in the form of prediction plots from the previous chapter should be read. On a given day t , the prediction is made for day $t + \text{delay}$ —in our case 14 or 28 days ahead. If we then look at the prediction

plots, we need to imagine that at a given day t we only have predictions available for the next 14 or 28 days (assuming that predictions were being calculated with every new observation—ideally daily, for the preceding 14 or 28 days, at least). Locally, within the prediction time frame, the predictions may not reach the accuracy suggested by the performance metrics calculated on the entire dataset. Let us check how the models that achieved the highest scores (the ARIMAX models) perform locally. Figure 5.1 presents the prediction plots known from Section 4.2, only this time there are additional arrows drawn along the original and predicted curves. They are supposed to aid the visualisation of local consistencies or inconsistencies (blue and red color, respectively) between the original and predicted SYS price curves. Their length approximates the prediction time frames. If the arrows are red, one of the curves increases and the other decreases locally, within the prediction time frame. Such situation is clearly undesirable from the perspective of, e.g. making a trading decision—for which a decision maker needs to know whether the price will go up or down. The arrows were placed in just a few positions in each graph, for visibility reasons, but if one analyses more locations it becomes clear that this level of accuracy has a moderate applicability in any trading activity. Furthermore, if we consider shorter-term trends, the divergence between the directions that the original and predicted curves take is even more apparent.

It is worth noting, that the ARIMAX models perform decently well in the intervals where the price maintains a constant trend for a longer period of time (e.g. August and September). This might be, however, related to the fact that the ARIMAX model was set up to use just a single autoregressive component ($p = 1$)—the prediction for the succeeding time steps was strongly influenced by one directly preceding time step. This enables the model to focus on the steeply increasing price. The machine learning models, on the other hand, were trained to use (in various variants) up to around 60 preceding time steps, which might have taught them to look for longer-term trends in the data, but apparently did not enable them to compete with the ARIMAX model on this dataset. In general, as observed in the previous chapter, practically all prediction plots resemble (to a smaller or larger extent) the historical price curves, slightly shifted in time by the value of the corresponding prediction time frames. This is not a desirable effect, as it can mean that the models are mostly driven by the SYS price feature, instead of finding relationships between the changes of the filling levels of the hydropower reservoirs and the price movements. Machine learning is particularly powerful when there are multiple variables that impact the response variable, especially in a non-linear manner. In this case, it appears that the algorithms found out that the loss is best minimised through focusing on the past values of the response variable itself. However, they failed to outperform the ARIMAX method, which specialises in

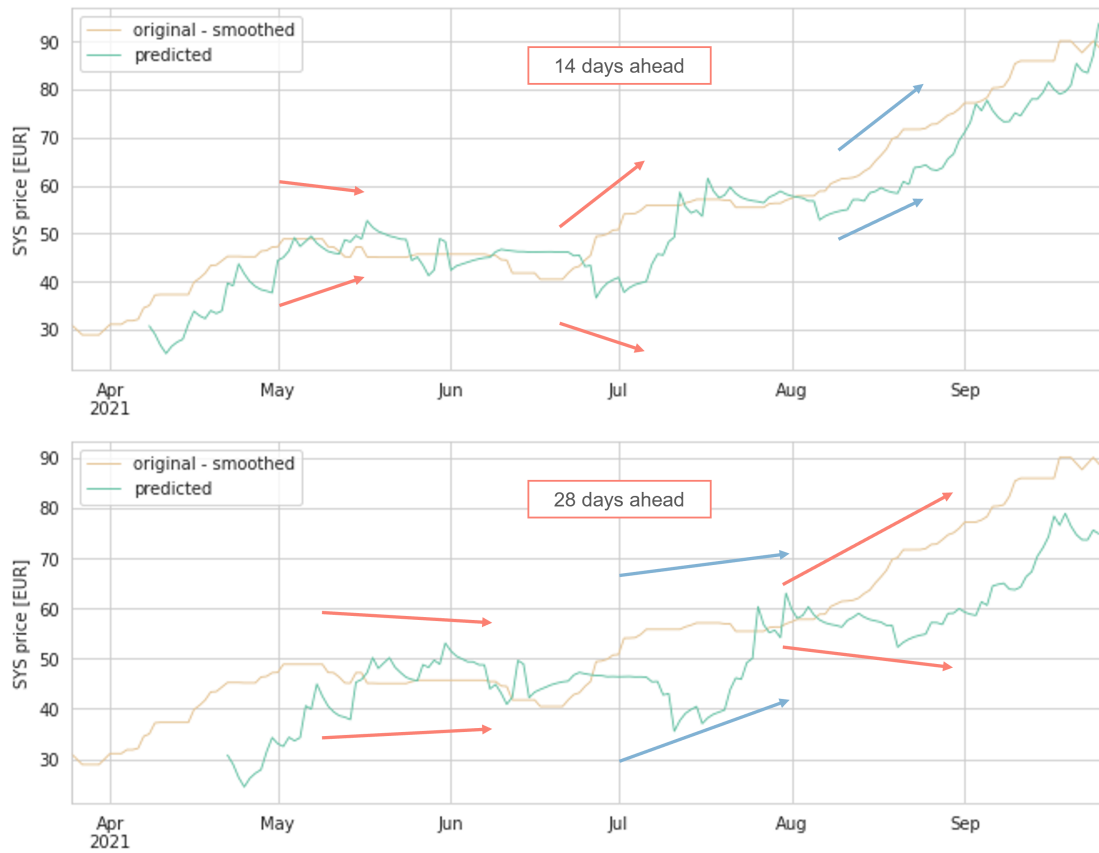


Figure 5.1: Analysis of ARIMAX prediction plots. Red arrows signalise that one of the curves increases and the other decreases in the interval limited by the arrows' length. Blue arrows mark the intervals where both curves follow the same direction of change.

utilising the autoregressive information.

5.2 Shortcomings of the current approach

There are several factors that might have negatively affected the performance of the ML models—multiple error modes of different origins could have been accumulated across different stages. Firstly, as noted in the introductory part of the thesis, the SYS price used a ground truth in this project is a real, historical time series. The hydro producers, however, run the production according to the price forecasts. When developing a model supposed to reconstruct relationships between the filling levels and the future power price, it is the forecasted price that should serve as a ground truth, not the real historical data. Any divergence between these two is expected to negatively affect the model training process. Unfortunately, we did not have access to the hydro producer’s forecast.

Secondly, the filling levels of the hydropower reservoirs are affected not only by the hydropower production, but also by the inflow of water. It might be worth considering supplementing the input dataset with features that could add information about the inflow, e.g. meteorological data.

Finally, we need to remember that for the machine learning algorithm to be trained on the train set and predict accurately on the test set, the data in both sets should be sampled from the same distribution—the mean and variance of data should remain constant within both sets. This assumption is violated in the case of power markets the moment any extreme geopolitical or weather-related events occur. As Chollet notes in [37, Chapter 6], forecasting the markets’ behaviour is extremely difficult. The past is usually not a good predictor of the future in the markets’ case, as opposed to, e.g. forecasting natural phenomena, such as weather.

5.3 Alternative approaches

After analysing the obtained results, several alterations of the current methodology, as well as potential extensions of the study come to mind.

Starting from the changes that should be introduced in the current setup—a completely basic, naive baseline model should be created. With this model, the predictions would be calculated by shifting the exact values from time step t to $t + delay$. Performance metrics obtained on thus calculated predictions would constitute a reference for any statistical or machine learning model developed at a later stage of the project.

Furthermore, the input dataset was divided into the training, validation and testing set chronologically, resulting in the testing set falling onto the period of anomalously high power prices, at the level unseen by the ML algorithms during the training. A direct consequence is the poor performance score of the ML models. [61] describes cross-validation techniques for time series data that could be explored for our dataset. A simple example involves developing a model on a small training subset of data and testing it on a small, directly succeeding subset. Next, both subsets from the previous step constitute a bigger training set used to train a new model, which is then tested on another small testing subset, consisting of directly succeeding samples. This procedure is repeated several times, until reaching the end of the original dataset. Then, the performance scores from different testing subsets are averaged to obtain an overall testing score.

To further investigate the actual contribution of the reservoir filling level time series in the model, compared to the contribution of the autoregressive price component, one could try to remove the price time series from the set of feature variables and treat it exclusively as a response (target) variable. Such a move should reveal the extent of the predictive power of the filling level data. Additionally, a Principal Component Analysis could be performed on the filling level time series to remove the redundant information present in the original set of input reservoirs and thus reduce the dimensionality of the input dataset. In case of the ARIMAX model, it would be worth to take a closer look at the parameters of the fitted models (the coefficients) to better understand the contribution of each model component, including the filling levels.

A potential extension of this study would be to gather and utilise expert knowledge of the characteristics of various reservoirs. The main feature that should be explored is the storage cycle of various reservoirs. Some can be emptied and refilled several times during a season, some are used to save the water over years, to secure the supply for a “dry” year. Using such an information to select the reservoirs for the input dataset and creating an additional feature that would inform about the current month or a season of the year could potentially allow for discovering valuable patterns.

Finally, it would be worth considering to modify the current price forecasting approach (regression task) into a trend forecasting approach (classification task) described in [44]. The output of such an alternative approach would be a label informing about the direction of the price change (e.g. *increase* or *decrease*), instead of the actual price prediction.

Chapter 6

Conclusion

This work explored the potential of using the hydropower reservoirs filling level data and the historical power price data in the problem of forecasting the future power price (particularly, the Nordic system price). Three modelling approaches were tested—a statistical method ARIMAX, a machine learning model with an RNN layer as a main building block and a hybrid machine learning model combining 1D-CNN layers with LSTM layers. Despite an extensive hyperparameter tuning in the machine learning approaches, the statistical method—although less computationally demanding—outperformed the machine learning models in the predictive performance on the testing set. The suspected reason is that the autoregressive price component was prioritised over the filling levels information by all the models. Despite reaching the R^2 score of 0.8 for the 14-day prediction time frame, we believe that even the ARIMAX model is not sufficiently accurate, at this stage of development, to be relied upon when making decisions related to, e.g. trading in the power market. We provide a set of suggestions for future improvements and alternative approaches that can be investigated in order to further boost the performance of the models.

Bibliography

- [1] *Glossary:Renewable energy sources*, https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Renewable_energy_sources, Accessed: 2022-06-09.
- [2] *Renewable energy explained*, <https://www.eia.gov/energyexplained/renewable-sources/>, Accessed: 2022-06-09.
- [3] *3-reasons-why-nuclear-clean-and-sustainable*, <https://www.energy.gov/ne/articles/3-reasons-why-nuclear-clean-and-sustainable>, Accessed: 2022-06-08.
- [4] *A European Green Deal*, https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal_en?lang=en, Accessed: 2022-06-08.
- [5] *What is the source of the electricity we consume?* <https://ec.europa.eu/eurostat/cache/infographs/energy/bloc-3b.html?lang=en>, Accessed: 2022-06-08.
- [6] *The History of Norwegian Hydropower in 5 Minutes*, <https://www.regjeringen.no/en/topics/energy/renewable-energy/the-history-of-norwegian-hydropower-in-5-minutes/id2346106/>, Accessed: 2022-06-09.
- [7] T. A. Johnsen, 'Demand, generation and price in the norwegian market for electric power,' *Energy Economics*, vol. 23, no. 3, pp. 227–251, 2001, ISSN: 0140-9883. DOI: [https://doi.org/10.1016/S0140-9883\(00\)00052-9](https://doi.org/10.1016/S0140-9883(00)00052-9). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0140988300000529>.
- [8] *The power market - Energifakta Norge*, <https://energifaktanorge.no/en/norsk-energiforsyning/kraftmarkedet/>, Accessed: 2022-06-11.
- [9] *The electricity market*, https://www.regjeringen.no/globalassets/upload/oed/pdf_filer/faktaheftet/evfakta08/evfacts08_kap07_eng.pdf, Accessed: 2022-06-11.

- [10] *The power market*, <https://www.nordpoolgroup.com/en/the-power-market/>, Accessed: 2022-06-11.
- [11] *Hydroelectric Power*, <https://www.usbr.gov/power/edu/pamphlet.pdf>, Accessed: 2022-06-12.
- [12] ‘Time series analysis,’ in *The R Book*. John Wiley & Sons, Ltd, 2012, ch. 24, pp. 785–808, ISBN: 9781118448908. DOI: <https://doi.org/10.1002/9781118448908.ch24>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118448908.ch24>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118448908.ch24>.
- [13] A. Tealab, ‘Time series forecasting using artificial neural networks methodologies: A systematic review,’ *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 334–340, 2018.
- [14] *Om magasinstatistikken - NVE*, <https://www.nve.no/energi/analyser-og-statistikk/om-magasinstatistikken/>, Accessed: 2022-06-13.
- [15] *Market data*, <https://www.nordpoolgroup.com/en/Market-data1/#/nordic/table>, Accessed: 2022-06-09.
- [16] *Retningslinjer for registrering av vannstand i regulerte vannmagasin*, https://www.nve.no/Media/4561/5_retn-magasin vannstand_20062016.pdf, Accessed: 2022-06-02.
- [17] *The electricity grid - Energifakta Norge*, <https://energifaktanorge.no/en/norsk-energiforsyning/kraftnett/>, Accessed: 2022-06-14.
- [18] *Regulation of grid operations - Energifakta Norge*, <https://energifaktanorge.no/en/regulation-of-the-energy-sector/regulering-av-nettvirksomhet/>, Accessed: 2022-06-14.
- [19] *What is spot power trading?* <https://www.kyos.com/faq/what-is-spot-power-trading/>, Accessed: 2022-06-16.
- [20] *Single Day-ahead Coupling (SDAC)*, https://www.entsoe.eu/network_codes/cacm/implementation/sdac/, Accessed: 2022-06-16.
- [21] *EUPHEMIA Public Description*, <https://www.nordpoolgroup.com/globalassets/download-center/single-day-ahead-coupling/euphemia-public-description.pdf>, Accessed: 2022-06-15.
- [22] *Bidding areas — Nord Pool*, <https://www.nordpoolgroup.com/en/the-power-market/Bidding-areas/>, Accessed: 2022-06-15.
- [23] *Price calculation — Nord Pool*, <https://www.nordpoolgroup.com/en/trading/Day-ahead-trading/Price-calculation/>, Accessed: 2022-06-14.

- [24] J.-M. Roldan-Fernandez, M. Burgos-Payan, J.-M. Riquelme-Santos and A.-L. Trigo-Garcia, ‘The merit-order effect of energy efficiency,’ *Energy Procedia*, vol. 106, pp. 175–184, 2016, Energy Economics Iberian Conference, EEIC 2016, 4-5 February 2016, Lisbon, Portugal, ISSN: 1876-6102. DOI: <https://doi.org/10.1016/j.egypro.2016.12.114>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1876610216316721>.
- [25] I. Micic, ‘Day-Ahead Markets,’ The Physical and Financial Power Markets - Workshop by Nord Pool Academy, Jan. 2022.
- [26] *Intraday trading: Definition, theory and practice*, <https://www.next-kraftwerke.com/knowledge/intraday-trading>, Accessed: 2022-06-16.
- [27] *Introduction to electricity markets, its balancing mechanism and the role of renewable sources - Jesus Lago*, <https://jesuslago.com/introduction-to-the-electrical-markets/>, Accessed: 2022-06-16.
- [28] *Power Price Risk Hedging Opportunities in the Norwegian Market*, <https://www.statnett.no/globalassets/for-aktorer-i-kraftsystemet/utvikling-av-kraftsystemet/power-price-risk-hedging-opportunities-in-the-norwegian-market.pdf>, Accessed: 2022-06-16.
- [29] *Speculation - Learn how Speculation Affects Different Types of Markets*, <https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/speculation/>, Accessed: 2022-06-16.
- [30] *Commodities Speculators: More Help Than Harm?* <https://www.investopedia.com/articles/basics/09/the-function-of-speculators.asp>, Accessed: 2022-06-16.
- [31] *Electricity production - Energifakta Norge*, <https://energifaktanorge.no/en/norsk-energiforsyning/kraftproduksjon>, Accessed: 2022-06-01.
- [32] J. Dimoski, S. Nersten, S.-E. Fleten and N. Löhndorf, ‘Hydropower reservoir management using multi-factor price model and correlation between price and local inflow,’ in *IAEE International Conference, Groningen, Netherlands*, 2018.
- [33] *EOPS - one area power-market simulator*, <https://www.sintef.no/en/software/eops-one-area-power-market-simulator/>, Accessed: 2022-06-05.
- [34] A. Bringedal, A. Søvikhagen, E. Aasgård and S. Fleten, ‘Backtesting coordinated hydropower bidding using neural network forecasting,’ *Energy Systems*, 2021. DOI: <https://doi.org/10.1007/s12667-021-00490-4>.
- [35] A. Mæland, ‘Energy Management & Water Values,’ The Physical and Financial Power Markets - Workshop by Nord Pool Academy, Jan. 2022.

- [36] S. Raschka and V. Mirjalili, *Python Machine Learning, 3rd Ed.* 3rd ed. Birmingham, UK: Packt Publishing, 2019, ISBN: 978-1789955750.
- [37] F. Chollet, *Deep Learning with Python*, 1st. USA: Manning Publications Co., 2017, ISBN: 1617294438.
- [38] C. Deb, F. Zhang, J. Yang, S. E. Lee and K. W. Shah, ‘A review on time series forecasting techniques for building energy consumption,’ *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 902–924, 2017, ISSN: 1364-0321. DOI: <https://doi.org/10.1016/j.rser.2017.02.085>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1364032117303155>.
- [39] R. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, 2nd ed. Melbourne, Australia: OTexts, 2018, Accessed: 2022-06-20. [Online]. Available: [OTexts.com/fpp2](https://otexts.com/fpp2).
- [40] H. Bousqaoui, I. Slimani and S. Achchab, ‘Comparative analysis of short-term demand predicting models using arima and deep learning,’ *International Journal of Electrical and Computer Engineering*, vol. 11, no. 4, p. 3319, 2021.
- [41] H. Wang, R. Yao, L. Hou, J. Zhao and X. Zhao, ‘A methodology for calculating the contribution of exogenous variables to arimax predictions.,’ in *Canadian Conference on AI*, 2021.
- [42] *Machine learning vs deep learning: what’s the difference?* <https://itbrief.com.au/story/introduction-deep-learning>, Accessed: 2022-06-22.
- [43] F. Cordoni, ‘A comparison of modern deep neural network architectures for energy spot price forecasting,’ *Digital Finance*, vol. 2, no. 3, pp. 189–210, 2020.
- [44] O. B. Sezer, M. U. Gudelek and A. M. Ozbayoglu, ‘Financial time series forecasting with deep learning: A systematic literature review: 2005–2019,’ *Applied soft computing*, vol. 90, p. 106 181, 2020.
- [45] O. Tomic, *NMBU, Applied machine learning II, Lecture notes: DAT300 - 5 - Universal workflow of machine learning*, Nov. 2021.
- [46] *Activation functions in Neural Networks - GeeksforGeeks*, <https://www.geeksforgeeks.org/activation-functions-neural-networks/>, Accessed: 2022-06-26.
- [47] *Keras documentation: Losses*, <https://keras.io/api/losses/>, Accessed: 2022-06-26.
- [48] *Keras documentation: Metrics*, <https://keras.io/api/metrics/>, Accessed: 2022-06-26.

- [49] Z. Chang, Y. Zhang and W. Chen, ‘Electricity price prediction based on hybrid model of adam optimized lstm neural network and wavelet transform,’ *Energy*, vol. 187, p. 115 804, 2019.
- [50] *Time Series Forecasting Performance Measures With Python*, <https://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python/>, Accessed: 2022-06-26.
- [51] C. Szegedy, W. Liu, Y. Jia *et al.*, ‘Going deeper with convolutions,’ in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [52] *Keras documentation: The Functional API*, https://keras.io/guides/functional_api/, Accessed: 2022-06-28.
- [53] Q. Guo, S. Lei, Q. Ye and Z. Fang, ‘Mrc-lstm: A hybrid approach of multi-scale residual cnn and lstm to predict bitcoin price,’ in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2021, pp. 1–8.
- [54] *Time Series Forecasting with ARIMA | Kaggle*, <https://www.kaggle.com/code/redwankarimsony/time-series-forecasting-with-arma/notebook>, Accessed: 2022-07-03.
- [55] *Coding the SARIMA Model : Time Series Talk*, https://www.youtube.com/watch?v=A18m6K_stfA, Accessed: 2022-07-03.
- [56] *Grid Search VS Random Search VS Bayesian Optimization — by Aashish Nair — Towards Data Science*, <https://towardsdatascience.com/grid-search-vs-random-search-vs-bayesian-optimization-2e68f57c3c46>, Accessed: 2022-07-03.
- [57] *Keras documentation: HyperParameters*, https://keras.io/api/keras_tuner/hyperparameters/#hyperparameters-class, Accessed: 2022-07-03.
- [58] *Keras documentation: EarlyStopping*, https://keras.io/api/callbacks/early_stopping/, Accessed: 2022-07-03.
- [59] *Keras documentation: TensorBoard*, <https://keras.io/api/callbacks/tensorboard/>, Accessed: 2022-07-03.
- [60] *Keras documentation: Getting started with KerasTuner*, https://keras.io/guides/keras_tuner/getting_started/, Accessed: 2022-07-03.
- [61] *Cross Validation in Time Series. Cross Validation: — by Soumya Shrivastava — Medium*, <https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4>, Accessed: 2022-07-11.



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitenskapelige universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway