Norwegian School of Economics
Bergen, Spring, 2018

Artificial Intelligence and Nord Pool's Intraday Electricity Market Elbas: A Demonstration and Pragmatic Evaluation of Employing Deep Learning for Price Prediction

Using Extensive Market Data and Spatio-Temporal Weather Forecasts

Johannes Krokeide Kolberg & Kristin Waage

Supervisor: Walter Pohl

Master's thesis, Economics and Business Administration,
Finance

NORWEGIAN SCHOOL OF ECONOMICS

Acknowledgements

We would like to thank the Professors at NHH that contributed to this thesis: our supervisor Walter Pohl for the opportunity and support to work on such an ambitious project, Thore Johnsen for his invaluable input and guidance, Ivan Belik for his comments and support, and Roger Bivand for his technical pointers. We would also like to thank Nord Pool for access to their market data, as well as Morten Hegna and Sigfred Sørensen at Montel for access to SMHI's weather forecast data. Finally, we would like to thank Ståle Størdal at Eidsiva Energi and Olav Johan Botnen at Wattsight for generously sharing their industry expertise, as well as Lars Ove Skorpen and Bjørn Erik Heiberg at Pareto Securities for their input and for facilitating talks with market participants to improve the thesis' practical appeal.

Abstract

This thesis demonstrates the use of deep learning for automating hourly price forecasts in continuous intraday electricity markets, using various types of neural networks on comprehensive sequential market data and cutting-edge image processing networks on spatio-temporal weather forecast maps. Deep learning is a subfield of artificial intelligence that excels on problems such as these with multifarious input data and manifold interacting factors. It has seen tremendous success on a range of problems across industries, and while it is important to have realistic expectations, there is little reason to believe that intraday electricity markets are different. Focusing on Nord Pool's intraday market Elbas, we predict Nordic buyers' volume-weighted average price over the last six hours of trading prior to each delivery hour. Aggregating this window gives buyers flexibility from many trades and sufficient time in which to act on the predictions, and solves issues with data sparsity while keeping sufficient resolution for predictions to be informative.

We develop various neural networks via extensive experimentation, with inspiration from other research and problem domains. To make the findings relevant in practice, we impose constraints on the input data based on what would be available to Elbas market participants six trading hours ahead of delivery. The neural networks are benchmarked against a set of simple domain-based heuristics and traditional methods from econometrics and machine learning. We conclude with a holistic evaluation of the efficacy of deep learning on our problem, whether it is economically justifiable in light of its value-add, what the salient hurdles are to implementing it in practice, and what the implications are for broader applications of AI in intraday markets.

The deep learning models[1] are relatively accurate and reliable under normal market conditions. The average price across all delivery hours in the held-out data is 30.95 EUR/MWh, where our best network is on average off by 2.72 EUR/MWh. It beats the best simple heuristic by 21–25%, and the best benchmark model by 12–16%. The network also anticipates major fluctuations in prices relatively consistently, and generally outperforms all alternative methods when prices are especially volatile or trading activity particularly high. In contrast to the benchmarks, there are also ample avenues for improving the network further. Beyond being promising in its own right, we also argue that the network demonstrates the wider potential of deep learning in a range of applications in intraday markets, and that these are worthy of serious consideration — though one should be aware of the practical hurdles to implementing them operationally.

[1]The Python-code for training and evaluating our final neural networks is available on our public Github repository at: `https://github.com/johannes-kk/Elbas-Deep-Learning`.

Contents

Acknowledgements **i**

Abstract **ii**

List of Tables **vi**

List of Figures **vi**

List of Abbreviations **1**

1 Introduction **1**

1.1 Problem Definition 3

1.2 Literature Review 4

1.3 Thesis Structure 7

2 The Nordic Electricity Market **7**

2.1 Nord Pool 8

2.1.1 The Elspot Market 8

2.1.2 The Elbas Market 10

2.2 The Regulating Market 10

2.3 The Decision to Trade in Elbas 11

2.4 Nordic Power Production 13

3 Identification of Elbas Price Determinants **14**

3.1 Available Intraday Capacities 14

3.1.1 Production Capacities 14

3.1.2 Transmission Capacities 16

3.2 Magnitude of Power Imbalances 16

3.2.1 Intermittent Energy Production Forecast Errors 17

3.2.2 Load Forecast Errors 18

3.2.3 Unplanned Power Outages 19

3.3 Strategic Behaviour 19

3.3.1 Available Price Information 20

3.3.2 Market Power 21

3.4 Conclusion 22

4 Data **23**

4.1 Data Collection 23

4.1.1 Nord Pool Market Data 24

4.1.2 SMHI Weather Forecasts via Montel 25

4.2 Pre-Processing of Market Data 25

4.2.1 Calculating Aggregated Elbas Buyer Prices 28

4.3 Pre-Processing of Weather Data . . . 29
4.3.1 Matching with the Volume-Weighted Average Price . . . 31
4.4 Splitting into Training, Validation, and Test Sets . . . 32
4.5 Exploring the Data . . . 33
4.5.1 Characteristics of Elbas Trading . . . 34
4.5.2 The Elbas Volume-Weighted Average Price . . . 36

5 Methodology **38**
5.1 Model Evaluation . . . 39
5.2 Benchmark Models . . . 40
5.2.1 Simple Heuristics . . . 41
5.2.2 Econometrics and Simpler Machine Learning . . . 41
5.3 Deep Learning . . . 43
5.3.1 A Brief Introduction to Deep Learning . . . 44
5.3.2 Designing Network Architectures . . . 47
5.3.3 Evaluating the Final Architectures . . . 49
5.3.4 Windowed Market Models: Deep Residual Networks . . . 50
5.3.5 Sequential Market Models: LSTM Networks . . . 52
5.3.6 Processing Spatio-Temporal Weather Data . . . 53
5.3.7 Market & Weather Model: Multi-Input Network . . . 54

6 Analysis and Results **56**
6.1 Model Performance Results . . . 56
6.1.1 Benchmark Models . . . 56
6.1.2 Deep Learning Models . . . 59
6.2 In-Depth Analysis of Model Performance . . . 62
6.2.1 Performance across Delivery Hours . . . 62
6.2.2 Performance across Trading Activity . . . 64
6.2.3 Performance across Price Levels . . . 65
6.2.4 Detecting Unusually High or Low Price Levels . . . 66
6.3 Conclusion . . . 69

7 Discussion **70**
7.1 The Potential of Deep Learning . . . 70
7.2 Practical Barriers to Deep Learning . . . 73
7.3 The "Black Box" Critique . . . 74
7.4 Theoretical Limitations and Avenues for Further Research . . . 74

8 Conclusion **76**

References **78**

Appendix **85**

A Theory: Deep Learning **85**
A.1 The Multi-Layer Perceptron . . . 85
A.1.1 Activation Functions . . . 86
A.1.2 Gradient-Based Learning . . . 87
A.1.3 Optimisation Algorithms . . . 88
A.1.4 Fighting Overfitting . . . 88
A.1.5 Residual Connections . . . 89
A.1.6 Batch Normalisation . . . 90
A.1.7 Checkpoint Ensembling . . . 91
A.2 Recurrent Neural Networks . . . 92
A.2.1 Bidirectional Recurrent Networks . . . 93
A.2.2 Long Short-Term Memory . . . 93
A.3 Image Processing: Convolutions and Pooling . . . 94
A.3.1 Convolutional Pyramids . . . 95

B Additional Information about Elbas **96**
B.1 Prices in the Regulating Market . . . 96
B.2 Elbas Trade Volumes per Area and Country . . . 98
B.3 Trading Activity across Trade Hours . . . 98

C Standardisation or Normalisation? **99**
C.1 Distribution Plots . . . 99

D Deep Learning Models **101**
D.1 ResNet 1 . . . 101
D.2 ResNet 2 . . . 104
D.3 LSTM . . . 108
D.4 BD-LSTM . . . 110
D.5 Multi LSTM ResNet . . . 112
D.6 Weather Models . . . 115
D.7 Results LSTM vs. GRU . . . 118

E Theory on Benchmark Models **119**
E.1 Shrinkage Methods . . . 119
E.2 Feature Selection . . . 120
E.3 Feature Extraction . . . 121
E.4 Gradient Boosting . . . 121

F Market Data Variables **123**
F.1 Extracted Variables . . . 123
F.2 Summary Statistics . . . 124

List of Tables

1 Market and weather datasets 24
2 Summary statistics for the *near* Elbas volume-weighted price in the training, validation and test sets 37
3 MAE and RMSE of simple heuristics baselines 57
4 MAE and RMSE of econometrics and simpler machine learning baselines 58
5 Validation MAE and RMSE of the deep learning models 60
6 Test MAE and RMSE of the deep learning models 60

List of Figures

1 Bidding areas and transmission capacities 9
2 Decision tree for Elbas trading 12
3 The merit-order curve 13
4 The *far* and the *near* volume-weighted intraday price 29
5 Downsampled and cropped temperature forecast 30
6 Weather and price data merged 32
7 Number of Elbas trades per delivery hour 34
8 Prices and volumes per Elbas trade 35
9 Variation in Elbas trade price for the same hour of power delivery 36
10 Development in the *near* Elbas volume-weighted average price 37
11 The *near* Elbas volume-weighted price across delivery hours 38
12 *ResNet 1* network architecture 51
13 *ResNet 2* network architecture 51
14 *LSTM* network architecture 52
15 *BD-LSTM* network architecture 53
16 *Multi LSTM ResNet* network architecture 55
17 Variation in model performance depending on delivery hour 63
18 Variation in model performance depending on number of trades 64
19 Variation in model performance depending on delivery hour 65
20 *Multi LSTM ResNet* model predictions compared to true values 67
21 Model ability to detect price jumps 68

List of Abbreviations

Abbreviation	**Explanation**
AI	Artificial intelligence
BD	Bidirectional
BN	Batch normalisation
CE	Checkpoint ensembling
CET	Central European time
CNN	Convolutional neural network
CSV	Comma-separated values
Elbas	Electricity Balance Adjustment Service
EPEX	European Power Exchange
GBM	Gradient boosting model
GIME	German Intraday Market for Electricity
GPU	Graphical Processing Unit
GRIB	Gridded binary
GRU	Gated Recurrent Units
IQR	Interquartile range
LASSO	Least Absolute Shrinkage and Selection Operator
LSTM	Long-Short Term Memory
MAE	Mean absolute error
MARS	Multivariate adaptive regression splines
MIBEL	Iberian Electricity Market
ML	Machine learning
MLP	Multi-layer perceptron
MSE	Mean squared error
MVL	Minimum validation loss
MWh	Megawatt hour
RC	Residual connection
ReLU	Rectifying Linear Unit
ResNet	Residual network
RF	Random forest
RIE	Random initalisation ensembling
RMSE	Root mean squared error
RMSprop	Root mean square propagation
RNN	Recurrent neural network
SDV	Semicolon divided values
SGD	Stochastic gradient descent
TSO	Transmission system operator
UMM	Urgent market message
vRES	Variable renewable energy sources
VWP	Volume weighted average price
XML	Extensible Markup Language

1 Introduction

Electricity markets require constant balance between supply and demand, since the product can only be traded cross-border or stored in economic terms to limited extents (Hagemann, 2013; Weron, 2014). Deregulated exchanges provide multiple trading opportunities to efficiently match supply and demand, and play a crucial role in helping ensure stability in the power system (Scharff & Amelin, 2016). For physical power delivery, market participants usually have access to *day-ahead* and *intraday* trading.

The day-ahead markets are usually auction-based, where power is traded for each consecutive hour of the following day, and are the main arena for trading power. They have therefore been the primary focus of academic and business interests, as day-ahead forecasts are essential to participants in such markets. A range of approaches have been employed for this purpose, including multi-agent, statistical, and computational intelligence models (Weron, 2014). In contrast, the intraday markets vary in design from auctions to continuous markets, or a mix of both (Scharff & Amelin, 2016). They are primarily used to correct imbalances that arise after the closure of the day-ahead markets, and have until recently received far less attention. They are, however, gaining importance as the shares of intermittent power generation in the energy mix continue to increase; variable renewable energy sources (vRES) — such as wind and solar — are characterised by their dependence on contemporaneous weather conditions, and since intraday markets offer such power producers opportunities to adjust production imbalances closer to real-time, they are important to increasing the share of vRES in the overall energy mix (Henriot & Glachant, 2013; Pape, Hagemann, & Weber, 2016).

Still, the limited attention given to intraday electricity markets means many questions linger, like how prices in such markets are best modelled; intraday prices for power delivered at a given hour can vary significantly (Scharff & Amelin, 2016), and electricity prices are in general characterised by intermittently large fluctuations (Hellström, Lundgren, & Haishan, 2012). Better methods of predicting intraday prices may therefore reduce these uncertainties, and improve the decision-making and profitability of market participants. It is, however, a highly complex field with a myriad of potentially influential variables[2], and where the relationships between these variables are often non-linear (Hagfors, Bunn, Kristoffersen, Staver, & Westgaard, 2016).

Recent years have seen a rise in the use of *artificial intelligence* (AI) for addressing similarly

[2]See, for instance, Hagemann (2013) or Pape et al. (2016).

complex and non-linear prediction problems in a variety of domains, including day-ahead electricity pricing (Weron, 2014), and may presumably offer an avenue for tackling the complexity of intraday electricity markets as well. *Machine learning* (ML) is a subfield of AI where systems are *trained* rather than explicitly programmed; they learn to recognise statistical structures in the examples they are shown, in order to define rules to automate the given task (Chollet, 2018). Such techniques have become very popular in recent years due to a proliferation of computational power and big data, as ML excels on complex problems with comprehensive datasets where many traditional statistical methods would be impractical (Chollet, 2018).

However, the number of combinations of variable values increases exponentially the more variables there are, and many ML techniques struggle when the data has very high dimensionality (Chollet, 2018). Many such methods also presuppose, either directly or indirectly, the kind of function they should fit to tackle the given problem (Goodfellow, Bengio, & Courville, 2016). *Deep learning* is a subfield of ML, usually in the form of *neural networks*, which emphasises learning successive layers of increasingly complex and meaningful representations (Chollet, 2018). Such deep networks incorporate more general-purpose assumptions that the data-generating process is a hierarchical composition of factors (Goodfellow et al., 2016), and automate *feature engineering* — the manual refining of input data into successive representation spaces needed for *shallow* methods (Chollet, 2018) — that made previous ML methods somewhat esoteric.

Deep learning has seen impressive results on a range of AI problems, from self-driving cars to fraud detection, but media hype and lofty promises from practitioners with an axe to grind tend to inflate expectations of what is possible and what time frames are reasonable (Chollet, 2018). Research has made tremendous leaps in recent years, but little of this has been put to extensive use in practice. However, while there is significant untapped potential of deep learning across many industries (Chollet, 2018), we also caution against taking unrealistic claims at face value. The aim of this thesis is therefore to demonstrate the use of deep learning to automate hourly price forecasts in the Nordic intraday electricity market, Elbas, using specialised sequential networks on comprehensive market data and cutting-edge image processing on weather forecast maps. The results demonstrate the potential of using deep learning to both address problems that have been left untouched due to their complexity, and to automate hitherto laborious and repetitive processes. We also attempt to elucidate the practical challenges to implementing such AI agents, based both on our own experience with this thesis, and industry experience.

1.1 Problem Definition

Specifically, we focus on the Nordic electricity exchange, Nord Pool, and its continuous intraday market, Elbas. We take the perspective of a Nordic[3] market participant with an interest in *buying* power for a given delivery hour. As Elbas operates continuously, many trading opportunities may arise for each hour of power delivery. Knowing when or if specific trades occur is outside the scope of this thesis, hence we do not predict prices of individual trades. Instead, we predict Nordic buyers' *volume-weighted* average price over the last six hours of trading prior to each delivery hour. This window is a balance between giving the buyer sufficient time to act on the prediction, and the fact that intraday trading activity is usually concentrated in these final hours (Garnier & Madlener, 2015; Scharff & Amelin, 2016). This aggregation also mitigates some challenges with data sparsity, while keeping sufficient detail for predictions to be interesting.

This thesis is intended for both decision-makers in the Nordic electricity market that are interested in trading in Elbas — but do not necessarily have knowledge of deep learning — and for technically inclined readers without a deep understanding of electricity markets. To make this thesis accessible to a broader audience, we therefore aim to provide sufficient background both on Elbas and how deep learning works. While the focus is on building deep learning models that are as accurate and reliable as possible in their predictions, we also incorporate constraints and trade-offs that would exist in practice, so as to demonstrate the potential of deep learning under realistic conditions. Given this, we investigate the following research questions:

1. To what extent can deep learning predict the volume-weighted average Elbas price six hours ahead of a given hour of power delivery, and how reliable are the forecasts in practice?
2. What does this suggest about the potential of deep learning in wider applications in the Elbas market, and what are the salient hurdles to implementing such AI agents?

To our knowledge, this is the first study to apply deep learning to predicting hourly prices in Elbas. It therefore provides several contributions to existing research: it introduces techniques that exploit multifarious inputs — such as hundreds of market variables, and spatio-temporal weather forecast images; it evaluates the benefits and practical challenges of using deep learning for price prediction in Elbas; the techniques presented may be adapted to specific bidding areas or extended for other purposes; finally, deep learning enables considerable automation in Elbas, where much of the trading is currently done manually (Scharff & Amelin, 2016).

[3]Sweden, Finland, Denmark, and Norway.

1.2 Literature Review

There are, to the best of our knowledge, few studies with the aim of predicting prices in Elbas. Botnen Holm (2017) uses multiple regression to model the Elbas price in her thesis, while Tangerås and Mauritzen (2014) construct a regression model to examine the difference between the Elspot and Elbas prices. The German intraday market (GIME) resembles Elbas in that it allows for continuous trading of electricity. For the GIME, Pape et al. (2016) attempt to model the hourly price by combining a fundamental supply-stack model with a linear regression model, while Kiesel and Paraschiv (2017) develop an asymmetric econometric model that accounts for two different market regimes. Additionally, Wolff and Feuerriegel (2017) and Hagemann (2013) apply multiple regression to model the intraday price and the difference between the day-ahead and intraday price, respectively.

Although the aforementioned studies present ways of modelling intraday prices, they incorporate one or more input variables that would not be available to Elbas market participants ahead of gate closure for a certain delivery hour. As intraday markets also differ regarding design and regulation, models developed for the GIME, albeit also continuous, may not simply be applied to Elbas.[4] In large part as it was not the goal of these studies, most existing models can therefore not be used in practice to obtain actionable Elbas price predictions.

Even though few studies explicitly model the Elbas price, price estimates are an important component in specialised multi-stage optimisation models, of which there are a few. In these models, price estimates must be available ahead of time, allowing the market participant to act so as to optimise buy/sell decisions. Using the mean, maximum and minimum Elbas prices six hours ahead of delivery, Bourry and Kariniotakis (2009) model the price as a triangular distribution, from which they obtain an estimate of the proportion of accepted bids, in their optimisation framework for a wind producer. Fodstad, Aarlott, and Midthun (2018) design an optimisation model for a hydropower plant, and use historical hourly average Elbas prices to evaluate the potential gains from multi-stage trading. In this context, they mention price estimates as one of the core challenges related to market modelling parameters. Engmark and Sandven (2017) incorporate estimates of Elbas premiums, for the Kristiansand area in Norway, by using historical order depth to construct a demand curve that is then used to derive the Elbas premium for a given volume.

[4]For example, gate closure in the GIME is later than in Elbas (Kiesel & Paraschiv, 2017), allowing for the inclusion of historic variables that would not be possible in Elbas.

It appears that most optimisation models estimate Elbas prices based on historical price/bid data, combined with probabilities. As such, one may expect them to pick up trends that are unique for a specific delivery day or delivery hour only to a limited extent. We seek to contribute to current research by constructing a model that can provide forecasts using real-time information specific for the delivery hour in question. However, the decisions on *how* to optimise trading to correct for imbalances are outside the scope of this thesis, especially as there already exist multiple such frameworks.

Various price models have also been developed for other intraday markets. AI has seen its application in intraday price forecasting in the Iberian Electricity Market (MIBEL), where intraday trading is organised as six auction session.[5] Monteiro, Ramirez-Rosado, Fernandez-Jimenez, and Conde (2016) build a neural network implemented with a multi-layer perceptron (MLP)[6] to forecast the prices in each of these six intraday sessions. They find that prices can be forecasted with relatively high accuracy by only including hour-of-day and day-of-week dummies, and saw little improvement from including additional variables. Price forecasting for the MIBEL intraday sessions is also explored by Andrade, Filipe, Reis, and Bessa (2017) who utilise probabilistic forecasting techniques. They agree with Monteiro et al. in that simpler models that primarily incorporate price information from previous sessions yield the best results.

However, the discrete intraday auction prices in MIBEL are highly correlated, and Andrade et al. (2017) point out that once there is a price jump between sessions, their preferred model is unable to pick it up. Furthermore, MIBEL intraday trading is essentially designed as auction-based day-ahead markets, except with later gate closure times (Andrade et al., 2017). Hence, a clear price signal is frequently established in the markets. Continuous intraday markets, such as Elbas and GIME, differ in this regard, as prices are set on a per-trade basis at any point in time within the markets' trading hours. Existing literature on the continuous intraday markets suggests that, in addition to prior market prices, a variety of factors may be relevant when modelling hourly prices. These factors may in turn provide information that enable models to capture intraday price jumps — which prior price information alone would not. For example, Pape et al. (2016) conclude that ramping costs and supply scarcity have significant influences in their model, in addition to previous price information, Wolff and Feuerriegel (2017) find significant effects of load and wind/solar in-feed levels, and Hagemann (2013) observes how unexpected outages, in

[5]These sessions are run day-ahead at 17:00–18:45 and 01:00–21:45, and intraday at 01:00–01:45, 04:00–04:45, 08:00–08:45, and 12:00–12:45.

[6]Appendix A.1 provides an introduction to MLPs.

addition to wind, solar and load forecast errors, contribute to explaining the difference between hourly day-ahead and intraday prices. Studies focusing on the effects of various intraday price drivers have been important in identifying which variables to include in models. However, they are not summarised here, as Chapter 3 provides a thorough discussion of how the Elbas price may be affected by their findings.

Since a range of factors are found to potentially influence continuous intraday market activity and prices, we therefore believe deep learning techniques that can leverage manifold input variables are very interesting to try. Applications of deep learning have been found to outperform more traditional methods within a variety of fields. Specifically, the use of Recurrent Neural Networks (RNN)[7] with Long Short-Term Memory (LSTM)[8] have yielded promising results in predicting the day-ahead electricity price (Peng, Liu, Liu, & Wang, 2018). LSTM is also applied to weather forecasting, where both S. Srivastava and Lessmann (2018) and Alzahrani, Shamsi, Dagli, and Ferdowsi (2017) find that LSTM outperforms other methods in forecasting solar energy, while Liu, Mi, and Li (2018b) and Liu, Mi, and Li (2018a) successfully integrate LSTM in a multi-step model for wind speed forecasting. For stock market predictions, LSTM has been found to outperform memory-free deep learning models in predicting directional movements of constituent stocks (Fischer & Krauss, 2018), and a hybrid model integrating LSTM yields promising results in predicting stock price volatility (Kim & Won, 2018).

The performance of deep learning models, including LSTM, on predicting day-ahead electricity prices is also studied by Lago, De Ridder, and De Schutter (2018), who find that their models have a better predictive accuracy than the simpler benchmark models in the Belgian market. Furthermore, they conclude that their MLP model outperforms the other deep learning models overall, but that their LSTM model achieves higher predictive accuracy during certain delivery hours. An MLP model with multiple hidden layers is also the preferred method in Lago, De Ridder, Vrancx, and De Schutter (2018) when analysing how incorporating market integration improves the accuracy of day-ahead price forecasting in the French and Belgian markets. Furthermore, Y. Chen, He, and Tso (2017) attempt to predict crude oil prices using an LSTM model, but it is also here outperformed by other deep learning techniques. The authors suggest that this may be due to the crude oil market being mainly characterised by shorter-term behaviour that is not captured properly by the LSTM model. As both MLP and LSTM models

[7]These networks are specialised to process sequential input data. Appendix A.2 provides more details on recurrent neural networks.

[8]LSTM is a type of recurrent neural networks specialised for learning long-term dependencies. Appendix D.3 provides more details on LSTM.

seem promising, but neither is clearly superior on the basis of existing literature, we will employ both in order to thoroughly evaluate the potential of deep learning in predicting hourly prices in Elbas.

1.3 Thesis Structure

Chapter 2 gives a brief introduction to the Nordic electricity market. Chapter 3 identifies factors that may affect the Elbas price, to motivate the selection of input data. Chapter 4 explains how our data were collected and pre-processed, and provides an exploratory analysis of Elbas trading characteristics and the volume-weighted price. Chapter 5 describes the methodology used to develop and evaluate our models, as well as detailed overviews of the final model designs. Appendix A includes a self-contained technical introduction to how various neural networks function, as well as more detailed information on the specific methods we employ. Chapter 6 presents and evaluates the performance of our final models, and several benchmarks for comparison. Chapter 7 discusses the implications for the wider potential of other applications of deep learning in Elbas, as well as the practical hurdles to implementing such solutions. The chapter also outlines the main theoretical limitations and suggests improvements for future work. Chapter 8 concludes this thesis.

2 The Nordic Electricity Market

This chapter serves as an introduction to the features of the Nordic electricity market that are of relevance for this thesis. It begins by introducing Nord Pool and its platforms for day-ahead and intraday trading, Elspot and Elbas, in Section 2.1. Then, Section 2.2 describes the regulating power market, which is used to secure the electrical stability of the system, and where participants may incur costs of they exacerbate imbalances. Following the descriptions of the three physical power markets, Section 2.3 elucidates the mechanisms that affect the decision of whether to trade in Elbas. Finally, Section 2.4 provides an overview of power production types and the supply curve in the Nordic countries, as production costs may impact the Elbas prices.

2.1 Nord Pool

In the Nordic countries, market participants are offered day-ahead trading in the Elspot market, and intraday trading in the Elbas, or *Electricity Balance Adjustment Service*, market. Elspot and Elbas are both run by Nord Pool, which is owned by the Nordic and Baltic *transmission system operators* (TSOs) (Nord Pool, 2016).[9] The TSOs are neutral and independent of the market members (Nord Pool, n.d.).

In addition to the Nordic countries, Nord Pool consists of the Baltic countries for day-ahead trading, and per January 2018 also Germany[10], the Netherlands and Belgium for intraday trading.[11] Due to transmission capacity constraints, there may be several bidding areas, also called price zones, within a country (Nord Pool, 2018a). Figure 1 shows these bidding areas. As this thesis focuses on Elbas buyers located in the Nordic countries, the relevant bidding areas are the Norwegian areas (NO1–NO5), the Swedish areas (SE1–SE4), the Danish areas (DK1–DK2), and the Finnish area (FI). In addition, we include Elbas trades with a seller from a German bidding area, since they make up a relatively large share of the electricity sold to the Nordics (Figure 24a, Appendix B.2). The cable connections between the areas are marked with red arrows in the figure. Maximum cable capacities restrict the flow of power that may be transferred from one bidding area to another, hence if settled Elspot trades take up capacity on one line, the capacity that remains for Elbas trading is affected.

2.1.1 The Elspot Market

In Elspot, electricity contracts are settled for power delivered the next day. For each hour, sellers estimate how much power they can deliver in the specific hour and at what price they are willing to deliver that volume, while buyers estimate the power they will need to meet the demand in the specific hour and how much they are willing to pay. Orders for power delivery/purchase over the next day are entered to the day-ahead trading system, with the deadline for submitting orders at 12:00 CET. Nord Pool then sets the hourly price so that balance is expected between demand and supply in each of the following day's 24 delivery hours (Nord Pool, 2018b).

[9]The TSOs in the Nordics are: Statnett SF (Norway), Svenska kraftnät (Sweden), Fingrid (Finland), and Energinet (Denmark). The TSOs in the Baltics are: Elering (Estonia), Litgrid (Lithuania) and Augstsprieguma tikls (Latvia).

[10]Elbas is an alternative intraday market for Germany (Scharff & Amelin, 2016), as Germany also is part of the EPEX Spot intraday market.

[11]As part of the cross-border intraday market project (XBID), Nord Pool also offers intraday trading in France and Austria, starting from June 2018 (Nord Pool, 2018g).

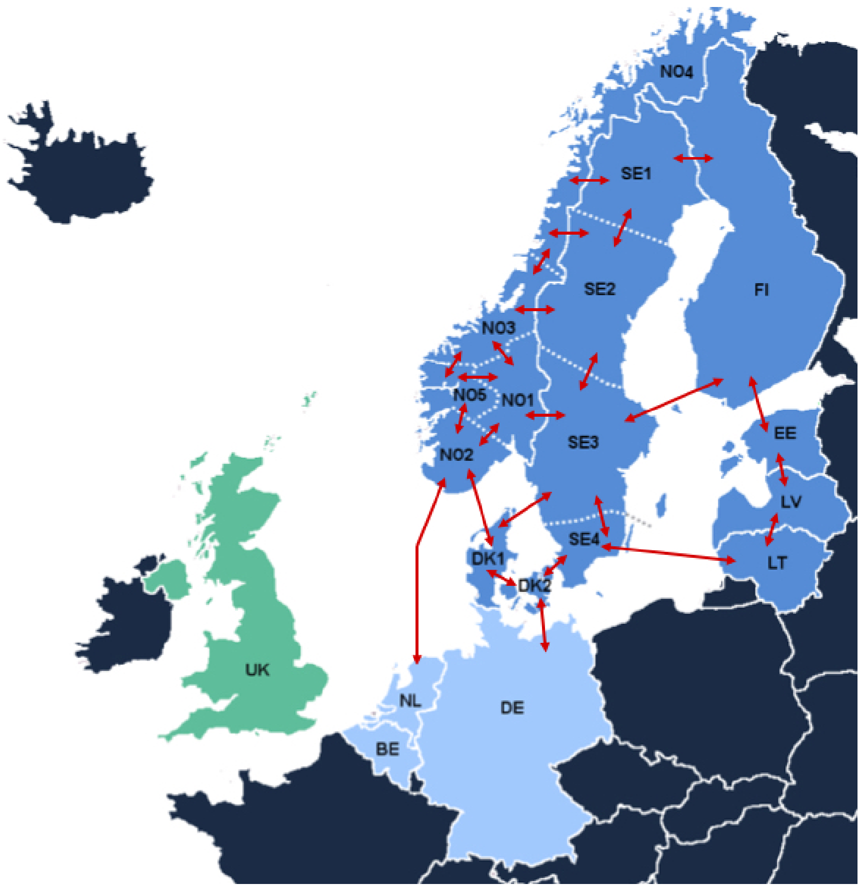


Figure 1: Bidding areas and potential transmission capacities in Elspot (dark blue areas) and Elbas (dark blue plus light blue areas) per January 2018. In addition, Nord Pool offers day-ahead and intraday trading locally in the UK (green), through the N2EX platform. The red arrows indicate the cable connections between the bidding areas. The illustration is based on Scharff and Amelin (2016), Nord Pool (2018c), and Statnett (2018).

The Elspot price that balances overall supply and demand is called the *system price*, and reflects equilibrium in a market with no transmission grid bottlenecks (Nord Pool, 2018a). This theoretical price is used as a reference for trading in the financial electricity markets. In reality, there are constraints on how much power can be transmitted between areas. As a result, the Nord Pool market is divided into the bidding areas shown in Figure 1, with individual *spot prices* calculated for each. This ensures equilibrium between supply and demand in each area. If there exists available transmission capacity, electricity may be transferred from an area with surplus power to an area with a power deficit. This type of congestion management in Elspot may cause area prices to converge. Still, as long as transmission constraints exist, the bidding areas may experience spot prices different from the system price, and from the spot prices in other areas.

2.1.2 The Elbas Market

For each market participant, settled Elspot trades constitute their binding production or consumption plan, which is submitted to the TSOs (Scharff & Amelin, 2016). The participants are committed to supply or consume corresponding to the plan, which may be updated until 45 minutes before the start of the delivery hour to incorporate deviations from the participants' day-ahead commitments. Here, the Elbas market complements the day-ahead market, Elspot, by providing market participants an opportunity to adjust any imbalances closer to real time.

Elbas is a continuous market, where prices are set based on a pay-as-bid basis for all transactions. In that way, Elbas resembles a regular stock market, except participants can trade in Elbas 24 hours a day, 365 days a year (Nord Pool, 2018d). For a given day of power delivery, Elbas opens at 14:00 CET, two hours after the gate closure of the day-ahead market, at which point participants know which bids were accepted and what volume they are expected to supply or buy in the day-ahead market. The gate closure in Elbas is 60 minutes before power delivery.

Several types of orders may be traded in Elbas, providing flexibility for market participants. Orders can be defined for 15 minute, 30 minute and 60 minute power products, in addition to block and iceberg orders (Nord Pool, 2018e). Block orders are specified by the user, and they may contain between one and 24 consecutive hourly products. Limit orders are buy or sell orders, and they are executed at the limit price or higher (lower) for buy (sell) orders. Furthermore, for block orders only the entire user-defined volume may be executed, while for limit orders the volume may be partially or entirely executed.[12]

2.2 The Regulating Market

Any imbalances that arise after the Elspot auction closes at 12:00 CET the day ahead may also be dealt with through the regulating, or *balancing*, power market. This market is run by the TSOs, and its purpose is to secure a constant balance between supply and demand in the hour of delivery. Producers with flexible power generation may submit bids to the regulating market, which are activated if necessary to ensure this balance. It opens for bid submission at 13:00

[12]Elbas also offers some specific types of limit orders. Iceberg orders (IBO) divide the full size of the order into smaller clips, where each clip only becomes visible to the market when the previous clips have been entirely executed. IBOs are typically used for large volumes so as to conceal the full order size from the market. Fill-or-Kill (FoK) orders are another type of limit orders that require the entire volume of the order to be matched immediately upon submission, or the order is withdrawn from the market. Finally, Immediate-or-Cancel (IoC) orders resemble FoK orders with the exception that as much as possible of the volume is matched immediately upon submission, and then the remaining volume is, like FoK orders, withdrawn from the market.

CET, which is just after day-ahead results are published and one hour before Elbas trading begins, and it closes 45 minutes before the hour of delivery (Pires Ferreira, 2016).

Imbalances are handled in the regulating market by computing an imbalance cost that participants must pay if they deviate from their binding plans (Scharff & Amelin, 2016). In Elbas, imbalances are corrected by trading *before* the delivery hour arrives, after which point participants pay their eventual imbalance cost in the regulating market. This cost reflects the extra price that has to be paid to ensure system balance. Appendix B.1 further explains how this price is derived and thereby how the imbalance costs are calculated. Basically, an *up-regulating* price is set above the day-ahead price if more power must be produced, as it incentivises additional production. Conversely, a *down-regulating* price, set below the day-ahead price, is used if production must be reduced, as it incentivises producers with higher marginal costs to purchase power from other producers rather than produce themselves. The TSOs set the imbalance costs on a per-area basis, using the same areas as in the Elspot and Elbas market (Figure 1).

2.3 The Decision to Trade in Elbas

Section 2.1 and Section 2.2 presented the three markets for physical power delivery. However, the question of why a market participant may decide to trade in Elbas deserves more attention, as it is linked to potential Elbas price drivers. To begin with, we can categorise Elbas market participants into two groups:

1. Participants who must correct an imbalance in their binding production/consumption plan.
2. Participants who can profit from buying/selling in Elbas and reallocate their own production/consumption, but are not in a situation where balancing is necessary.

Both participant types may be either a power consumer or a producer.

The options that are available for a market participant who experiences deviations, either a surplus or a shortfall, to the volumes settled in Elspot, are marked in blue in Figure 2, which is based on an equivalent illustration in Mauritzen (2013). To sell/buy power in Elbas is one option; leaving the imbalance to the regulating market as explained in Section 2.2 is another. If possible, participants may also utilise internal balancing, such as rescheduling other power plants within their portfolio to compensate for deviations (Holttinen & Koreneff, 2012; Scharff, 2012). Hence, not all market participants will consider Elbas as their optimal strategy of improving imbalances. The main benefit of Elbas trading for participants that experience imbalances, is the possibility to reduce the imbalance costs that they may otherwise incur in the regulating market (Hagemann, 2013; Mauritzen, 2013; Scharff & Amelin, 2016). In other words, trading in Elbas may be motivated by expectations of it being the least expensive alternative for balancing.

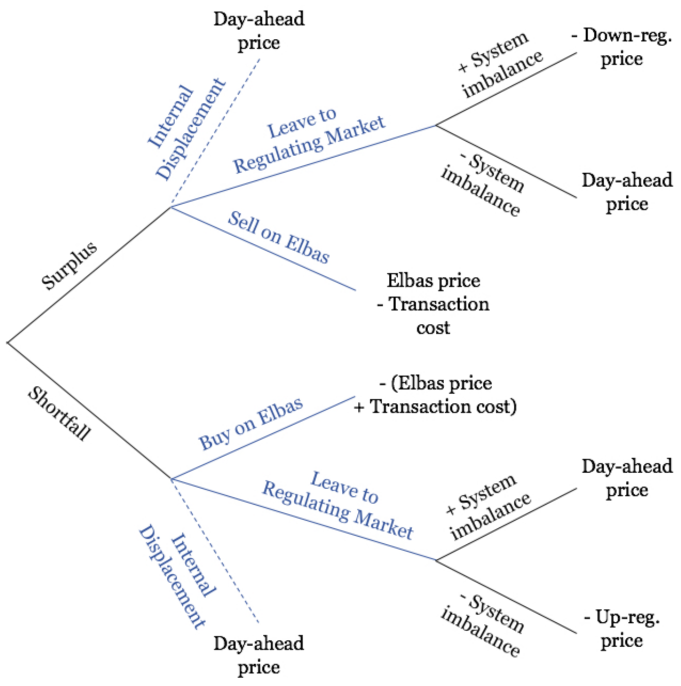


Figure 2: Decision tree for Elbas trading. The illustration is based on Mauritzen (2013).

However, participants without imbalances — the other type of Elbas participants — may also find it attractive to trade in Elbas. Producers with flexible power generation, such as hydro and thermal, have the possibility to offer this unused flexibility to buyers who are willing to pay a price that exceeds the marginal cost of production (Scharff & Amelin, 2016). Thus, participants may be able to make a profit by selling power in Elbas that was not profitable to trade in the day-ahead market. Consumers with flexible consumption may also sell extra power if they find buyers willing to pay a price that covers the costs of rescheduling their own consumption (Scharff & Amelin, 2016). Alternatively, these participants may offer the excess power to the regulating market, but they will not know beforehand whether their bids will be accepted or not. Finally, Elbas may be used by participants to optimise their committed production/consumption plans — for example, producers may find it less costly to buy a given volume of electricity from an Elbas seller than to produce the same volume by generating their own (Scharff & Amelin, 2016).

2.4 Nordic Power Production

Although electricity is a homogeneous good, power production can vary considerably in terms of production method, and hence, production costs. In total, about 400 TWh of electricity is generated annually in the Nordics. Hydro makes up more than half of this production, and almost all Norwegian generation comes from hydro power plants (International Energy Agency, 2016d). In Sweden and Finland, nuclear power production is also a major energy source — providing around 35–41% of domestic generation (International Energy Agency, 2016b, 2016e). Wind constitutes the largest share of Danish electricity generation (International Energy Agency, 2016a), and is the third largest energy source in the Nordics as a whole with a share of 8%. Hence, wind is the most important vRES in the Nordics. In Germany, solar is also an important vRES in addition to wind, and in total they constitute 18% percent of German generation (International Energy Agency, 2016c).

The types of power production determine in turn the shape of the merit-order curve, which represents supply in an electricity market. This curve plays an important role for price setting, as it represents the marginal costs of power production (Hagemann, 2013; Hagemann & Weber, 2013). In the merit-order curve, all energy sources are arranged in increasing order of marginal cost as shown in Figure 3. Renewable energy sources, such as hydro and wind, have close to zero marginal cost. The costs then increase depending on energy source, with coal, oil and gas being the most expensive. As Figure 3 illustrates, the annual Nordic consumption usually lies in a volume range where coal is the most expensive energy source.

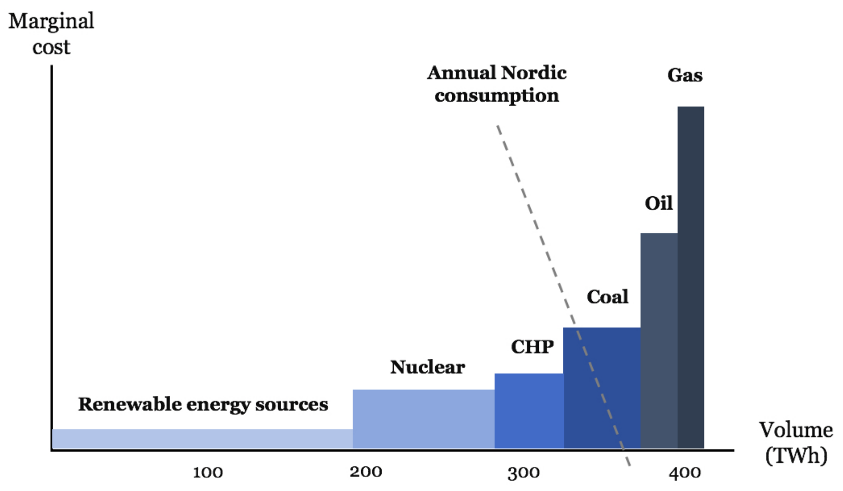


Figure 3: The merit-order curve shows how marginal cost of production varies with energy type. CHP is *combined heating and power production.* The illustration is based on Nord Pool (2018f).

3 Identification of Elbas Price Determinants

This chapter provides a synopsis of factors that may affect the Elbas price. Its objective is to better understand the dynamics of the Elbas market and to establish relevant variables to include in the models. It is also of importance in identifying variables that potentially influence the price, but are omitted from the models due to lack of data. Section 3.1 examines how available production and transmission capacities in the intraday market may affect the Elbas price, before Section 3.2 considers the types of imbalances that may occur, and how the magnitude of these can impact the price. Then, Section 3.3 evaluates how participants' strategic behaviour may lead to prices that deviate from what would be expected if only physical market factors had an effect. All sections also evaluate to what extent we are able to take the given factor into account in the models. Finally, Section 3.4 provides concluding thoughts on the various price determinants and their interplay.

3.1 Available Intraday Capacities

In general, the closer the market works to its capacity constraints, the higher the intraday prices may be (Hagemann, 2013; Hellström et al., 2012). This section outlines how available capacities and their effect on the price may be captured in the models. Section 3.1.1 evaluates the influence of available production capacities, before Section 3.1.2 considers capacities related to the transmission grid.

3.1.1 Production Capacities

Available intraday production capacities are the ones not committed in the day-ahead market, and they may impact on intraday prices as the marginal cost of production is dependent on the type of energy source available (Section 2.4). If there is a power deficit, intraday demand for power will increase. The cost of the next marginal plant, which must be activated to meet this additional demand, will be higher than the day-ahead price.[13] Consequently, the intraday price must exceed the day-ahead price so as to incentivise additional production beyond day-ahead volumes, ceteris paribus. Conversely, excess intraday supply may have to be offered at prices below the day-ahead price for producers to have an incentive to buy power to fulfil their day-ahead commitments instead of generating their own — and in that way, absorb intraday

[13]The cheaper energy sources are already committed in the day-ahead market.

supply surpluses. It is worth mentioning that the presence of ramping costs — that is, the costs associated with ramping up[14] or down[15] — a power plant, may require intraday prices to be even higher (lower) than marginal costs of production for additional (less) production to occur (Hagemann, 2013; Pape et al., 2016). As it represents the marginal cost of the last committed day-ahead production unit,[16] the day-ahead equilibrium between demand and supply (the merit-order curve) for a given delivery hour can shed light on how the intraday price reacts to intraday power imbalances (Hagemann, 2013). This equilibrium can be represented in the models through data on Elspot settlements for all Nordic bidding areas, namely *Elspot price and buy/sell data*.

As the merit-order curve for the Nordic electricity market is steeply increasing for higher production volumes, the higher the volume settled in the day-ahead market, the higher the intraday price must be to stimulate additional production (Hagemann, 2013). Furthermore, it is not uncommon for smaller companies to choose to close their intraday positions during office hours instead of employing a shift-team during out-of-office hours (Hagemann & Weber, 2013), which seems to also be the case in Elbas where the majority of trades are settled during office hours (Scharff & Amelin, 2016). In periods with less available production capacities, due to fewer participants trading in the market, intraday demand and supply curves are steeper (Hagemann, 2013), which may lead to more extreme price impacts if imbalances occur. Hence, in addition to Elspot data, incorporating *hourly and daily dummies* for power delivery time may allow the models to recognise delivery periods for which intraday trading activity is systematically lower or higher.

When it comes to available production capacity, data explicitly representing the types of energy sources committed in the day-ahead market may also be of relevance. In particular, hydro production constitutes a considerable share of Nordic power production (Section 2.4), its marginal cost of production is close to zero (Tangerås & Mauritzen, 2014),[17] and hydro units can be rescheduled nearer gate closure than e.g. thermal power plants (Scharff & Amelin, 2016). As such, periods with relatively high levels of hydro reservoir content may contribute to restraining

[14]Up-ramping costs cover costs such as depreciation, additional maintenance, and loss of life expectancy for machinery, in addition to balancing and fuel costs necessary to reach scheduled output (Hagemann, 2013).

[15]Down-ramping costs include reduced lifetime of machinery parts exposed to high pressure and heat, increased inspection and repairs costs, and the opportunity cost of not being able to start up fast enough to deliver in future hours with potentially higher prices (Hagemann, 2013).

[16]Or alternatively, the marginal cost of the first available intraday production unit, all else being equal.

[17]The main cost in hydro production is the opportunity cost — the so-called *water value* — which reflects the expectations about the future value of hydro resources (Tangerås & Mauritzen, 2014). Hence, hydro power may be traded in the intraday market if the intraday price exceeds the water value.

intraday prices from accelerating, while periods with lower levels may be more exposed to price jumps (Knapik, 2017). Including data on *hydro reservoir content and inflow* in the models may capture these dynamics. However, hydro data are only available with a weekly resolution, while the Elbas price is predicted on an hourly level. Though potentially important to the price, hydro data are therefore not included in our dataset.

3.1.2 Transmission Capacities

Just as transmission capacities may result in different spot prices for different bidding areas (Section 2.1.1), transmission capacities may also affect intraday prices. Scharff and Amelin (2016) look at trading within and between areas, and observe that Elbas is, to a large extent, used for cross-border trading — which requires available line capacity. However, they find that several lines were unavailable for intraday trading for at least a third of all hours in their dataset. In particular, one may expect lower system-wide prices when there is abundant transmission capacity, while periods with less intraday line capacity may result in higher prices overall.[18]

Data on available *Elspot and Elbas transmission capacities* between bidding areas may capture their potential intraday price impact, and is therefore included.[19] We do not, however, include *flow data* — that is, continuously updated scheduled intraday capacity; ex-post, we only have final numbers available and cannot deduce what the scheduled intraday flow was at the time of prediction. Instead, we include *volumes for settled Elbas trades* — if any — for each given delivery hour. Though these data do not provide information on which lines are occupied by the settled trades, they may partly capture increased flow in the transmission grid as a whole.

3.2 Magnitude of Power Imbalances

With a better understanding of how available intraday capacities may affect the degree to which intraday imbalances have an impact on the price, we now move on to considering why imbalances may arise in the first place. In the literature, two types of imbalances in particular are mentioned in relation to developments in intraday trading and price, namely forecast errors and unforeseen

[18]Available transmission capacities also impact to what extent the Elbas market is integrated across the Nordic bidding areas (Fridolfsson & Tangerås, 2009), and therefore whether one system-wide price may be representative for each respective area.

[19]By including both Elspot and Elbas capacities, we can get additional information about the dynamics in the market, and the models may be able to learn additional patterns. If Elbas capacities e.g. are low, we would not be able to tell if this is due to capacity being taken up in the day-ahead market or that there is a malfunction on the line so that capacity is unavailable both in Elspot and Elbas, unless we also include the Elspot capacities.

power outages. Forecast errors can in turn be separated between those related to intermittent, or vRES, energy production and those related to the electricity load.

In general, negative forecast errors or unplanned power outages can be expected to result in increased intraday demand due to power shortages, and may as such drive up prices, ceteris paribus. Positive forecast errors, on the other hand, are expected to result in lower intraday prices due to excess production, which indirectly incentivises producers to substitute their own production with power bought in the intraday market. The rest of this section explains how we may account for vRES forecast errors (Section 3.2.1), load forecast errors (Section 3.2.2), and power outages (Section 3.2.3) in our models. However, it should be noted that not all participants would balance deviations in the intraday market; if possible, power plants may first internally match opposing deviations in their portfolio or ramp up flexible plants, before trading the net difference in the market (Hagemann, 2013; Hagemann & Weber, 2013). The total impact of imbalances on intraday price and trading may therefore be lower than expected by only looking at the magnitude of deviations.

3.2.1 Intermittent Energy Production Forecast Errors

Intermittent energy generation, such as wind and solar, is strongly dependent on real-time weather conditions. Changes in weather forecasts can therefore cause surpluses or deviations from planned and committed day-ahead vRES generation levels that are reflected in intraday price movements. The inclusion of *data on weather forecasts* may capture this effect of vRES forecast errors on the price in our models. In the Elspot market, particularly wind speed is relevant when modelling the system price (Mosquera-López, Uribe, & Manotas-Duque, 2017), and for Elbas, wind forecast errors are found to correlate with the probability of trading (Mauritzen, 2013). This is not surprising, as wind energy makes up a relatively large share of the Nordic production, while solar energy only constitutes a minor share (International Energy Agency, 2016a, 2016b, 2016d, 2016e). In the GIME, both the level of wind and solar energy, and forecast errors with respect to these energy sources, seem to have a significant influence on the price (Hagemann, 2013; Wolff & Feuerriegel, 2017). As Germany is integrated in the Elbas market, solar energy may also impact on the Elbas price.

As we make the prediction six hours in advance, the latest available weather forecasts at that time should, ideally, be compared to those that were available before gate closure in the day-ahead market. However, technical limitations[20] only allow us to use one weather forecast per delivery hour. Hence, we have chosen the most recent. As we also include Elspot settlement data, the information loss by not including double forecasts per delivery hour may be mitigated.[21] To further compensate for the lack of minimum two forecasts, we also incorporate *production prognoses* for each bidding area, as some bidding areas — such as the DK1 and DK2 areas (Mauritzen, 2013) — may be characterised by producing more wind power than others.[22] Still, it is important to be aware that using only one weather forecast per delivery hour does not directly capture forecast *errors*.

Production may also depend on seasonal effects (Wolff & Feuerriegel, 2017), in that solar energy may only be produced during hours with sun and that certain months experience more sun or wind than others. Including *monthly* — in addition to the aforementioned *daily and hourly* — dummy variables may enable the models to account for seasonal production patterns.

3.2.2 Load Forecast Errors

It is not only the intraday supply side that may be affected by weather forecast errors; electricity load may also shift as a result of changes in weather. In particular, temperature is found to be an important price driver in the Nordic electricity market due to its influence on electricity load (Hellström et al., 2012; Knapik, 2017; Mosquera-López et al., 2017). Furthermore, stronger wind than predicted day-ahead can mean that more people stay inside and use appliances, resulting in higher electricity consumption than would be expected (Mauritzen, 2013). The same applies to heavier rainfall or less sunshine than forecasted day-ahead.[23]

The load forecast error in Elbas is difficult to estimate. *Actual consumption data* are published continuously. However, as the data are revised and final numbers are not available until days later, we have excluded these data from our models. *Consumption prognoses* for each bidding

[20]Section 4.3 provides more details on the technical limitations.

[21]For example, a relatively high day-ahead price, without production being unusually high, can indicate that renewable energy sources constitute a low share of the energy mix in that hour. If the weather forecast in question then shows relatively high levels of wind, the model may be able to derive that a positive forecast error has occurred.

[22]As such, if production for example is low in these bidding areas, while the most recent forecasts show strong winds for the delivery hour in question, it may indicate a positive forecast error.

[23]To be clear, people would not stay inside *because* the weather forecasts were wrong, as some impotent form of puerile pouting over the sudden realisation of day-ahead markets' mistake. Rather, weather different than day-ahead markets expect can lead to other consumption or production levels than were incorporated in the pricing.

area are available day-ahead, and including these may partially mitigate the problem.[24] In this regard, it is also relevant to take seasonal effects into account. Consumption is systematically higher during day hours than night hours (Hagemann, 2013; Hagemann & Weber, 2013), and during weekdays than weekends (Hagemann & Weber, 2013; Tangerås & Mauritzen, 2014). The aforementioned *seasonal dummies* may allow for such patterns to be accounted for.

3.2.3 Unplanned Power Outages

Production failures, in particular for coal and hydro, have been found to increase the Elbas price compared to the Elspot price (Tangerås & Mauritzen, 2014), and depending on the size of the outages, they may help explain price jumps in Elbas. This may especially be the case during weekdays when less capacity is available than during weekends (Tangerås & Mauritzen, 2014). Nord Pool has a reporting system called *Urgent Market Messages* (UMMs), where information about power outages is published. The effect of unplanned outages may be included in price models through extracting relevant information about outage size and duration from these UMMs. Only outages that happen after the closure of the day-ahead market may lead to purchases in the intraday market (Hagemann & Weber, 2013; Tangerås & Mauritzen, 2014).

In addition to day-of-the-week effects, the influence of unplanned outages on intraday prices may also vary depending on the hour of the day. As the electricity demand is lower at night, a share of the power plants that default may not be operating anyway as their marginal costs are above the day-ahead prices in this period. If so, there will be no need for compensating the outages in the intraday market (Hagemann, 2013). Hence, reported outages are not necessarily reflected in price movements in the intraday market. Combining the UMM data with aforementioned *hourly and daily dummy variables* may account for different effects of power plant outages over the course of the day and the week.

3.3 Strategic Behaviour

Strategic behaviour among market participants may distort intraday prices beyond that of the physical market conditions assessed in Section 3.1 and Section 3.2. This section seeks to explain how such strategic behaviour may depend on the market participants' available price information

[24]For example, if consumption is scheduled day-ahead to be lower than usual, while the updated weather forecasts show that the temperature will be fairly low too, it might indicate that scheduled consumption will not represent actual consumption.

at the time of decision making (Section 3.3.1) and to what extent the participants may exhibit market power (Section 3.3.2).

3.3.1 Available Price Information

As intraday trading happens continuously, market participants may be expected to consider information about Elspot prices, the previous Elbas prices and/or the regulating prices when they make their trading decisions. Botnen Holm (2017) finds that prices in Elbas correlate with Elspot prices, though Elbas price spikes were not necessarily reflected in the day-ahead market. Pape et al. (2016) also include the day-ahead price in their fundamental model for determining the intraday price, and they find that doing so improves the model compared to a model where the day-ahead price is left out. Hence, they suggest that using day-ahead price information is beneficial when forecasting the intraday price. In our models, this can be done by including the aforementioned Elspot *spot prices* for each bidding area and the *system price*.

Past Elbas price information may also be of relevance. In GIME, including previous intraday prices is found to improve model performances (Kiesel & Paraschiv, 2017; Pape et al., 2016). Price information from previous intraday auctions are also important in predicting the next MIBEL intraday price (Andrade et al., 2017; Monteiro et al., 2016). Information on past Elbas prices can be incorporated in our models in two ways, namely the *Elbas volume-weighted prices for previous hours* of power delivery — that are available to participants at the time of prediction — and the *Elbas volume-weighted prices for trades already settled* for the hour of power delivery in question at the prediction time.

Imbalance costs for a given delivery hour are not available ex-ante, but both Scharff and Amelin (2016) and Mauritzen (2013) explain how *expected* imbalance costs are related to market participants' incentives to trade in Elbas to adjust their deviations. There is reason to believe that these expectations about future imbalance costs are, at least partially, based on the costs for previous delivery hours that market participants observe during the course of the day. However, by comparing intraday prices to regulating prices, Scharff and Amelin find that it may be hard for market participants to predict the direction of regulating prices. Thus, the way previous imbalance costs affect the intraday price is not definite. Still, due to the strong interaction between the Elbas and the regulating market, we consider it relevant to include *regulating price data for previous hours* of power delivery available at the time of prediction.

3.3.2 Market Power

Though electricity is a homogeneous product, and the Nordic and Baltic electricity market consists of over 370 producers (Nord Pool, 2018f), a few companies constitute larger shares than others (NordREG, 2014).[25] In addition, as transmission bottlenecks may constrain the possibilities of importing electricity from other bidding areas, these producers may constitute even larger shares in their local markets (Fridolfsson & Tangerås, 2009; Tangerås & Mauritzen, 2014). Joint ownership of power plants is also common.[26] Furthermore, there exist both economic and political barriers to entering the market (Tangerås & Mauritzen, 2014). As such, it is relevant to consider the degree to which market power may be present in Elbas.[27] Dominating players may be able to influence prices so as to obtain a profit margin, in addition to covering their power plant's marginal and ramping costs (Hagemann, 2013; Hagemann & Weber, 2013). For example, they may apply trading strategies such as retention of capacity or offering it at non-competitive prices to maximise profit. This may ultimately drive up intraday prices.

In particular, Tangerås and Mauritzen (2014) analyse to what extent producers may exercise market power in Elspot and Elbas, and find that Elbas prices tend to exceed Elspot prices during weekdays, while the relationship is reversed during weekends. The authors discuss how the day-ahead planned power production, and consequently also the transmission constraints, tend to be higher during weekdays than weekends, leaving less capacity for the Elbas market in these periods. Consequently, the observed price relationships may be due to producers exercising seller power in Elbas during weekdays.

It is challenging to fully account for the effects of participants exercising market power on the Elbas price in our models, as each individual participant is likely to behave based on unobservable factors such as the participant's market share and marginal production cost. As we do not know which participants are involved in a given Elbas trade, these factors are difficult to estimate. However, if market power to some degree is the explanation for the price effects observed by Tangerås and Mauritzen (2014), another important aspect related to a participant's ability to influence prices is highlighted; market power may not be constant, but rather depend on temporal effects. The possibility of producers exercising market power during peak hours where

[25]In terms of Nordic generation capacity, Vattenfall holds a market share of 19%, Fortum and Statkraft 12% each, and E.ON. 7% (NordREG, 2014).

[26]For example, Vattenfall, Fortum and E.ON jointly own all Swedish nuclear power (Tangerås & Mauritzen, 2014).

[27]In fact, Hagemann and Weber (2015) find that Elbas trading in Denmark is best described by an oligopolistic market model.

the market operates close to its capacity constraints, is also brought up in the context of the UK market (Hagfors et al., 2016). On the other hand, the argument can be made that *off-peak* hours are exposed to the use of market power. Section 3.1.1 explained how trading activity may affect intraday prices in the context of available production capacities. In addition, competition is likely reduced in these periods as smaller companies do not necessarily participate in the market. With fewer participants, opportunities for dominating participants to exercise market power may increase (Hagemann & Weber, 2013).[28] Given these temporal patterns, the way in which we may incorporate — at least partially — the effects of market power is through the aforementioned *production* and *transmission* capacity variables (Section 3.1) and the dummies for *hour-of-day and day-of-week effects.*

3.4 Conclusion

Liberalised electricity exchanges are complex markets, and their intraday markets are no exceptions. For the purposes of clarity, potential price determinants are subdivided into broader categories in this chapter. It should, however, be noted that these may be strongly interconnected, and it may be difficult to conclude how a given factor affects the market prices in isolation. For example, hour-of-day or day-of-week effects on the intraday price may both be explained by variations in load forecast errors and by variations in available intraday production and transmission capacities. And a combination of the these factors may in turn affect the possibility to exercise market power. Also, it is argued how higher day-ahead production volumes may lead to higher intraday prices. However, the effect may also be the opposite; Hagemann (2013) found that higher day-ahead prices rather showed tendencies to reduce, than increase, the intraday price. This unexpected finding is explained by the fact that relatively high day-ahead prices often occur due to relatively low predicted shares of wind and solar energy — hence, the probability of increased wind and solar production in the intraday market is higher than the probability of a decrease, consequently leading to lower, instead of higher, intraday prices. As these examples illustrate, it can be very challenging to deduce how each factor affect the intraday price due to the interplay between them being highly complex and at times even unexpected — a complexity which strongly motivates the use of deep learning in modelling the price.

[28]There are also factors that may limit the use of market power by a participant altogether. For example, Nord Pool or the TSOs are likely to detect producers that hold back power, e.g. wind, so as to increase price (Mauritzen, 2013).

4 Data

Section 4.1 introduces the data used in this thesis. It consists of almost 2GB of market data supplied by Nord Pool, and over 300GB of weather forecast maps from SMHI supplied by Montel — all of which cover the period from November 2011 to December 2017. Extensive pre-processing is necessary to make them conform to data structures accepted by our machine learning libraries, and to constitute a complete and consistent *timeline*. The market data consists of thousands of separate files of differing formats that must be combined into a single dataset with one observation per consecutive hour of power delivery. The required pre-processing steps are covered in Section 4.2. The weather forecast data, however, effectively consist of images that must be transformed to constitute a complete *timeline* of matrices that can be mapped to the market dataset. This necessitates a different set of pre-processing steps, which are covered in Section 4.3. Section 4.4 describes how we split the resulting timeline of two aligned datasets into training, validation, and test subsets. Finally, Section 4.5 provides an exploratory analysis of Elbas trading characteristics and the volume-weighted price.

4.1 Data Collection

Table 1 presents an overview of all datasets used in this thesis, including sources, the number of files per year, the time of publication, and data resolutions. Each of the datasets are briefly described in the following two sections. We limit the datasets' time period to start on November 2, 2011, as this is when Sweden was split into four bidding areas to better manage congestion in the power transmission grid (Nord Pool, 2009).[29] The last hour of power delivery is midnight[30] on December 31, 2017. In total, this corresponds to 54,048 consecutive delivery hours. After the pre-processing steps, this is reduced to 53,400 observations with some gaps in the timeline.

The historic Nord Pool market data files, with the exception of UMMs, are stored as daily or weekly files on their FTP-server. These are the files we merge and pre-process. However, for market participants, the data is published at the time specified in Table 1. To build models that could feasibly be implemented in practice, we therefore limit the models' access to data depending on what would be available in practice when making a prediction.

[29] I.e. the first hour of power delivery is 00:00–00:59 on November 2, 2011.
[30] Hour 24, i.e. the window 23:00–23:59.

Dataset	**Source**	**File format**	**Files per year**	**Published**	**Data resolution**
Elbas ticker	Nord Pool	CSV	365	Continuously	Continuously
Elspot file	Nord Pool	SDV	52	Day ahead at 12:42 CET	Hourly
Regulating prices	Nord Pool	SDV	52	1–2 hours after hour of power delivery	Hourly
Elspot capacity	Nord Pool	SDV	52	Day ahead around 10:00 CET	Hourly
Elbas capacity	Nord Pool	SDV	52	Day ahead around 14:00–15:00 CET	Hourly
Operating	Nord Pool	SDV	52	Day ahead around 14:00–15:00 CET	Hourly
Urgent market messages	Nord Pool	XML	Approx. 10–15,000	Continuously	Continuously
Weather data	Montel	GRIB	604	00:00 and 12:00 UTC	Each 3rd hour

Table 1: Information about the collected market and weather datasets.

4.1.1 Nord Pool Market Data

The Elbas ticker files contain data on each individual Elbas trade. Each file contains information on the time of trade, delivery hour of the power traded, price and quantity, the parties' respective bidding areas, and whether the trade was cancelled or executed. The Elspot files contain price and buy/sell volume data for each hour of power delivery and for each bidding area. The files are published the day ahead, usually at 12:42 CET. Regulating power data for each hour of power delivery and for each bidding area are found in the regulating prices files. The data is published 1 to 2 hours after the end of each delivery hour (Scharff & Amelin, 2016). The Elspot and Elbas capacity files contain data on transmission capacities between bidding areas for each hour of power delivery. Elspot capacities are published around 10:00 CET the day ahead. Elbas capacities are given as initial available capacities per delivery hour in the market. They are published the day ahead after trades are settled in Elspot, normally around 14:00 CET.

The separate files with operating data for each Elspot country contain prognoses for production and consumption for each hour of power delivery. The data are published around 14:00–15:00 CET the day ahead. Finally, the urgent market messages (UMMs) are stored as XML-files with one per published message, totalling around $10,000$–$15,000$ per year. UMMs are categorised as either concerning production, consumption, transmission capacity, or market information. They provide information about the volumes concerned, the time over which the outages will last, and various metadata. New UMMs are published continuously, but we only include those available at the time a prediction is made, and that were published after the Elspot gate closure.

4.1.2 SMHI Weather Forecasts via Montel

The weather forecast data take the form of gridded binary (GRIB) files, which are commonly used for meteorological maps. Each contains forecasts of five weather properties for all of Europe, made at a given date and time for a given number of hours ahead. Since these maps always cover the exact same geographical area, we treat these five properties as five channels of an image — similarly to the one black and white channel in greyscale images, or the red, green, and blue (RGB) channels in a colour image. These five properties, referred to as *bands*, are:

- Temperatures in Kelvin, at 2 metres above ground.
- Total precipitation in metres.
- Wind vector component U in metres per second, at 10 metres above ground.[31]
- Wind vector component V in metres per second, at 10 metres above ground.
- Surface net solar radiation in Joules per metres squared.

Forecasts are made at (UTC) noon and midnight every day for horizons up to 240 hours, in steps of three or six hours. Since the market data is a complete timeline of all 24 hours each date, we are therefore missing forecasts for two thirds of our delivery hours. Another limitation is that the weather data only contain forecasts, but no observed weather; their differences at hours preceding that of delivery could potentially be very informative when predicting the price.

4.2 Pre-Processing of Market Data

For the most part, the historic market data files seem formatted for human readability. As such, a number of pre-processing steps is necessary to extract the relevant data, then to transform and store it in data structures accepted by our machine learning libraries.[32] The purpose of these steps is to get a complete and *tidy* time series from November 2, 2011 to December 31, 2017 with one row per consecutive hour of power delivery, one column per feature (i.e. input variable), and

[31]A detailed explanation of atmospheric air flow vectors is provided by George Mason University (n.d.).

[32]We used the open-source softwares *R* and *Python* for the pre-processing, and the open-source machine learning libraries *scikit-learn* and *Keras* for building the neural networks.

one column for the output value (i.e. the response variable). As this thesis focuses on buyers in the Nordic countries in Elbas, we filter to data related to Nordic bidding areas, where relevant. For the Elbas ticker data, trades between a Nordic buyer and German seller are also included, as explained in Section 2.1.

For each dataset provided by Nord Pool in Table 1, the data from the raw files are read in for processing using the open-source language *R*. In order to combine the datasets into one consistent time series with one row per delivery hour, the datasets must have the same *grain*, or level of detail. Some have higher grain and must therefore be aggregated — such as the Elbas ticker data at the level of individual trades. Others have a *long* format where a given delivery hour is repeated for different values of some variable, and must be transposed to a *wide* format with such combinations instead stored as separate features — such as the Elspot files, with spot prices for multiple areas expressed as separate rows for the same delivery hour, which must instead be transposed to one row of separate columns.

The rest of this section describes key pre-processing steps. After the Nord Pool datasets have been transformed and compiled into one cohesive time series, the resulting *market dataset* is exported to *Python* and pre-processed further for use in modelling. The resulting sets of in total 328 final input variables are summarised in Table 14 in Appendix F.

Limiting to the data available for market participants

To build models that can feasibly be used in practice, we limit their access to data based on what is realistically available to market participants when making a prediction six trading hours in advance of delivery. As such, the latest available Elbas price is that of the delivery hour that precedes it by six hours,[33] hence we add corresponding lagged variables of the price and volume. Regulating prices and dominating directions are also published one to two hours after delivery (Scharff & Amelin, 2016), hence the related variables are lagged by eight hours.

Dealing with missing input values

Nearly 3% of the 264 input variables' values are missing, but the vast majority belong to 6 features with nearly 100% sparsity[34]. Dropping these variables reduces it to 0.3%[35], but our

[33]Market participants can see individual trades, but we aggregate the last six hours into one volume-weighted price. Hence, the latest available such price is that of six hours ago. Section 4.2.1 provides more details.

[34]*Sparse* data have a lot of gaps, or missing values.

[35]Table 16 and Table 19 in Appendix F.2 give an overview of the remaining 46 variables with missing values, and how many that are missing for each.

models cannot handle missing values directly. We therefore replace them with the value zero.[36] For any relevant feature, we also add a "missing" indicator saying whether the associated feature was missing for the corresponding observation. This combination of zeroimputation and missing-indicators[37] was successful in other work, both related to deep learning (Lipton, Kale, & Wetzel, 2016) and linear modelling (Wooldridge, 2016).

Outliers and missing output values

Similarly to Lago, De Ridder, Vrancx, and De Schutter (2018), we do not omit outliers of the output variable, in order to train models that are better at predicting significant spikes in the price. However, the dependent variable is missing for 156 of $54,048$ observations. When the preceding and succeeding values exist, the missing value is simply imputed linearly. This more than halves the number of missing values, and the remaining gaps are removed by dropping the corresponding dates of power delivery.[38] The resulting dataset contains 53,400 observations.

One-hot-encoding categorical variables

Similarly to dummy variables in traditional methods, *one-hot-encoding* splits categorical variables into indicator variables corresponding to each level of the category. This makes it easier for the model to capture their meaning, as it might otherwise treat them as *ordinal* variables whose relative order of levels holds meaning — such as a quality rating. Thus, we one-hot-encode categorical variables related to the hour, weekday, month, and each Nordic buyer area's regulating price dominating direction. The number of features therefore increases from 264 to 328.

Standardising continuous numeric variables

Neural networks have been found to perform better when continuous numeric variables have similar scales (Hastie, Tibshirani, & Friedman, 2009). Common approaches are to either scale variables in the range $[0, 1]$ — referred to as *normalisation* — or to centre each variable around zero mean and with unit variance — referred to as *standardisation.*[39] Normalising is simpler and makes no assumptions of the distribution of the variables, but is sensitive to outliers. *Standardising* is less susceptible to outliers, but assumes the variables follow a Gaussian distribution. Most

[36]For some variables the value 0 already has meaning, in which case replacing missing values with zeroes indubitably adds noise. However, there are so few such cases that we asses the downside to be negligible, compared to using a more convoluted strategy of handling missing values.

[37]There are more sophisticated methods that potentially could have been used. However, due to time constraints and the degree of missing values being limited in scale, we chose the imputation/indicator approach.

[38]Alternatively we could drop only the observations, but we would like to build sequential models based on whole days of data. Hence, we drop the 27 dates that contain the remaining 72 missing delivery hours of prices.

[39]Appendix C provides details.

of our numeric variables are relatively normally distributed[40], hence we standardise them all[41] based on their parameters in the training set[42]. Standardisation is done before missing values are replaced, so the non-missing values are unaffected by the imputation. The one-hot-encoded categorical variables and "missing" indicators are excluded, as these are already binary.

4.2.1 Calculating Aggregated Elbas Buyer Prices

As there may be several Elbas trades for each hour of power delivery, we use a *volume-weighted average price* (VWP) of trades for a given hour of power delivery as the hourly intraday buyer price. This is the output variable our models will be trained to predict. A similar approach is taken by e.g. Knapik (2017) and Tangerås and Mauritzen (2014)[43] in their analyses. For each hour of power delivery, h, the volume-weighted average price, VWP_h, is calculated as:

$$VWP_h = \frac{\sum_{n=1}^{N}(P_{n,h} \times V_{n,h})}{\sum_{n=1}^{N} V_{n,h}} \tag{1}$$

where $P_{n,h}$ is the price (EUR/MWh) and $V_{n,h}$ is the volume (MWh) in trade n for delivery in hour h. N is the total number of transactions for hour h.

As we predict an aggregate VWP over the six hours remaining until delivery, it is likely some trades for the corresponding hour have already been settled. This information is available to traders, and we engineer a feature that captures the VWP over all hours of trading up to the point where a prediction is made. We do not divide it into shorter intervals, as this resulted in excessive data sparsity, which would necessitate extensive imputation and make it too noisy to be useful. Hence, we predict the *near* VWP, whereas this additional feature of information on preceding trades is the *far* VWP. An illustration of calculating these two prices is provided in Figure 4. The *far* price for delivery at hour 18 is calculated by aggregating trades settled between when the corresponding Elbas market opened at 14:00 CET the preceding day, and when we make a prediction at the beginning of hour 12. The *near* price is what our model aims to predict, and represents an aggregate of any trades settled from, and including, hour 12 and when trading closes at the end of hour 16.

[40]Appendix C.1 shows the distributions of some of the continuous numeric variables.
[41]A possible improvement could be to instead normalise those variables that are clearly non-normal, but we saw impact on performance from we tested doing so.
[42]We thus avoid passing information from the validation or test sets to the training set in allowing variables' values and distributions in those sets to influence what the model sees during training. These sets should be out-of-sample, and are therefore scaled using parameters computed on the training set.
[43]Tangerås and Mauritzen (2014) compute the volume-weighted average on a daily frequency.

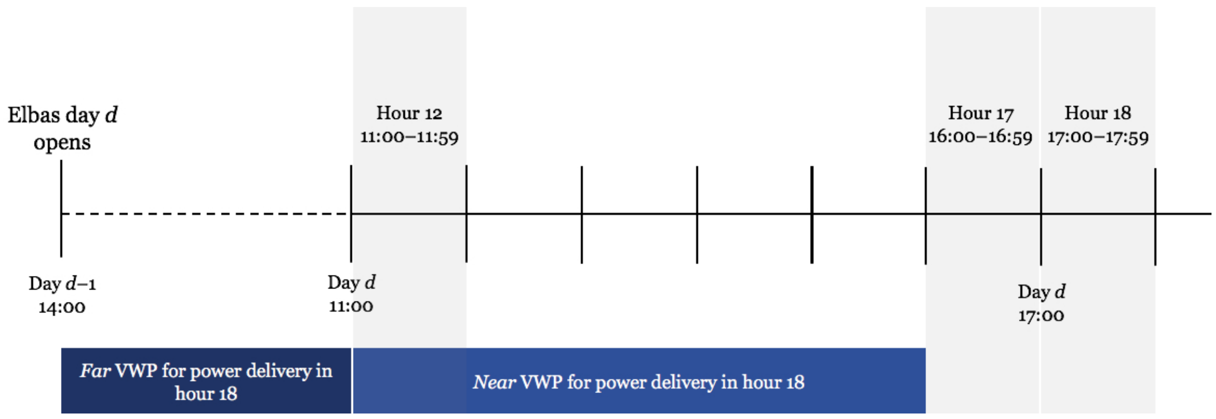


Figure 4: Calculation of a *far* and a *near* volume-weighted average intraday price (VWP), with price prediction for power delivered during hour 18 as an example.

4.3 Pre-Processing of Weather Data

The GRIB files cannot be consumed by our neural networks directly, hence the binary data must first be extracted and restructured as consistent *tensors*[44]. Each such file contains forecasts of five weather properties, or *bands*, as explained in Section 4.1.2, which effectively represent five maps of Europe. Similarly to colour images with the three channels red, green, and blue (RGB), we treat these five bands as five channels of one image, as they cover the exact same geographical area. The file structure is consistent until the middle of 2017, so that we can simply assign each band to the correct channel based on its ordering in the binary file. Starting with July 2017, however, the ordering of the five bands not only changes, but also varies — seemingly randomly. Trying to manually ensure consistency among these shuffled bands would be impractical due to the amount of data. Instead, we fit a designated machine learning model to automatically re-order these input data by learning to classify bands using aggregate measures.

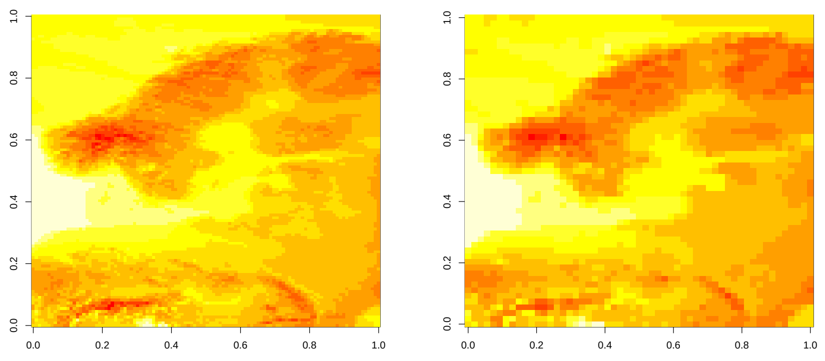
The complete set of files is over 300GB, hence we need to limit it considerably if we are to feasibly include weather forecast data in models built in the time and on the hardware available. Once we have limited the scope of the data, we also scale the images — similarly to the market data — to improve model performance. Finally, each forecast's set of five bands is mapped to corresponding delivery hours so that it can be joined with the market data and, therefore, the associated volume-weighted Elbas prices. Each raw GRIB file is read into R, where the relevant matrix that comprises each band is separated and extracted, then converted and exported to

[44]Tensors are the basic data structures for storing numerical data used by most modern deep learning systems (Chollet, 2018). They are a generalisation of 2-dimensional matrices to an arbitrary number of dimensions.

Python, where the below steps are performed. The resulting weather dataset is just shy of 6.5GB, which fits into memory alongside the market data on the available hardware.

Cropping and downsampling the images

The original maps cover all of Europe. Since we limit the market data to Nordic buyer and seller areas (with the addition of Germany among the latter), we reduce the enclosed geographical area correspondingly by cropping the images to that region. Each image is also *downsampled* by reducing squares of four contiguous pixels to one that contains their average value. Halving both dimensions reduces image sizes, or resolution, by 75%. An example is shown in Figure 5.



(a) Cropped original (b) Cropped downsampled

Figure 5: Downsampling a temperature forecast for Northern Europe by averaging.

Limiting to one forecast per delivery hour

Each delivery hour is mapped to one forecast (i.e. set of five bands), and the rest dropped. Each mapped forecast is the latest available for the given delivery hour when the prediction is made six trade hours in advance. Although we would like to incorporate multiple forecasts for each delivery hour to let the neural networks map changes in the forecasts, this would come at the cost of a proportionate increase in memory use[45]. Though there are a few solutions that could be engineered[46], we consider this simplification necessary due to time and resource constraints.

Interpolating absent forecasts

Since forecasts are only made at midnight and noon with horizons in increments of three hours, there exist no forecast for two thirds of the delivery hours in the timeline. These are therefore

[45]Strictly speaking, the datasets do not have to fit into memory. However, retrieving it from the solid state drive would significantly slow down the preparation of every batch during training and evaluation.

[46]Such as downsampling further, or creating new images that correspond to the difference between two or more forecasts.

linearly interpolated using the latest available forecasts for the closest preceding and succeeding delivery hours whose forecasts exist.[47]

Normalising the weather data

Similarly to the market data, the resulting images are scaled to improve model performance. Traditional images' pixel values are in the range $[0, 255]$, and therefore typically scaled to $[0, 1]$ by dividing by 255. In contrast, our "images" correspond to weather properties that are not necessarily within similarly fixed ranges.[48] Instead, we *normalise*[49] each band's values to $[0, 1]$ based on the extreme values of each band observed in the training set.

Rectifying inconsistent input data with machine learning

The last six months of randomly ordered bands must be made consistent with the rest of the weather data. Rather than attempting to do it manually, which would be very time-consuming and arguably unreliable, we instead treat the nearly 6 years of consistent data as a training set and fit a Random Forest classifier to automatically re-order the inconsistent bands. The model classifies bands with over 80% accuracy on the data with known ordering, as estimated using 10-fold cross-validation. *All* errors stem from it struggling to separate the wind vectors. After we manually impose additional decision rules based on more likely sequential orderings[50], the model agrees with our own best guess in well over 95% of cases on the inconsistently ordered data.[51] Though not ideal, we assert that this level of input weather data quality is sufficient, as the alternative would be having to halve the test set.

4.3.1 Matching with the Volume-Weighted Average Price

The resulting set of scaled weather forecast maps, or images with 5 channels of 55 by 55 pixels, comprise a complete timeline corresponding to that of our market dataset. As such, we can map

[47]Due to the six-hour difference between when a prediction is made and the delivery hour arrives, the most recent forecast available for the closest preceding hour may not be the same as when the price was predicted for that hour. In other words, a more updated forecast for that hour may have arrived in the meantime. If so, this updated forecast is used to interpolate.

[48]For instance, temperatures are measured in Kelvin with no a priori hard limits — other than some absolute extreme like 0 degrees Kelvin, which would never be observed (even in northern Scandinavia). Hence, scaling by pre-determined values such as these would compress the pixel values between an artificially constrained range.

[49]Normalisation is explained in Appendix C.

[50]Random Forests process each sample individually, and therefore do not capture sequential information unless explicitly included as distinct features.

[51]Since the correct order of the shuffled data is unknown and the training data does not have that problem, the only way to evaluate the impact of these manually defined decision rules is to compare the model's resulting restructuring with our own best guess. Though we are clearly fallible, the model agrees in slightly less than 100% of cases. Hence, since we cannot do much better, we feel confident in using the Random Forest model to rectify the last six months of weather input data.

a corresponding price to each set of forecasts, as illustrated in Figure 6.[52] Designing a neural network using only this data effectively amounts to placing a camera in front of a televised weather report and instructing it to predict a price in Euros. As absurd as it may sound, we did test a few such networks to assess whether the weather data seemed to hold any predictive power. These networks did surprisingly well, especially if also given explicit information about the time of day and time of year — essentially showing the camera a clock and a calendar for context, so that the networks would not have to waste capacity on determining its location on seasonal curves. Appendix D.6 provide details on these experiments.

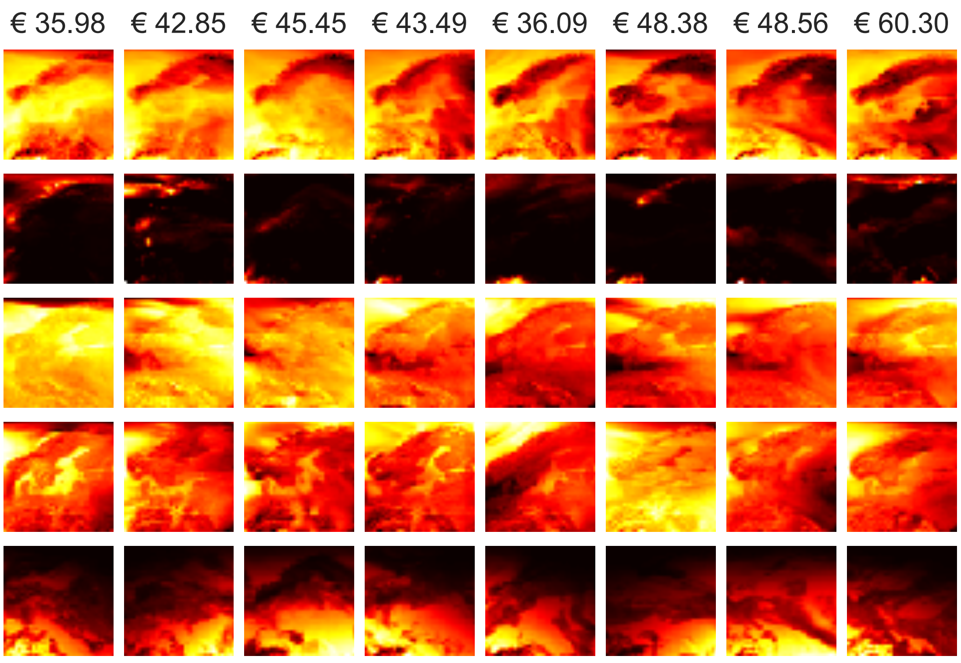


Figure 6: Forecasts of the five bands and the corresponding VWP (EUR/MWh) at delivery hours in intervals of 36 hours, starting at midnight on November 2, 2011. From the top, each row corresponds to a band: temperature, precipitation, two wind vectors, and solar radiation.

4.4 Splitting into Training, Validation, and Test Sets

The challenge of all supervised[53] machine learning is capturing patterns in the training data that allow the model to generalise its performance to unseen data (Hastie et al., 2009). During training, the algorithm adjusts its parameters by learning statistical structures in the *training*

[52]Note that all plots of the weather data are *heat maps* that colour pixels based on their relative value within that band; hence, a given shade of red in one band has no relationship with the same shade in another, or even between images of the same band (as the range in each image may differ).

[53]Where the desired outcome, at least on the training data, is known, so that the algorithm can strive to be as correct as possible in as many cases as possible.

set, but will at some point start memorising relationships that are specific to this data rather than representative of the true data-generating process (Chollet, 2018). They will, in other words, tend to *overfit* to the training data, so that we need a separate *test set* on which to estimate their performance on unseen data. However, most machine learning methods involve one or more *hyperparameters* that control some aspect of how the method is applied (Hastie et al., 2009), where deep learning is particularly notorious for having many such hyperparameters (Chollet, 2018). These are defined externally, and usually tuned by evaluating how models with different configurations perform on a *validation set*[54]. Since one is therefore indirectly overfitting to this set, it should be separate from the test data (Chollet, 2018).

The combined dataset constitutes a continuous timeline[55]. Since we would like to build models that explicitly capture this temporal structure, we simply split the timeline into three contiguous sections, rather than creating three randomly sampled sets. Some form of *resampling* or *cross-validation* might yield more robust and reliable estimates of model performance on unseen data (Hastie et al., 2009), but would be cumbersome to implement while keeping the temporal structure intact, and be prohibitively time-consuming due to repeated training of deep learning networks that are inherently computationally costly to fit (Chollet, 2018). Instead, we argue that we have sufficient data for a train–validation–test split to yield reliable results. Thus, we set the test set to consist of the last year of data, and the validation set to consist of the preceding year. These sets therefore each contain $8,760$ observations, while the training set contains the preceding $35,880$ observations — constituting a 68–16–16 split of the data.

4.5 Exploring the Data

Before proceeding to the methodology employed in the thesis, we take a closer look at trading in Elbas and the *near* VWP. In addition to enhancing our understanding of the market, trading and price characteristics are important when evaluating the practical reliability of models. Section 4.5.1 explores various aspects of trading in the Elbas market. Then, Section 4.5.2 examines the *near* Elbas VWP, its development over the training, validation and test sets periods and how the price level and volatility may depend on delivery hour.

[54]Machine learning models will generally overfit to the training data for most hyperparameter values within reasonable ranges, meaning the training set is an effective basis on which to tune these values.

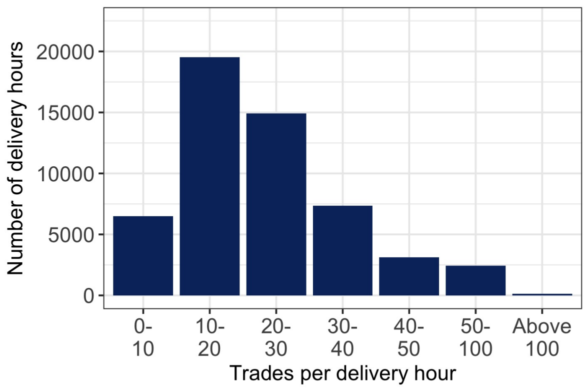
[55]Except a handful of gaps after we remove 27 dates where at least one delivery hour is missing its output value.

4.5.1 Characteristics of Elbas Trading

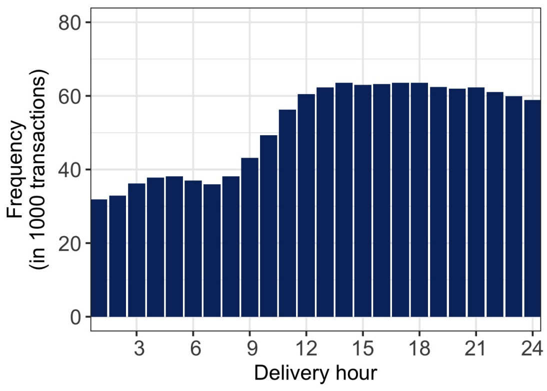
This section examines the distributions of trades per delivery hour, prices and volumes per trades, and the degree of variation in trade prices in our dataset. The dataset contains all trades in the complete timeline over the period November 2, 2011 to December 31, 2017.

Trading activity

Many trades may be settled in Elbas for power delivered during a given delivery hour. More than 10 transactions are settled for most delivery hours, as shown in Figure 7a, with an average of 23 trades per delivery hour (Table 15, Appendix F.2). However, there are examples of up to 986 trades being settled for one single hour of power delivery. The distribution in Figure 7b shows that trading activity may vary depending on the delivery hour in question. More trades are settled for power delivery in the second half of the day than in the first. This is not surprising as the probabilities of forecast errors or other unforeseen events increase the more time passes from the settlements in the Elspot market. In addition, similarly to the findings in Scharff and Amelin (2016), trading activity in our dataset tends to match office hours (Figure 25, Appendix B.3), which may also explain why there are more trades settled for later delivery hours than for earlier. Regardless of the reason, we conclude that the trading markets for the later delivery hours seem to be more liquid than the markets for the earlier ones.



(a) Number of trades across seven buckets



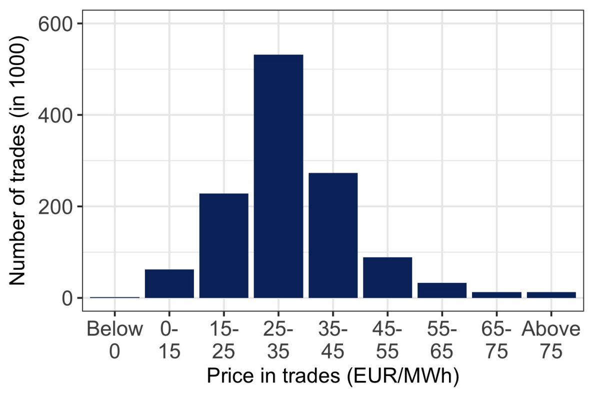
(b) Number of trades across delivery hours

Figure 7: Number of Elbas trades per delivery hour.

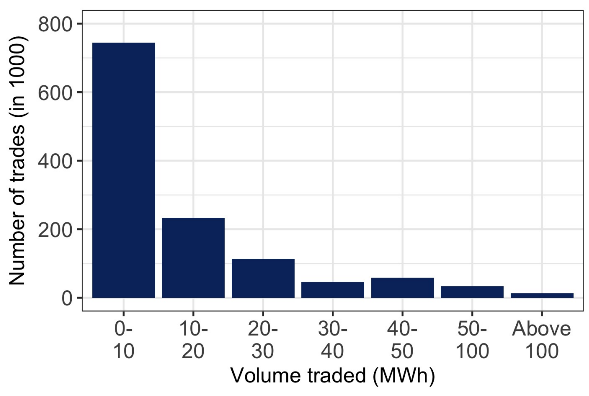
Trade prices and volumes

Figure 8 shows how the trade prices and volumes are distributed in our dataset. Per Figure 8a, the Elbas price range 25–35 EUR/MWh includes approximately 40% of the trades. Expanding the range to 15–45 EUR/MWh covers 80%. On average, the price per trade was approximately

33 EUR/MWh, while the highest and lowest prices observed in the period were 555 EUR/MWh and -150 EUR/MWh, respectively (Table 15, Appendix F.2). Figure 8b shows that a majority of Elbas trades concern relatively low volumes, with volumes of 0–10 MWh and 10–20 MWh as the most frequent categories, respectively; together, they account for nearly 80% of trades.[56] The average volume traded is 10 MWh, and the highest observed is 890 MWh (Table 15, Appendix F.2). The fact that most trades concern relatively low volumes may be advantageous to market participants, in that it allows for more flexibility in meeting the desired buy volume.



(a) Trade prices



(b) Trade volumes

Figure 8: Prices and volumes per Elbas trade.

Variation in trade price

Figure 9 shows how the variation in trade price is distributed, where spread is measured both by the *interquartile range* (IQR),[57] and the distance between the lowest and highest trade price for a given hour. For the majority of delivery hours, the IQR is within 0–10 EUR/MWh. However, as pointed out by Scharff and Amelin (2016), trades for power delivered during the same hour may vary significantly in price. This tends to be the case also in our dataset, as the spread between the maximum and minimum price exceeds 10 EUR/MWh for several delivery hours.

[56] An explanation of the observed low trade volumes may be the use of Iceberg orders (IBO) in Elbas. IBOs conceal the full order size from the market by dividing it into smaller clips (Nord Pool, 2018e).

[57] The prices may divided into quartiles, where each quartile represents 25% of the observations. The *first* quartile represents the bottom 25%, the second 50%, the third 75% and finally, the fourth quartile covers all observations. The *interquartile range* is the distance between the first quartile and the third quartile.

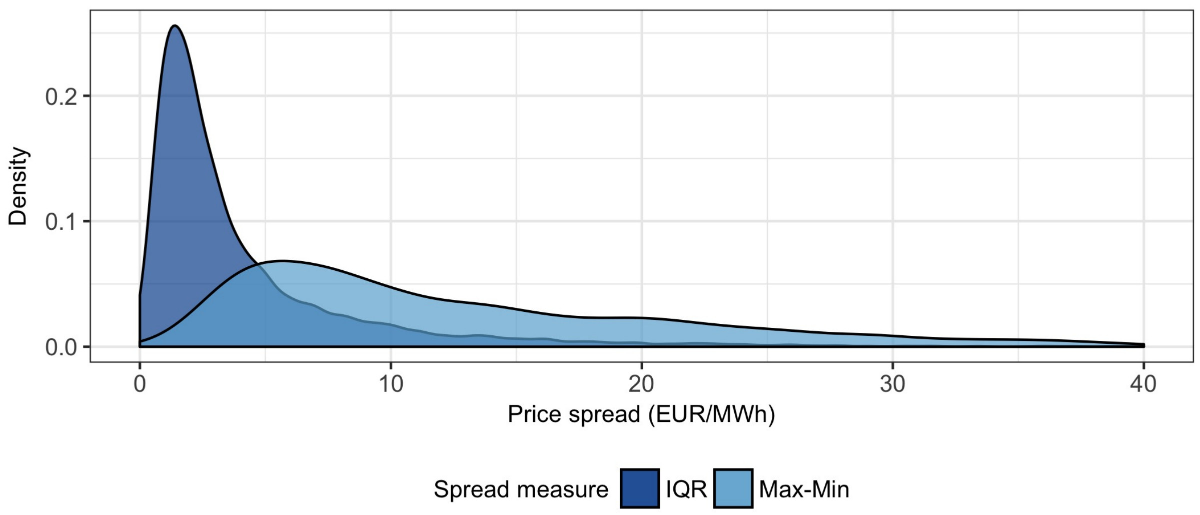


Figure 9: Variation in Elbas trade price for the same hour of power delivery. There is a long tail to the right beyond 40 EUR/MWh, however, due to very few observations, it is left out here.

4.5.2 The Elbas Volume-Weighted Average Price

Price development

Figure 10 shows the development of the *near* Elbas VWP over the dataset timeline, split into the training, validation and test sets, and Table 2 provides summary statistics for the *near* Elbas VWP for each of the three sets. Both the highest and the lowest VWP over the full period occurred around the beginning of 2012, and their respective values were 259.55 EUR/MWh and -75.68 EUR/MWh. Hence, the training set contains the most extreme values observed in the period. In the validation set, the VWP peaked in the beginning of 2016, and dropped below zero (to -22.84 EUR/MWh) once, at the end of 2016. The test set period generally exhibits less extreme fluctuations, with peaks just above 100 EUR/MWh, and the lowest point just below zero at the end of 2017.

In the training set, the IQR goes from 25.63 EUR/MWh to 38.83 EUR/MWh, with a median of 32.09 EUR/MWh. The values are somewhat lower for the validation set, where the IQR is 21.91–34.16 EUR/MWh and the median is 27.58. For the test set, the upper limit of the 1st quartile is similar to that in the training set, but the 3rd quartile ends on a value that is slightly lower. The median, 29.97 EUR/MWh, is also slightly lower.

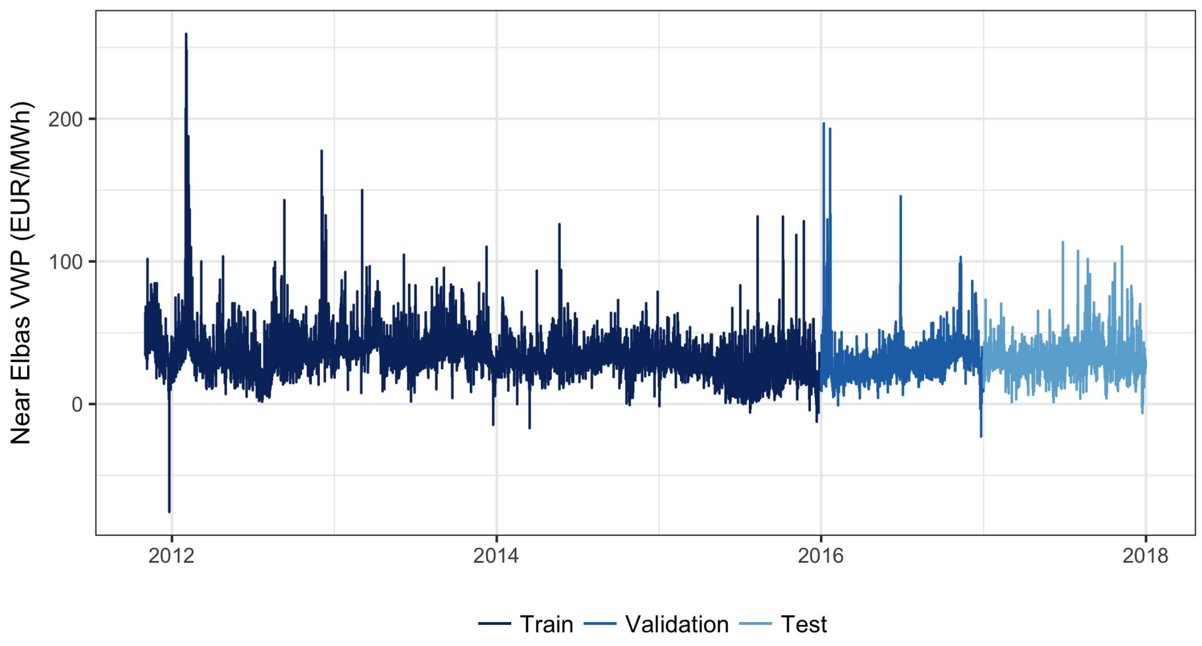


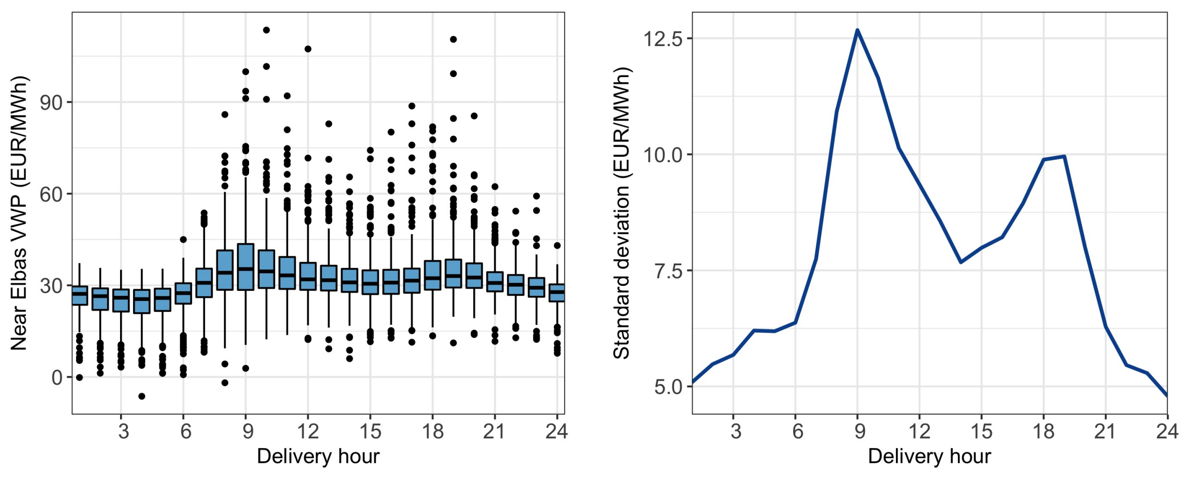
Figure 10: Development in the *near* Elbas volume-weighted average price over the period November 2, 2011 to December 31, 2017. The training set period (dark blue), validation set period (”middle” blue) and test set period (light blue) are also indicated.

Dataset	**Minimum**	**1st Qu.**	**Median**	**Mean**	**3rd Qu.**	**Maximum**	**Stdev**
Training set	-75.68	25.63	32.09	32.83	38.83	259.55	13.07
Validation set	-22.84	21.91	27.58	29.19	34.16	196.74	12.07
Test set	-6.324	26.26	29.97	30.95	34.36	113.56	9.07

Table 2: Summary statistics for the *near* Elbas VWP in the training, validation and test sets.

***Near* VWP across delivery hours**

The level and volatility of electricity prices may vary depending on delivery time (Hagfors et al., 2016), which is also the case in our dataset, as shown in Figure 11. The *near* VWP is generally higher and more volatile, measured by the standard deviation, during delivery hours 7–20, with notable peaks around hours 8–9 and 18–19. As such, gauging how performance varies across delivery hours may be relevant when contrasting the different models and benchmarks.



(a) Distribution (b) Volatility

Figure 11: Distribution and volatility of the *near* VWP across delivery hours. Data is from the test set period.

5 Methodology

Per Section 1.1, we aim to address the research questions:

1. To what extent can deep learning predict the volume-weighted average Elbas price six hours ahead of a given hour of power delivery, and how reliable are the forecasts in practice?
2. What does this suggest about the potential of deep learning in wider applications in the Elbas market, and what are the salient hurdles to implementing such AI agents?

We therefore need a set of metrics to evaluate and compare how various models perform on our prediction problem. This allows us to both asses in absolute terms how well various implementations of deep learning perform, but also to compare against benchmarks to assess the relative value-add versus the added complexity of these implementations. To this end, we define and develop a set of simple heuristics and baselines from more traditional techniques to serve as benchmarks. With this empirical harness in place, we experiment extensively with a range of various deep learning implementations, before selecting a final set of model designs with which to thoroughly evaluate the effectiveness and pragmatic value of deep learning in intraday electric markets. In the interest of brevity, the focus is on these final implementations, rather than all the underlying experiments that lead up to them.

The following sections outline the choice of metrics (i.e. how we measure success), the simple domain-based heuristic baselines, the more traditional benchmark models used, and finally the design and implementation of the various deep learning networks developed. We assume the reader has some knowledge of more traditional techniques in keeping the descriptions of these to a minimum, but do provide simple summaries in Appendix E. Seeing as deep learning is the focus of this thesis, we provide some brief descriptions of how various types of deep learning function, but focus on the reasoning behind our design choices and on providing a sufficiently detailed overview for reproducability. Appendix A is intended to be a self-contained introduction to how various forms of neural networks work from a technical, albeit accessible perspective, and includes more detailed explanations of established concepts used in our implementations. The specific sections of the appendix are referenced for convenience in footnotes, where relevant.

5.1 Model Evaluation

For a given problem we assume there is a true, unknown relationship f^* between the observed outcome y and the input predictors $\mathbf{x}$ such that $y = f^*(\mathbf{x}) + \varepsilon$, where ε is an independent random error term with assumed variance of 0 (Hastie et al., 2009). We estimate this relationship to make predictions that are as close to the true outputs as possible: $\hat{y} = f(\mathbf{x}) \approx y$, where ε averages to zero and is excluded. Our main concern is getting accurate predictions rather than knowing the exact functional form of f^*. This accuracy depends on a *reducible* and an *irreducible* error. The former can be mitigated by better statistical learning methods or ways to estimate f, but the former stems from the ε component of the true y, which by definition is random and cannot be predicted using $\mathbf{x}$ (Hastie et al., 2009). Most *supervised*[58] machine learning algorithms tune a set of parameters to estimate f with the goal of minimising some *loss function* — i.e. some chosen measure of how well the model performs. For regression problems, it is common to use either the *Mean Absolute Error* (MAE) or *Mean Squared Error* (MSE) — alternatively the *Root Mean Squared Error* (RMSE) — although it is possible to use tailor-made loss functions to reward or penalise specific behaviours (Chollet, 2018).

We choose MAE as the loss function, for two reasons. First, we want models that can be better relied upon in typical day-to-day situations. Second, we believe such models will better complement analysts and decision-makers — or even specialised models — that are better suited to spotting exceptional situations, such as price spikes. As Equation 2 shows, MAE is calculated

[58]Supervised problems are where the output is known, so that the algorithm can iteratively improve its performance compared to a defined ”correct” answer.

as the mean of the absolute deviation between the predicted value $\hat{y}$ and true value y, over the samples $i = 1, ..., N$. Hence, MAE penalises all errors proportionately, which increases performance in general, but at the cost of greater magnitudes of peak errors. In other words, being better across the board means that the model will miss by more when it first does miss.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{2}$$

Although we use MAE as the loss function to train the models, RMSE is also reported as an additional performance metric to inform model choice and design. RMSE is calculated as the square-root of the MSE, as shown in Equation 3. As the deviation between predicted and target values is squared, RMSE penalises larger deviations more severely. Hence, reporting both MAE and RMSE allows for a deeper understanding and evaluation of models' merits and limitations.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{3}$$

The MAE and RMSE metrics are reported on both the validation and test sets, as the latter is truly unseen data and a better representation of how our models would perform in practice.

5.2 Benchmark Models

Two categories of benchmarks are developed: simple domain-based heuristics, and models from among more traditional econometric and machine learning methods. The simple heuristics reflect the degree to which we are able to predict the *near* Elbas VWP without building a model, while the econometric and machine learning models represent simpler approaches to price modelling than deep learning — although they require much of the same data preparation. The simple heuristics are explained in Section 5.2.1, while Section 5.2.2 covers the methodology for developing the econometric and machine learning benchmarks. The benchmark models only use the Nord Pool market data, as it would have been very cumbersome and too computationally expensive to also integrate the weather data.

5.2.1 Simple Heuristics

We define six simple heuristic benchmarks, beginning with benchmarks that use the Elbas price information that is available to the market participants at the time of prediction. Two such approaches are to always predict that the *near* Elbas VWP will equal the previous observed values. Here, variables lagged with 6 and 24 hours are particularly relevant. The 6-hour lagged *near* VWP represents the latest available Elbas price information at the time of prediction, as the prediction is done six trade hours in advance. The 24-hour lagged *near* VWP reflects the VWP for a given hour the day before, and may capture patterns in the Elbas price that depend on the hour of power delivery. Another way of incorporating available Elbas price information is to predict the *near* VWP in a given hour to simply be the mean of the *near* VWP, which in the training set is 32.83 EUR/MWh. For observations where it exists, the *far* VWP — that is, the volume-weighted price of all trades concerning the given delivery hour that have occurred before the time of prediction — may also be a good indicator of the *near* VWP.[59]

The Elspot market's system-wide and area spot prices might also be effective simple heuristic benchmarks. While the spot prices are calculated per bidding area, the system price is calculated for the market as a whole, similarly to the *near* Elbas VWP that is aggregated across all Nordic bidding areas. In terms of specific area spot prices, the Swedish bidding area SE3 has the highest volume bought in Elbas over the period November 2, 2011 to December 31, 2017 (Figure 24b in Appendix B.2). In total, the volumes bought by buyers in SE3 constitute 24% of all volume bought in Elbas. As such, the spot price in SE3 may have a larger influence on the *near* Elbas VWP compared to the spot prices in other bidding areas.

5.2.2 Econometrics and Simpler Machine Learning

The benchmark models from more traditional methods are run on a laptop using the open-source software R, so as to be representative of approaches that require minimal investment in software or hardware. Except from lags, no variables were engineered for the benchmark models beyond those available in the deep learning dataset.[60] We avoid generating new variables in order to make the benchmark approaches comparable to the more advanced deep learning approaches — that is, we would like any differences in performance to be due to the technique employed, rather

[59] For about 13% of the observations, there is no *far* VWP that may be used to predict the *near* VWP.

[60] For example, one could have considered to aggregate the capacity variables to variables containing total capacity in and out of each bidding area.

than the data used. The rest of this section explains the methodology employed to deal with each of the following elements in constructing the econometric and machine learning benchmarks:

1. How to incorporate the temporal structure of the dataset, as there is a gap of 6 hours between prediction time and delivery hour.
2. How to deal with the high dimensionality of the dataset.
3. How to account for non-linear relationships between the variables.

Capturing the temporal structure

Since there is a gap of six hours between the time at which we make the price prediction and the hour we make the prediction for, time-series models that incorporate directly preceding output(s) as input(s) when making a prediction — such as *autoregressive integrated moving average* (ARIMA) models — have not been employed. If we make the prediction at time $h-6$ for the *near* Elbas VWP at hour h, we do not know at the time of prediction, $h-6$, the VWP for the hours $h-1$, $h-2$, $h-3$ and so on. The latest VWP we have information about is the one at $h-6$. To leverage the temporal structure of the dataset, we instead add lagged variables manually. They are included for the dependent variable, the *near* VWP, and among the independent variables for the Elspot system price and the spot prices in the various bidding areas. These variables have been identified to have the highest correlation between their lags and the *near* VWP at time $h=0$.[61] To avoid a considerable expansion in the dataset's dimensionality, we do not add lags for the remaining variables.

Handling high-dimensionality

Our dataset contains a high number of variables. To identify the variables that are most important for predicting the *near* Elbas VWP six hours ahead, we have tried a number of techniques. These are briefly described below, while Appendix E expands on their theoretical frameworks.

- *Variable selection based on literature*: As the importance of previous price information is highlighted in literature, we build a linear regression model with only Elspot prices (system price and area spot prices) and prior Elbas prices (*near* VWP lag and *far* VWP) as inputs.
- *Feature selection*[62]: We apply automated feature selection using random forest (RF) and multivariate adaptive regression splines (MARS), where, for each method, the most important variables are extracted. A linear model is then fitted, using least squares, on the

[61]Specifically, for each variable, we add 22–26 hours and 46–50 hours lags.
[62]Details are provided in Appendix E.2.

reduced set of variables.

- *Shrinkage methods*[63]: All input variables are included, but a regularisation term, $\Omega(\beta)$, is added to the loss function to shrink the coefficient estimates towards zero. This is done to reduce the variance, at the expense of slightly higher bias. We tested both ridge regression, LASSO[64], and elastic nets, which are hybrids between ridge regression and LASSO. The tuning parameter λ controls the extent of regularisation imposed on the loss function, and the optimal values are found using cross-validation. For the elastic nets, we experiment with α values of $0.2, 0.4, 0.6$ and 0.8.[65]

- *Principal component regression*[66]: This is a feature extraction method where dimensionality is reduced by linearly transforming the original input variables to principal components. Each component captures the largest (remaining) variance amongst all linear combinations of the set of variables. The n first components are then used to fit a linear model using least squares, where we let n vary from 10 to 200 in steps of 5 components. We choose the number of components, N, that results in the lowest MAE on the validation set.

Accounting for non-linear relationships

We experiment with integrating second and third degree polynomial terms in the linear models to allow for non-linear relationships between the *near* VWP and the independent variables. To further account for non-linear relationships, we also build a non-linear baseline model using the gradient boosting algorithm[67], which has been found to perform well in, for instance, predicting intraday prices in the MIBEL (Andrade et al., 2017). The optimal number of trees, t, in the model is found by letting t vary from 100 to $10,000$ with interval steps of 50 trees. We select the number of trees, T, that results in the lowest MAE on the validation set.

5.3 Deep Learning

Deep learning is particularly well-suited to the aforementioned three areas of focus: there are specialised network types for sequential data that capture temporal structures; they essentially automate feature engineering and -selection by prioritising and learning hierarchies of increasingly abstract representations of the inputs, making them especially well-suited to high-dimensional

[63]Details are provided in Appendix E.1.

[64]Least Absolute Shrinkage and Selection Operator

[65]The higher the value of α, the higher weight put on the LASSO term, while a lower value of α puts a higher weight on the ridge regression term.

[66]Details are provided in Appendix E.3.

[67]Details are provided in Appendix E.4.

data (Chollet, 2018); finally, they can learn arbitrarily complex non-linear mappings, and are therefore theoretically suited to approximating virtually any non-linear relationship (Goodfellow et al., 2016). In addition, there are specialised network types for image processing, which will enable us to fully utilise our weather forecast maps.

We begin with networks that only use the market data. This allows for more direct comparisons with our benchmark models, as these then use the exact same input data. Additionally, it allows us to evaluate the value-add of also incorporating the weather forecast data, as these networks should arguably beat those that only use market data to justify the increased complexity and computational cost. We build a set of final architectures with increasing complexity and specialisation, in order to comprehensively evaluate the efficacy and feasibility of deep learning for predicting intraday electricity prices. The first networks are the jack-of-all-trades type, followed by networks specialised for explicitly processing sequential data. To incorporate the weather data we use networks specialised for image processing, as part of a larger architecture that also utilises the market data to make predictions using all the available information.

5.3.1 A Brief Introduction to Deep Learning

Neural networks approximate the target function $y = f^*(\mathbf{x})$ as $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$ by learning the set of network parameters $\boldsymbol{\theta}$ (Goodfellow et al., 2016) — of which a neural network can have millions. Generally, such networks can be depicted as a directed flow of data from raw inputs to an output via a chain of functions that extract increasingly complex and abstract representations (Chollet, 2018). This chain consists of *hidden layers* that constitute the learned mapping f. Informally, the number of hidden layers defines the network's *depth*, which loosely determines the complexity of functions the network can compute. Each hidden layer l consists of a number of computational units, also called *neurons* or *nodes*, that transform input data into representations more useful for solving the given problem (Borovykh, Bohte, & Oosterlee, 2017). Each neuron accepts a vector of inputs $\mathbf{x}$ from the preceding layer — either $\mathbf{x}^{(l-1)}$ from the input layer or $\mathbf{h}^{(l-1)}$ from a hidden layer — and performs an affine transformation $z = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{w}$ and b are learned as constituent parts of $\boldsymbol{\theta}$ (Goodfellow et al., 2016). To map non-linear representations, a non-linear *activation function*[68] $g(z)$ is applied, such that $h = g(\mathbf{w}^\top \mathbf{x} + b)$. The resulting activation is then passed forward to the neurons in the next layer in the network until, ultimately, the *output layer* — in our case, a single node with no activation — takes the features $\mathbf{h} = f(\mathbf{x}; \boldsymbol{\theta})$ from the units

[68]Sometimes referred to as a *transfer function*.

in the final hidden layer and outputs a prediction $\hat{y}$ (Chollet, 2018).[69]

The network learns the parameters $\boldsymbol{\theta}$ through repeated exposure to the training data by minimising the provided *loss function* $C(\hat{\mathbf{y}}, \mathbf{y}) = C(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})$, which computes a distance score between the network's predictions $\hat{\mathbf{y}}$ and the true targets $\mathbf{y}$ (Goodfellow et al., 2016). Since all constituent functions are differentiable, the *gradient* ∇ of the loss with respect to the network parameters, $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$, can be calculated using the *chain rule*. An optimiser algorithm then descends the loss function by updating $\boldsymbol{\theta}$ accordingly.[70] While adding depth increases a network's capacity to map more abstract and, potentially, informative features, it also makes it more likely that the gradient will *vanish*[71] due to repeated multiplication as it is propagated through the network. This has been a persistent challenge with deep or complicated forms of neural networks, as it hinders training and causes performance to stagnate.

The generalist neural network

The quintessential deep learning model is arguably the *multi-layer perceptron* (MLP), often referred to as *fully-connected* networks (Goodfellow et al., 2016). This type of network is used frequently on a wide variety of problems, and has been found to perform well on day-ahead electricity price forecasting (Lago, De Ridder, & De Schutter, 2018; Lago, De Ridder, Vrancx, & De Schutter, 2018). Using a sliding window, such networks can incorporate a number of time steps of preceding input values to indirectly capture the temporal dimension when making predictions. The window length, or the *lookback*, defines how many time steps, or lags, that are included. An approach to mitigating vanishing gradients in *feed-forward* networks such as these is to add *residual connections*[72] (RC), which help signals propagate through deep networks (He, Zhang, Ren, & Sun, 2015). Another is applying *batch normalisation*[73] (BN), which has also been found to improve the performance of deep feed-forward networks (Ioffe & Szegedy, 2015).

Capturing the temporal dimension

While fully-connected networks treat time series as a single temporal point by collapsing time steps of the same features into separate features, *recurrent* networks[74] are specialised for sequential data, and can incorporate much longer sequences than would be practical otherwise. A recurrent network temporally generalises its mappings, meaning a given pattern will be recog-

[69]Since we are predicting a continuous numeric price, we do not need to apply a non-linear transformation in the output layer.

[70]The approach is therefore commonly referred to as *gradient descent*.

[71]Become infinitesimally small or explode (mathematically, to be clear) from repeated multiplication.

[72]Details are provided in Appendix A.1.5.

[73]Details are provided in Appendix A.1.6.

[74]Details are provided in Appendix A.2.

nised no matter where in the sequence it appears, rather than having to appear in the exact same place as before (Chollet, 2018). *Long Short-Term Memory*[75] (LSTM) is one attempt at mitigating vanishing gradients[76] in recurrent networks by introducing self-loops that facilitate propagation of the gradient in very long sequences, where these loops are contextual rather than fixed (Goodfellow et al., 2016). An extension is using *bidirectional* layers[77], which also process input sequences in the reverse temporal order. These significantly improve performance on a range of problems where the full sequence is necessary to disambiguate signals, such as speech recognition (Graves, Jaitly, & Mohamed, 2013). Our market data is a strictly causal economic time series, but bidirectional LSTM networks have performed surprisingly well on other seemingly ill-suited problems, such as traffic speed prediction (Cui, Ke, & Wang, 2018).

Processing the weather forecast images

The simplest approach to processing the weather forecast images would arguably be to use a fully-connected network that flattens each image into a vector and treats each of the $15,125$ pixels as distinct parameters. As such, it will only recognise a spatial pattern if it appears in the same part of the map as before — like with a temporal dimension (Chollet, 2018).[78] An alternative is using *convolutional* networks[79], which are specialised for grid-like data such as 2-dimensional images[80] (Goodfellow et al., 2016). These learn to recognise spatial patterns no matter where they appear. In addition to this *translation invariance*, they also learn spatial hierarchies of patterns, from initially small, local patterns to larger, more complex patterns made up of the underlying ones (Chollet, 2018). For example, it can learn to recognise a car in an image as a combination of certain edges, textures, and shapes. Presumably, similar spatial hierarchies also apply to weather patterns in forecast images.

The final deep learning models

Section 5.3.4 elucidates the design and implementation of the two final windowed fully-connected networks with residual connections and batch normalisation. The two final recurrent networks

[75]Details are provided in Appendix D.3.
[76]Details are provided in Appendix D.3.
[77]Details are provided in Appendix A.2.1.
[78]Visual recognition problems are usually translation invariant in nature (Chollet, 2018), hence fully-connected networks tend to perform relatively poorly. Weather forecasts might be an exception, as a weather pattern's location is presumably more important than the fact that the pattern appears; e.g. heavy rainfall over parts of Norway with significant hydro-power production is likely more consequential than deep in the Barents Sea. Nevertheless, our experiments showed convolutional networks to generalise better on the weather data, and at lower computational cost. As such, we use these.
[79]Details are provided in Appendix A.3.
[80]These can also be used with 1-dimensional time series, but after extensive experimentation we found these to perform poorly on our problem.

are covered in Section 5.3.5. Before an amalgamating architecture that combines market and weather data can be implemented, we first need a way to capture the temporal structure of the weather forecasts. This is covered in Section 5.3.6, followed by the resulting final amalgamating network in Section 5.3.7. The most significant computational cost by far lies in training these networks, where the time needed to do so varies greatly; fitting one of the residual networks takes approximately 17 minutes, one of the LSTMs takes 2.5 hours, and one of the final amalgamating architectures takes 5.5 hours. Although training one of each final architecture can be done in a day, trying different variations and configurations can be prohibitively time-consuming[81].

Technical implementation and requirements

All data processing and neural network building was done in Python v3.6 via Jupyter Notebooks — the most important of which are available in our public Github repository[82]. Data pre-processing was largely performed using the open-source Pandas v0.22.0 and Numpy v1.12.1 libraries. Neural networks were implemented in the open-source Keras v2.1.3 library[83], using the TensorFlow v1.1.0 backend on a *Graphical Processing Unit* (GPU) with NVIDIA's CUDA v8.0 toolkit and cuDNN v6.0 library. Keras reduces the technical expertise required to implement deep learning considerably, but there is still some programming knowledge needed — not to mention competence in machine learning. Networks were trained and evaluated on a desktop computer with a GTX 980 Ti GPU with 6GB VRAM, an Intel i7 4790k CPU, and 16GB of RAM. Though originally designed for computer games, GPUs are fortuitously excellent at the matrix multiplications involved in fitting neural networks. As such, they are generally much better suited to deep learning than CPUs (Chollet, 2018); indeed, our scripts ran more than an order of magnitude faster on our GPU. Hence, we would argue that a desktop computer with a relatively powerful GPU and at least a modicum of technical expertise are minimum requirements for implementing deep learning projects in practice.

5.3.2 Designing Network Architectures

Most machine learning methods involve one or more *hyperparameters* that control some aspect of how the method is applied (Hastie et al., 2009). Ideally, one would select their values empirically via an exhaustive evaluation of how various configurations affect model performance on

[81]Not to mention that the desktop computer used to do so is an awful roommate when it's spewing out hot air at 80 degrees Celsius and running five fans at over 1,000 RPM.

[82]Publicly available at: `https://github.com/johannes-kk/Elbas-Deep-Learning`

[83]There are many alternative deep learning frameworks, such as MXNet, Caffe, PyTorch, or using TensorFlow directly, but these are generally more technically involved and, arguably, unnecessary for our purposes.

a *validation set*, or via a *resampling* technique. However, such a *grid search* quickly becomes prohibitively expensive as the number of parameters increases (Chollet, 2018). Even more refined versions, such as trying a random subset of possible configurations via a *random sweep* or using *genetic algorithms*, can also be impractical for the same reason. In addition to having notoriously many hyperparameters to tune, deep learning also require that one defines the network's *topology*, i.e. the types, ordering, and configuration of the functional components that comprise the network (Goodfellow et al., 2016). We refer to the combination of a network's hyperparameter values and topology as its *architecture*.

Though the same automated approaches can be used to designing architectures as for optimising hyperparameters, these can be extremely expensive to run, especially due to the high computational cost of fitting neural networks (Chollet, 2018). We therefore instead use an iterative approach that combines experimentation and inspiration from other research to pursue a more guided path through the space of possible architectures. Networks will generally overfit to the training data, hence we evaluate them on the validation set and use this performance to select architectures for further development. Like an automated approach would, we are therefore indirectly overfitting to the validation data. Hence, after designing the final set of architectures, we evaluate these on the test set to get a more realistic estimate of their performance on unseen data. Our experimentation resulted in hundreds of candidate architectures evaluated on the validation set, but in the interest of brevity we only include the final architectures here.

Design choices common for all architectures

We make a set of design choices common to all architectures we evaluate, both to reduce the space of possible architectures to try and to make our approach more internally consistent. These are rooted in a combination of findings from research and our own empirical results:

- *Loss function*: The networks minimise MAE on the training set. This choice over alternatives like MSE is explained in Section 5.1.
- *Activation function*[84]: Where relevant, we use *Rectified Linear Units* (ReLU). These are a good default choice, and mitigate vanishing gradients (Goodfellow et al., 2016).
- *Mini-batch size*[85]: After some experimentation, we use 128 as an effective compromise between accurate estimates of the gradient and hardware memory use[86].

[84]Details are provided in Appendix A.1.1.
[85]Details are provided in Appendix A.1.3.
[86]Multiples of 2 fit better onto CPU or GPU cache since these are binary. We tested 64, 128, and 256.

- *Optimiser algorithm*[87]: ADAM is used with default values, as these are generally effective (Goodfellow et al., 2016). We also tested RMSprop and NADAM[88], but found them to be inferior.
- *Learning rate schedules*[89]: To help networks escape local minima during training, we specify that the learning rate should be halved when training loss plateaus. This is triggered upon no improvement in three or five epochs, depending on the total number of epochs.
- *Regularisation*[90]: Dropout has been found to outperform alternative cheap regularisation methods, can be combined with other methods, and does not limit the possible architectures (Goodfellow et al., 2016). As such, we use it extensively in our networks. Experiments with norm penalties suggested that they constrain training performance without justifiable improvements on the validation set, and are therefore not used.
- *Epochs*[91]: Depending on each architecture's general pattern of convergence and time needed per round of exposure to the training data, the number of epochs is set to either 30 or 100, after some experimentation.

5.3.3 Evaluating the Final Architectures

One popular approach to fighting overfitting is the *minimum validation loss* (MVL) model selection method, where the algorithm records the network parameters whenever validation loss improves, and picks those that generalised best rather than those from the last update (H. Chen, Lundberg, & Lee, 2017). Another popular approach is using *ensembling* methods, where different models are fit and combined to make predictions that are, usually, and on the whole, more accurate than those of any of its constituents (Dietterich, 2000). The simplest method is arguably *random initialisation ensembling* (RIE), where the same architecture is used to fit multiple models whose predictions are averaged. However, even this can be computationally prohibitive due to the cost of training large neural networks on comprehensive datasets. *Checkpoint ensembling* (CE) is an inexpensive way to capture some of the generalisation gains from ensembling without the need to fit multiple networks; rather than picking the best weights, CE combines the predictions from the best *checkpoints* of weights during training, as measured on

[87]Details are provided in Appendix A.1.3.
[88]ADAM with Nesterov momentum. See Goodfellow et al. (2016) for more details.
[89]Details are provided in Appendix A.1.3.
[90]Details are provided in Appendix A.1.4.
[91]Details are provided in Appendix A.1.2.

the validation set (H. Chen et al., 2017). The model corresponding to the weights at the end of various successful epochs may be particularly adroit in its respective subspace of prediction problems. By combining the best of these checkpoints, one might get a model that generalises better to the complete space of all possible prediction problems (H. Chen et al., 2017).

Consequently, for each of our final architectures we use the MVL method to extract the best weights and evaluate the resulting performance, but also use CE to try to improve performance further. Following He, Zhang, Ren, and Sun (2016)'s example, we initialise and run each architecture five times, and report the median for each relevant metric. Although it is computationally costly, we get more reliable estimates of each architecture's performance and robustness by assessing five models' performance using MVL and CE. Since we are fitting five models for each architecture, we also perform RIE by averaging predictions across the five models using their respective minimum validation loss weights.

5.3.4 Windowed Market Models: Deep Residual Networks

With inspiration from He et al. (2016), we design a deep multi-layer perceptron (MLP) with residual connections between consecutive stacks, or blocks, of three hidden layers. After extensive experimentation, we found such networks to perform better by also adding residual connections from the original flattened input data where there were such connections between stacks. Residual connections are added together rather than concatenated, per He et al. (2016). As experimentation showed that networks with residual connections clearly outperformed those without, we have no version without them.

These networks only use the market data. They use a sliding window of input data to capture the temporal dimension, meaning *lookback* lags of each input variable are included. Since MLPs only accept a vector of input data for each sample in the mini-batch, the input layer *flattens* the temporally multi-dimensional input data. Fully-connected networks treat these combinations of lag and variable as individual features, but should generalise with sufficient data and training. Experiments showed that window lookback lengths of between 6 and 18 hours worked best, so we use 12 hours. The architecture consists of six blocks of three hidden layers, for a total of eighteen hidden layers. The first block uses 512 nodes with 50% dropout at the end of the block, the following two use 256 with 45%, the two after that use 128 with 40%, and the last uses 64 with 30%. Adding significantly more depth did not seem to improve performance. Hence, we stop at eighteen layers. The output layer is, as always, a single node with no activation.

We make two versions of the architecture: one with some additions and adjustments based on experimentation, and one that is more faithful to He et al. (2016)'s *pre-activation* residual blocks. The first architecture, *ResNet 1*, applies ReLU after each hidden layer, with batch normalisation applied at the end of the block prior to adding residual connections, and dropout after. Figure 12 illustrates the *ResNet 1* architecture. The second architecture, *ResNet 2*, more closely resembles the residual block proposed by He et al. (2016); batch normalisation and ReLU are applied, in that order, *before* each hidden layer, with dropout applied after residual connections are added at the end of each block. Figure 13 visualises *ResNet 2*. For both architectures, exhaustive model topologies are presented in Appendix D.1 and Appendix D.2, respectively.

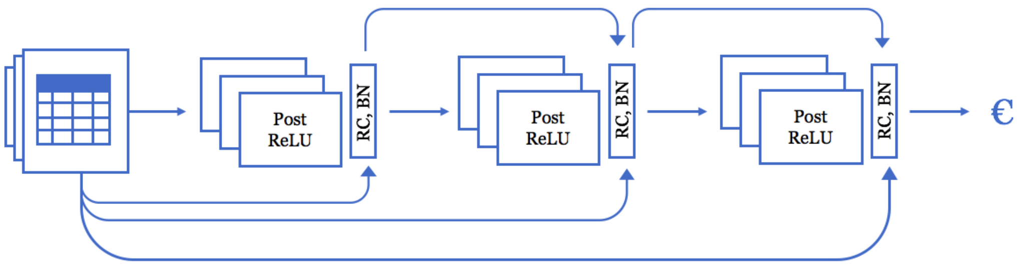


Figure 12: *ResNet 1* network architecture. Uses the market data with a sliding window of 13 hours (including the delivery hour in question) of input values. Three residual blocks, or stacks, are shown for readability, but the full architecture uses six. Each block consists of three equally wide hidden fully-connected feed-forward layers, with ReLU activations applied *after* each layer. Batch normalisation (BN) is applied at the end of each block, followed by *adding* any relevant residual connections (RC), indicated by arrows, and dropout. The first block uses 512 nodes with 50% dropout at the end of the block, the following two use 256 with 45%, the two after that use 128 with 40%, and the last uses 64 with 30%.

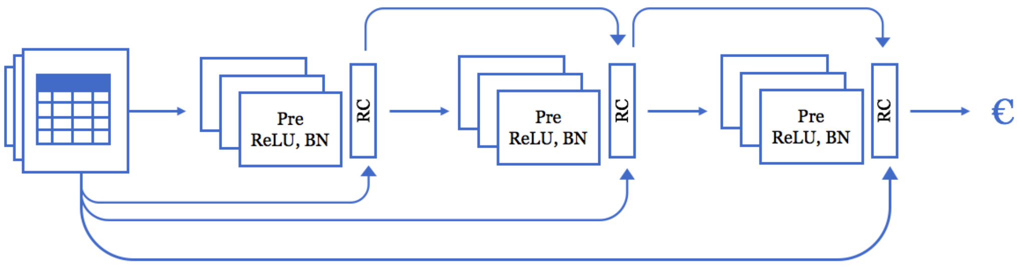


Figure 13: *ResNet 2* network architecture. Uses the market data with a sliding window of 13 hours. Three residual blocks, or stacks, are shown for readability, but the full architecture uses six. The architecture is the same as *ResNet 1*, except that ReLU and batch normalisation (BN) are applied *before* each hidden fully-connected feed-forward layer. Residual connections (RC) and dropout are applied, in that order, at the end of each block.

5.3.5 Sequential Market Models: LSTM Networks

We design two LSTM recurrent networks that only use the market data. Some of our experimentation with recurrent architectures is summarised in Table 13 in Appendix D.7. While LSTMs have seen tremendous success on state-of-the-art problems (Goodfellow et al., 2016), there have been attempts to simplify it by dropping unnecessary components or redesigning it altogether. One of these is the *gated recurrent units* (GRU), which is theoretically less computationally costly, but also has less representational power (Chollet, 2018). As such, we first experiment with various architectures using LSTM or GRU to select which is best suited for our problem, and conclude much the same; as expected, LSTM outperforms GRU at a higher computational cost, but we incur this extra time to get models that are in fact better. We found negligible performance gains from going significantly deeper than two or three hidden layers. The recurrent networks consistently perform better the longer the window is, but these longer sequences also come at proportionately higher cost, so there is a practical upper limit. The first recurrent architecture, referred to simply as the *LSTM*, therefore uses a window size of 21 days, i.e. 505 time steps (including the corresponding delivery hour). It consists of *one* hidden layer of 512 LSTM nodes with 40% dropout on both the recurrent connections and the activations. This architecture is illustrated in Figure 14, while Appendix D.3 contains the exhaustive model topology.

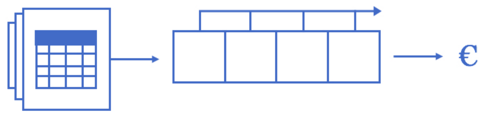


Figure 14: *LSTM* network architecture. Uses the market data with an input sequence length of 505 hours (21 days). Consists of *one* hidden layer of 512 LSTM nodes with 40% dropout on both the recurrent connections and activations.

The second recurrent architecture, the *BD-LSTM*, is a bidirectional LSTM, which combines a traditional recurrent network that steps chronologically through time with another that steps in the opposite direction, illustrated in Figure 15. The exhaustive model topology is included in Appendix D.4. We try bidirectional recurrent networks in case they pick up some valuable patterns by also evaluating the inputs in the reverse chronological order. In our case, this means stepping backwards from time τ to 1 rather than evaluating the complete temporal sequence, as that would incorrectly incorporate information from the future. Although a shorter sequence negates part of the benefit of LSTMs, namely to capture long-term dependencies, the network uses a window of 7 days, i.e. 169 time steps. This is largely a practical consideration due to the computational cost of bidirectional networks, and because experimentation showed deteriorating performance when the sequences grew much longer — presumably because, as the ideal length for our residual networks suggests, the most important information might be contained in the most recent observations, hence the reversed order going further back than this might only be adding noise. It consists of a bidirectional LSTM layer (i.e. two layers) with 512 nodes followed by a standard LSTM layer with 256. All layers use 40% dropout on both the recurrent connections and the activations.

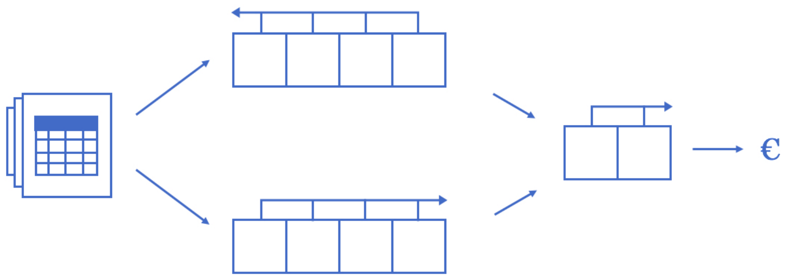


Figure 15: *BD-LSTM* network architecture. Uses a window of 169 hours (7 days). Consists of a bidirectional LSTM layer with 512 nodes, followed by a regular LSTM layer with 256 nodes. All use 40% dropout on both the activations and recurrent connections.

5.3.6 Processing Spatio-Temporal Weather Data

Each weather forecast is a five-channel geospatial image processing problem, but we have sequences of such forecasts. To properly incorporate this temporal dimension, we can use a *time-distributed* convolutional pyramid[92]. Time-distributed layers repeat a given operation for each temporal slice of the input[93], i.e. learn a shared set of parameters used to compute a separate output for each time step. We therefore convolve over each forecast as usual, but maintain the time steps which can then be picked up by a mechanism specialised to sequences — such as recurrent layers. An alternative is using 3-dimensional convolutions, which in this case convolve over one temporal and two spatial dimensions. This has the downside of being translation-invariant in the temporal dimension, which would likely fail to capture the relative importance of more recent observations — or vice versa. We nevertheless tested such architectures, but

[92]Details are provided in Appendix A.3.1.

[93]See the Keras documentation: `https://keras.io/layers/wrappers/#timedistributed`

found them to perform poorly and have an unjustifiably high computational cost.

The parts of our neural networks designed to process weather forecast images are therefore time-distributed convolutional pyramids where we, after extensive experimentation, use convolutional kernels of dimensions 3×3 with a stride[94] of 1 and ReLU activations. We use *valid* padding[95] in Keras, which does not pad with zeroes, but rather restricts the kernel to traversing only within the image. Each convolution layer is followed by a max pooling layer with a 2×2 kernel and stride 2. These layers are all time-distributed, so that the temporal dimension is maintained by applying the convolutional pyramid separately to each time step of forecasts. This architecture does not make much sense to run by itself, but is used as a component of a more involved architecture that combines market and weather data.

5.3.7 Market & Weather Model: Multi-Input Network

In order to create neural networks that utilise all of the available data, we design an amalgamating multi-input architecture, referred to as the *Multi LSTM ResNet*, that leverages both the temporal structure of the market data and the spatio-temporal structure of the weather data.[96] Compared to the 328 features per sample of market data, a single five-channel weather forecast image contains 15,125 parameters. Including additional time steps of forecasts is therefore much more computationally costly than with the market data. Since longer windows of market data seem advantageous, particularly with recurrent networks, we design the architecture so that it accepts two different sequence lengths. This architecture consists of three *pipes*, or distinct sequences of hidden layers: one each for the market and weather data, and one that combines their outputs to process the high-level features and make a final prediction.

Designing such a network is more involved, as we must coerce two different spatio-temporal structures to conform so that they can be used as a combined input. A simple approach is to make a pipe that outputs a set of features without a temporal component. This could be either to flatten each pipe's output to collapse the temporal dimension — which could erase potentially

[94]The *stride* is the number of steps the convolution takes when extracting the next patch. With no stride (stride of 1), successive receptive fields will overlap somewhat, which is usually desirable when searching for local patterns.

[95]The kernel, or receptive *field* covers an area that might extend outside the image when near the edges. We therefore either need to restrict the kernel so that it stays within the image, or use some method of adding a border of zeroes around the edges so that the kernel activation is not mathematically undefined due to undefined values.

[96]The more exhaustive and arguably correct name would be: Multi-input time-distributed spatial convolutional pyramid long short-term memory residually connected network.

valuable temporal relationships — or to finish each with a recurrent layer. We have used both approaches when designing networks that only take in market data, but choose to do only the latter when designing the final amalgamating architecture. This is both a result of the findings in Section 6.1.2, and an educated simplification of the space of possible architectures.

After comprehensive experimentation, we design an amalgamating network architecture that combines recurrent layers to consolidate the temporal dimensions, a time-distributed spatial convolutional pyramid, and a residual network to capture non-linear combinations of the resulting discrete spatio-temporal high level market and weather features. Figure 16 illustrates this architecture, and Appendix D.5 presents the exhaustive model topology.

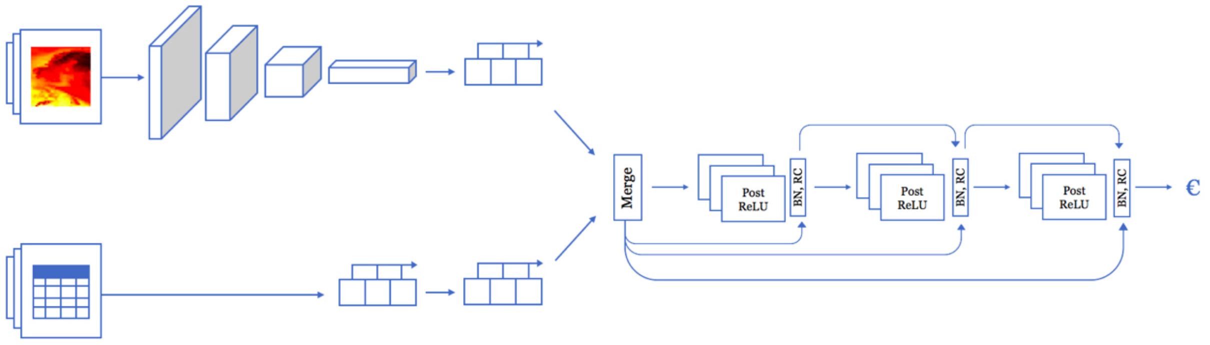


Figure 16: *Multi LSTM ResNet* network architecture. The top pipe takes in sequences of 7 weather forecast image sets, and consists of a convolutional pyramid with four rounds of time-distributed spatial 3×3 convolutions and 2×2 max pooling from 32 to 256 filters, followed by an LSTM layer with 64 nodes, and 50% dropout on the activations and recurrent connections. The bottom pipe takes in sequences of 505 hours (21 days) of market data, and consists of two stacked LSTM layers, both with 256 nodes and 50% dropout on the activations and recurrent connections. The two pipes' outputs are merged via *concatenation*, and the resulting 320 features processed by a sequence of three residual blocks of respectively 256, 128, and 64 units with respectively 50%, 40%, and 30% dropout. Each consists of three fully-connected layers with ReLU activations. Batch normalisation is applied at the end of each block, followed by *adding* residual connections, and dropout.

The market pipe uses a sequence length of 21 days and consists of two stacked recurrent LSTM layers with 256 nodes and 50% dropout on the recurrent connections and activations. The weather pipe uses a window length of 6 hours, and consists of a convolutional pyramid with four rounds of time-distributed spatial 3×3 convolutions and 2×2 max pooling from 32 to 256 filters, followed by an LSTM layer with 64 nodes and 50% dropouts. The resulting outputs are concatenated, resulting in a one-dimensional vector of 320 features, and processed by a sequence of three residual blocks of respectively 256, 128, and 64 units. Each consists of three fully-connected layers with ReLU activations and post-activation batch-normalisation, and a dropout amount that decreases from 50% to 30%. The final layer is, as usual, a fully-connected layer with one node and no activation function, which makes the corresponding prediction of volume-weighted price.

6 Analysis and Results

Section 6.1 assesses the overall respective performances of the non deep learning benchmark approaches and the deep learning models, before Section 6.2 evaluates performance in greater detail by considering how the models perform across delivery hours, trading activity, price levels, and in detecting unusually high or low prices. In this way, we can evaluate potential strengths or weaknesses among the models that are not revealed on an aggregate level. Finally, Section 6.3 wraps up this chapter by gauging to what extent deep learning can predict the *near* volume-weighted average Elbas price six hours ahead of a given hour of power delivery, and how reliable our forecasts are in practice.

6.1 Model Performance Results

This section first presents the results from using approaches which do not apply deep learning techniques when predicting the *near* VWP (Section 6.1.1). They serve as benchmarks against which the performance of the deep learning models can be judged. Then, Section 6.1.2 assesses the performance of each of the five final deep learning models.

6.1.1 Benchmark Models

Simple heuristics

We begin by considering the simple heuristics, which are the benchmarks that only use one historic price variable to predict the *near* Elbas VWP in a given delivery hour. Table 3 summarises the performance of each of these six simple approaches, in order of ascending MAE on the validation set. Predicting the *near* VWP to be the same as the spot price in the SE3 bidding area is the simple benchmark approach with the lowest validation set MAE of 2.96 EUR/MWh. In comparison, the system price baseline achieves an MAE of 4.15 EUR/MWh, and the baselines using available Elbas price information have at best an MAE of 4.81 EUR/MWh through the *far* VWP baseline. The SE3 heuristic also obtains the lowest MAE on the test set, 3.43 EUR/MWh. Though the corresponding RMSE values are somewhat higher than the MAE values, 5.94 EUR/MWh for the validation set and 5.65 EUR/MWh for the test set, the SE3 baseline is still the one that results in the lowest RMSE among the simple benchmark models.

That the simple heuristic of predicting the *near* Elbas VWP to be the SE3 price achieves an MAE as low as 2.96 EUR/MWh, indicates that a large share of the VWP may be explained by the day-ahead spot prices. This is not unreasonable as the spot prices represent the market situation after the day-ahead settlements. If there are few deviations in the intraday market, one would expect prices to be on the same level. Additionally, the spot prices probably also send a strong price signal to market participants trading in Elbas. However, the correlation between the SE3 spot price and the *near* VWP is stronger during both the training and validation set period than during the test set period, hence the MAE on the test set is higher than on the validation set.[97]

Simple Benchmark	**Validation**		**Test**	
	MAE	**RMSE**	**MAE**	**RMSE**
Predicting spot price in SE3 of corresponding delivery hour	2.9555	5.9388	3.4278	5.6467
Predicting system price of corresponding delivery hour	4.1459	7.5390	4.4676	7.0356
Predicting *far* VWP from the corresponding delivery hour	4.8094	7.5873	4.0038	6.2014
Predicting VWP of corresponding delivery hour of previous day	5.3167	10.1989	6.0267	9.0201
Predicting 6-hour lagged VWP (delivery hour 6 hours prior)	7.2566	12.2718	7.1669	10.2758
Predicting mean VWP from training set	9.0018	12.6057	6.6829	9.2645

Table 3: *Mean absolute error* (MAE) and *root mean squared error* (RMSE) of simple heuristics baselines, in order of ascending MAE on the validation set. All values are in EUR/MWh.

Econometrics and simpler machine learning

The performances of the more advanced benchmarks are summarised in Table 4. For each distinct approach, only the top performer, based on lowest MAE on the validation set, is included.[98] Not surprisingly, all of the more advanced benchmarks outperform the best simple heuristic that

[97]The correlation between the SE3 spot price and the *near* Elbas VWP is 0.86 during the training set period, 0.88 during the validation set period, and 0.78 during the test set period.

[98]For example, the elastic net models with other values of α than 0.2 are left out.

predicts the VWP to be the SE3 spot price. The best model, measured in terms of MAE on the validation set, is the non-linear gradient boosting model, which obtains a validation set MAE of 2.69 EUR/MWh. Its test set MAE is 3.04 EUR/MWh. Again, we observe that the test set MAE exceeds the validation set MAE. This is not surprising as we tuned hyperparameters so as to minimise the MAE on the validation — and not the test — set. However, it may also partly be due to the fact that the correlation between the *near* VWP and the SE3 spot price is lower during the test set period than the training and validation set period. As the SE3 spot price is consistently identified as an important variable in the models due to its strong correlation with the *near* VWP, we would expect the models to predict with slightly higher errors when the correlation between these two variables is somewhat weaker.

Model	**Validation**		**Test**	
	MAE	**RMSE**	**MAE**	**RMSE**
Gradient boosting, N of trees = 1,000	2.6931	4.9436	3.0388	4.8079
MARS feature selection, linear and polynomial terms	2.7282	4.9111	3.0235	4.7522
Elastic net, alpha = 0.2, lambda = 0.2553	2.7631	5.0894	3.0944	4.8061
LASSO, lambda = 0.0511	2.7642	5.0923	3.0845	4.7962
Ridge regression, lambda = 1.2357	2.8302	5.0974	3.1962	4.8662
RF feature selection, N of trees = 1,000, linear terms	2.8319	5.3848	3.1301	4.9216
Principal component regression, N of components = 115	2.8331	5.2064	3.2853	5.0119
Linear regression, Elspot and prior Elbas prices	2.8531	5.3772	3.0836	4.8840

Table 4: *Mean absolute error* (MAE) and *root mean squared error* (RMSE) of econometrics and simpler machine learning baselines, in order of ascending MAE on the validation set. All values are in EUR/MWh.

The second best performance is achieved through feature selection using MARS, combined with fitting a linear model with both linear and polynomial terms. Here, the MAE is approximately 2.73 EUR/MWh on the validation set, which is only slightly above the MAE for gradient boosting. On the test set, the model obtains a marginally lower MAE than gradient boosting, at 3.02 EUR/MWh. The gradient boosting and the model employing MARS feature selection also perform better in terms of RMSE than the other models.[99] As the remaining models all yield higher errors on both the validation and test sets, we conclude that models allowing for non-linear relationships achieve the best performances, which in turn is promising for deep learning.

6.1.2 Deep Learning Models

So far we have seen that it is possible to predict the *near* Elbas VWP with relatively good accuracy six hours ahead of the delivery hour. Furthermore, simply predicting the *near* VWP to be the spot price in SE3 gave an MAE of 2.96 EUR/MWh on the validation set, while applying gradient boosting improved the MAE to 2.69 EUR/MWh. The question then is whether deep learning can improve this further.

We recall from Section 5.3 that the two final ResNet models consist of one where ReLU is applied after each layer while batch normalisation is only applied after each stack of three layers (*ResNet 1*), and one where ReLU and batch normalisation are applied, in that order, before each layer (*ResNet 2*). The two final LSTM models differ in that one is a bidirectional network (*BD-LSTM*), while the other (*LSTM*) only incorporates the original temporal structure. Finally, the *Multi LSTM ResNet* integrates weather data with market data. Table 5 and Table 6 summarise the performances of each of these final models on the validation and test sets, respectively. Each table presents both the MAE and RMSE using *minimum validation loss* (MVL), *checkpoint ensembling* (CE) and *random initialisation ensembling* (RIE). The models were run five times each, but in the tables, only the median performance per column is presented for MVL and CE. Appendix D contains the results per run for each of the five models. RIE, on the other hand, is an ensemble of the MVLs from each of the five runs.

Some findings are common across all models. First, comparing the models' results on the validation set to those on the test set, there are signs that we have been indirectly overfitting to the validation data. As with the benchmark models, all MAEs and RMSEs on the test set are consistently higher than their counterparts on the validation set, with some differences being rather stark; some MAEs are nearly 0.50 EUR/MWh higher. However, this may also be partly due to the aforementioned weaker correlation between the *near* VWP and the SE3 spot price in the test set. Second, all deep learning models do in fact achieve a lower MAE than the simpler benchmarks on both the validation and test sets. Considering the MVL MAE values, they obtain

[99]Except the elastic net and LASSO, that obtain marginally lower RMSE values on the test set than gradient boosting.

Model	**Validation**					
	MVL		**CE**		**RIE**	
	MAE	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**
ResNet 1	2.3738	4.4474	2.3636	4.4267	2.3390	4.4135
Resnet 2	2.4089	4.4930	2.3549	4.4284	2.3411	4.4246
LSTM	2.3553	3.4757	2.2475	3.3728	2.2464	3.3544
BD-LSTM	2.5423	3.9569	2.4168	3.8373	2.3551	3.7617
Multi LSTM ResNet	2.3383	3.5277	2.2854	3.4525	2.2482	3.3856

Table 5: *Mean absolute error* (MAE) and *root mean squared error* (RMSE) of the deep learning models (median run per column) on the validation set. MVL is *minimum validation loss*, CE is *checkpoint ensembling*, and RIE is *random initialisation ensembling.* All values are in EUR/MWh.

Model	**Test**					
	MVL		**CE**		**RIE**	
	MAE	**RMSE**	**MAE**	**RMSE**	**MAE**	**RMSE**
ResNet 1	2.7082	4.7409	2.6922	4.7222	2.6789	4.6917
ResNet 2	2.7029	4.7459	2.6776	4.7074	2.6567	4.6956
LSTM	2.8278	4.3563	2.7414	4.2803	2.7131	4.2377
BD-LSTM	2.8510	4.3781	2.7439	4.3147	2.6983	4.2964
Multi LSTM ResNet	2.7824	4.3390	2.7359	4.2672	2.7242	4.2443

Table 6: *Mean absolute error* (MAE) and *root mean squared error* (RMSE) of the deep learning models (median run per column) on the test set. MVL is *minimum validation loss*, CE is *checkpoint ensembling*, and RIE is *random initialisation ensembling.* All values are in EUR/MWh.

a margin of between 0.15–0.35 EUR/MWh on the validation set and 0.17–0.32 EUR/MWh on the test set. The CE MAE improves this margin to 0.27–0.44 and 0.28–0.34 respectively, which is again marginally improved using RIE to 0.33–0.44 (validation) and 0.30–0.36 (test). Still, the deep learning models are predicting the *near* VWP with, at best, an average absolute error of 2.25 EUR/MWh on the validation set and 2.66 EUR/MWh on the test set.

Third, there seem to be inklings of consistent improvements from ensembling. In particular, there is no doubt that ensembling improves performance for the LSTM models, especially on unseen data. Overall, the largest enhancement tends to come from CE, while RIE only provides slightly better results. Still, these early signs of the impact of ensembling could, perhaps, warrant extending to more sophisticated forms of ensembling, or ensembling altogether very different models. Finally, the MAE values are considerably lower than the RMSE values on both the validation and test sets, which is not surprising as the models were trained with the objective of minimising MAE, and not RMSE. Hence, we allow the models to miss more when they first miss, as it is not punished as hard as it would have been had RMSE been used instead.

From Table 5, we conclude that the *LSTM* and *Multi LSTM ResNet* models perform better than the other three, reaching RIE MAE values of 2.25 EUR/MWh and RIE RMSE values of about 3.35–3.39 EUR/MWh. The RIE MAE values of the other models are about 0.1 EUR/MWh higher, while the RIE RMSE values are around 0.4 EUR/MWh higher for the *BD-LSTM* model and more than 1 EUR/MWh higher for the ResNet models. The considerable reductions in RMSE are not surprising, as the ResNet models do not have as much context with which to predict extreme spikes, while the models with LSTM units do as they incorporate much longer sequences and may learn patterns regardless of where in the sequence they appear. There are, however, only small differences between *LSTM* and *Multi LSTM ResNet*, and it is not entirely clear which is better than the other. Hence, the inclusion of the weather forecast images does not seem to yield noticeable improvements in performance. One should, however, keep in mind that the time to train the weather models were limited, and that we only were able to include one weather forecast per hour — thus, limiting the extent to which forecast *errors* are represented in the models.[100]

Inspecting the results for the test set in Table 6, a different picture of the models' performances emerges. Here, the ResNet models consistently achieve lower MAE values than the LSTM models — albeit at the cost of higher RMSE values, in terms of which the LSTM models still outperforms the ResNet models. *Multi LSTM ResNet* also has a higher MAE than the ResNet models. Furthermore, it can be noted that, in terms of MAE, *ResNet 2* achieves marginally lower errors than *ResNet 1* on the test set. In contrast, *ResNet 1* generally performed better than *ResNet 2* on the validation set (Table 5). *BD-LSTM* also improves its performance compared to *LSTM*, in fact obtaining slightly lower RIE MAE than the latter. For the in-depth analysis, we want to include one model from each of the three distinct architectures. However, the best models should be identified from their validation set performances, as the purpose of the test set is to evaluate the models' performances on unseen data. Despite the improved performances of *ResNet 2* and *BD-LSTM* on the test set compared to the validation set, we therefore move on with the *ResNet 1*, *LSTM*, and *Multi LSTM ResNet* models for the in-depth analysis.

[100]Forecast errors may be more important in explaining intraday trading and prices than the current forecasts in isolation (Chapter 3).

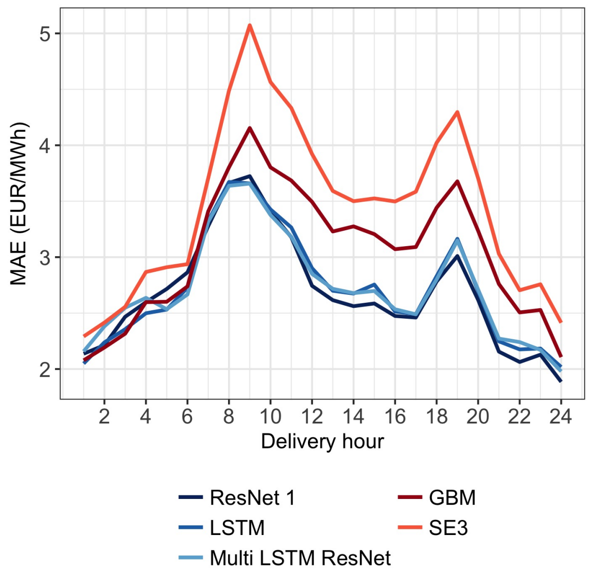
6.2 In-Depth Analysis of Model Performance

So far, we have seen that the deep learning models outperform the simpler benchmarks in terms of both MAE and RMSE. However, these are aggregate measures. A model that is stable in most situations may be preferred to one that performs extremely well in some and quite poorly in others. This is especially relevant since Elbas trading for the different delivery hours may vary considerably in terms of trading activity and price (Section 4.5). For this reason, we will now examine the performance of the models based on the delivery hour in question (Section 6.2.1), the trading activity for the delivery hour in question (Section 6.2.2), and the level of the true *near* Elbas VWP in the delivery hour in question (Section 6.2.3). As unusually high or low price levels are found to potentially cause challenges for electricity price models (Botnen Holm, 2017; Lu, Dong, & Li, 2005; Voronin, 2013) — and as Section 6.2.3 will show, our models are no exceptions — Section 6.2.4 finally evaluates to what extent the different models may predict that a price will rise above or fall below certain limits, though potentially failing to predict the precise VWP. All analyses are conducted on the test set to evaluate the models' performances on unseen data.

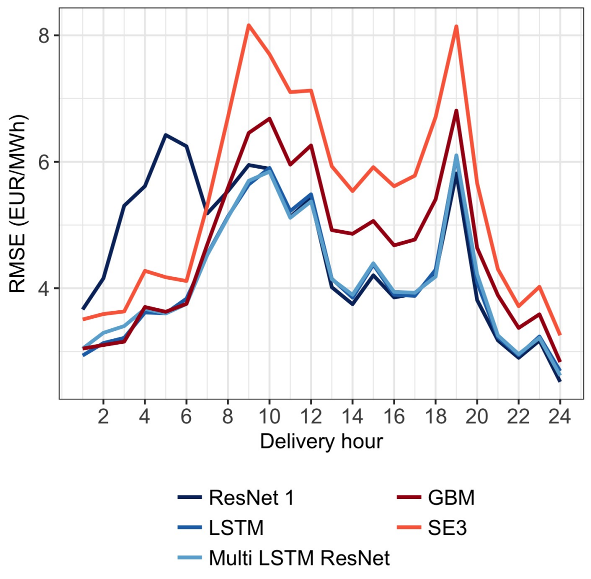
For the deep learning models, we evaluate the performance in this section using the RIE values, as this ensembling approach is found to yield the best results. The best simple heuristic — the SE3 spot price — and the gradient boosting benchmark model (GBM) are also included in the following analyses so as to evaluate whether deep learning is in fact consistently better than the benchmarks. Gradient boosting is selected as the best candidate among the more advanced benchmark models, as it obtained the lowest MAE on the validation set. In the figures in the following sections, the benchmarks' performances are displayed in nuances of red, while nuances of blue are used for the deep learning models.

6.2.1 Performance across Delivery Hours

Different delivery hours may be characterised by different market conditions, for example due to variations in the number of active participants, the level of available intraday capacities, and potentially the extent to which participants may exercise market power. The plots in Figure 17 reveal how the performances of the deep learning models are not even over the course of the day, but vary depending on the delivery hour in question. In particular, it seems to be the case that the hours over which the *near* VWP exhibits higher volatility (Section 4.5.2) are characterised by higher prediction errors. Here, all models exhibit the same pattern. Specifically, the MAE and RMSE peak around delivery hours 8–9, and a new peak around hours 18–19. Recall that these were also the hours when the *near* VWP volatility peaked, as well.



(a) MAE



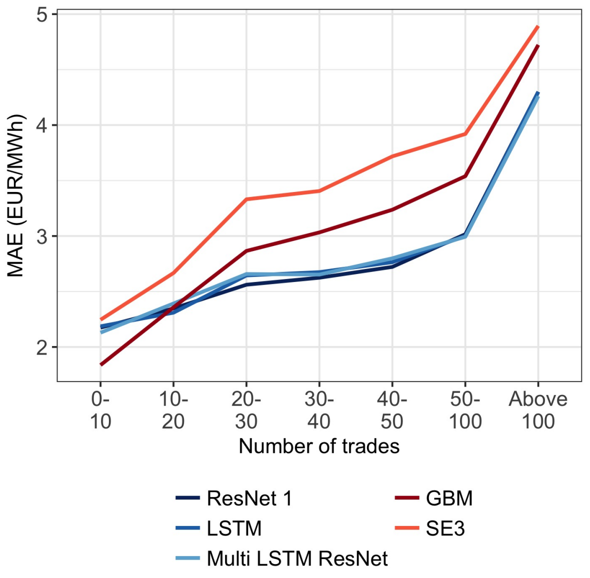
(b) RMSE

Figure 17: Variation in model performance depending on delivery hour. For the deep learning models, MAE and RMSE are the *random initialisation ensembling* (RIE) values. The performances are evaluated on the test set data.

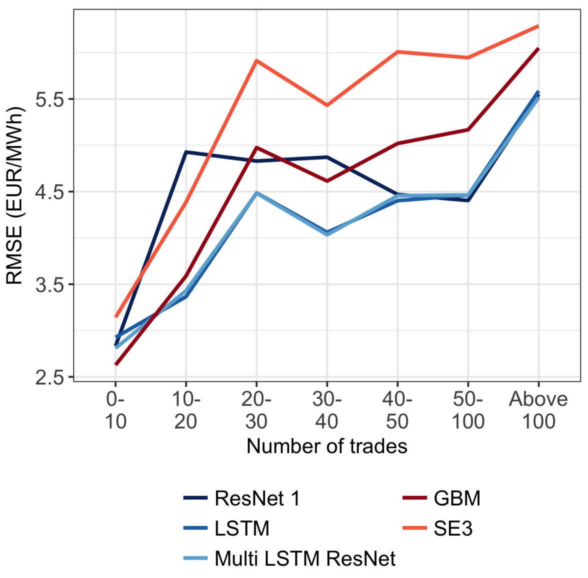
Furthermore, from Figure 17a, which shows performance measured by MAE, it becomes evident that the deep learning models are not superior to the simpler gradient boosting model — and only slightly better than the simplest baseline — for the first eight hours of the day. It is in the hours where prices are more volatile that the deep learning models are noticeably better than the simpler benchmarks. The same pattern emerges in Figure 17b, which shows the corresponding RMSEs. Interestingly, the *ResNet 1* model has especially high RMSE during the night hours 1–7 compared to the other deep learning models. In fact, starting with delivery hour 10, its RMSE is indistinguishable from those of the *LSTM* and *Multi LSTM ResNet* models. The higher RMSE of *ResNet 1* in Table 6 can therefore likely be attributed to its poor performance over the first hours of the day. In conclusion, the *LSTM* and *Multi LSTM ResNet* models stand out as the models with the best overall performances over the course of the day.

6.2.2 Performance across Trading Activity

Trading activity, measured in the number of settled trades for a given hour of power delivery, can differ considerably (Section 4.5.1). For some hours, only a few trades may be settled, while for others, the number may approach — or even exceed — 100. Figure 18 shows how the models perform across various activity levels per given delivery hour.



(a) MAE



(b) RMSE

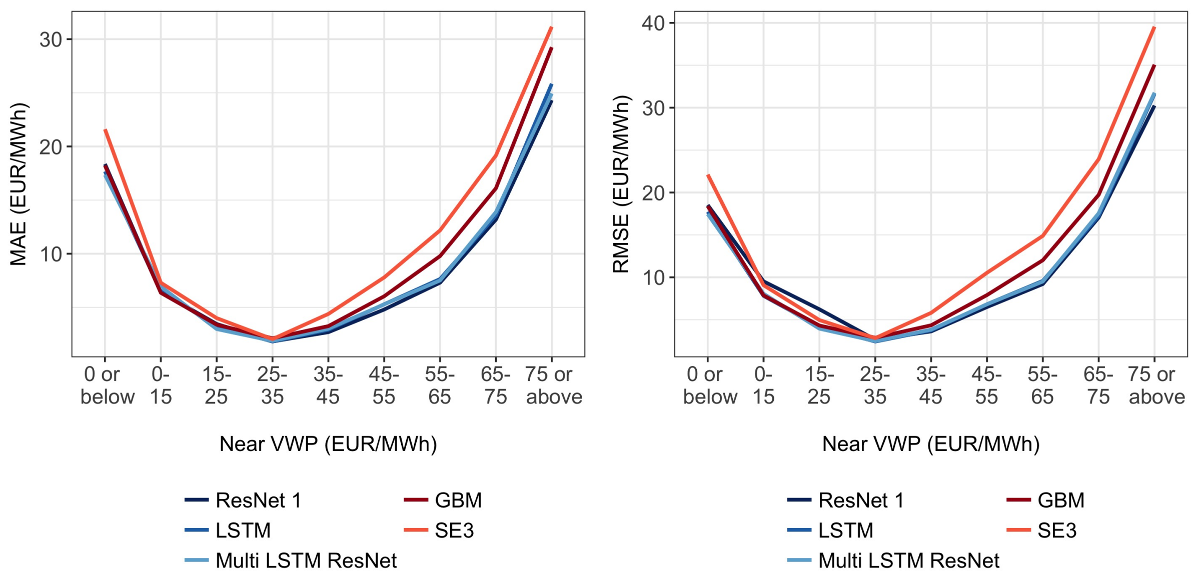
Figure 18: Variation in model performance depending on number of trades. For the deep learning models, MAE and RMSE are the *random initialisation ensembling* (RIE) values. The performances are evaluated on the test set data.

Performance appears to be negatively correlated with trading activity levels for all models; i.e. when trading activity in Elbas is higher, it becomes harder to predict the average price. When the number of trades for a given delivery hour is low, the simpler gradient boosting benchmark model actually performs better than the deep learning models. However, when trading activity increases, the deep learning models start outperforming the simpler benchmarks. This can be regarded as an advantage of the deep learning models; accurate price predictions may generally be more valuable when there is higher activity in the market and multiple trades to choose between. However, when trading activity is at its highest, with more than 100 transactions settled for one delivery hour, the MAE values of the deep learning models also increase sharply from around 3 EUR/MWh to about 4.25 EUR/MWh (Figure 18a). Still, the simpler benchmarks have higher MAE values in this range, between 4.5–5 EUR/MWh.

In terms of RMSE, we again see that *ResNet 1* performs worse than the other deep learning models for certain levels of trading activity. Unsurprisingly, this mainly occurs at lower activity levels; recall that trading activity is lower for the first hours of the day (Figure 7b), and that *ResNet 1* perform worse during these hours (Figure 17b). The performances of the other two deep learning models are, once again, difficult to distinguish. Due to their better relative performance when trading activity increases, we conclude that *LSTM* and *Multi LSTM ResNet* still emerge as superior to *ResNet 1*.

6.2.3 Performance across Price Levels

The *interquartile range* (IQR) for the *near* Elbas VWP in the test set is 26.26–34.36 EUR/MWh (Table 2, Section 4.5.2). That is, half of the delivery hours experience prices within a span of 10 EUR/MWh. However, prices may deviate significantly from the IQR in certain hours, with minimum and maximum prices of respectively -6.32 and 113.56 EUR/MWh. However, it is not clear how models' performances may vary depending on the level of the true *near* Elbas VWP. Figure 19 therefore shows the MAE and RMSE for the models over nine *near* VWP buckets.



(a) MAE

(b) RMSE

Figure 19: Variation in model performance depending on the true *near* Elbas VWP level. For the deep learning models, MAE and RMSE are the *random initialisation ensembling* (RIE) values. The performances are evaluated on the test set data.

Unsurprisingly, the predictive errors for all models reach their lowest point when the true *near* Elbas VWP is approximately within the IQR — i.e. the 25–35 EUR/MWh bucket. The more the *near* VWP deviates in a positive direction, the higher are the predictive errors for all models, with MAEs up to 25 EUR/MWh and RMSEs up to 30 EUR/MWh for the deep learning models, and values closer to 30 EUR/MWh and 35–40 EUR/MWh, respectively, for the two benchmarks. The errors also increase sharply when the *near* VWP deviates in a negative direction, albeit to a lesser degree, with MAEs around 20 EUR/MWh for all models when the *near* VWP is zero or below. From both Figure 19a and Figure 19b, we can see that when prices are *lower* than the typical *near* VWP range, the different approaches are not as distinguishable. On the other hand, the deep learning models obtain lower errors compared to the simpler benchmarks when prices are *higher*. Still, the high MAEs and RMSEs render even the deep learning models considerably less reliable when price levels are especially high.

6.2.4 Detecting Unusually High or Low Price Levels

The results in Section 6.2.3 may suggest that the models are basically useless once prices start to deviate from the most common price range of about 25–35 EUR/MWh, but this is not necessarily the case. Figure 20 plots the *near* VWP predictions from using the *Multi LSTM ResNet* model (red) compared to the true values (blue), over a three-week period where the price was at times quite volatile. The stippled lines are included to mark the test set IQR. Looking at the figure, it becomes evident that the *Multi LSTM ResNet* model follows the true price fluctuations fairly well — even when the prices are outside of the IQR — and that it mainly falls short when it comes to predicting the full *magnitude* of price fluctuations.

A model's ability to predict *when* prices will deviate — as opposed to predicting the *exact* fluctuation — may also be useful for market participants. The model may, for example, indicate when the *near* VWP is likely to go outside the range within which it is usually able to provide accurate prediction, and prudently involve a human decision-maker. Hence, we also consider the different models' ability to detect unusually high or low prices as a criterion that is relevant to take into account when settling on a preferred model. Potential applications related to the detection of deviating prices are further discussed in Chapter 7; here, we will evaluate to what extent the models manage to do so. That is, instead of asking what the exact price will be, we ask whether the *near* VWP will exceed a certain limit — like, for example, 50 EUR/MWh — and evaluate each model's success rate in anticipating such events.

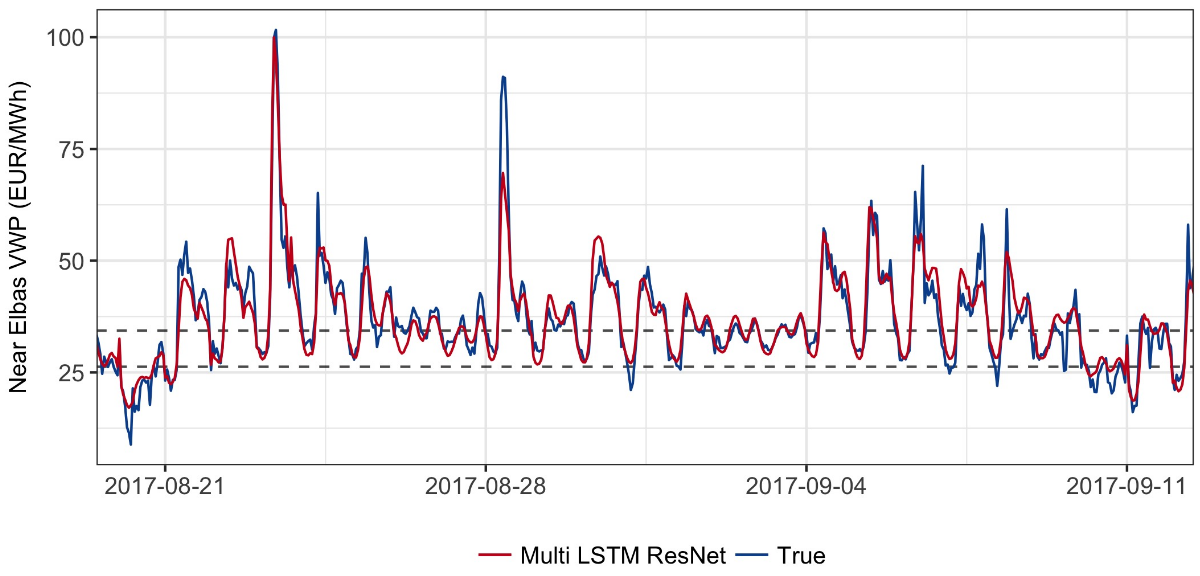
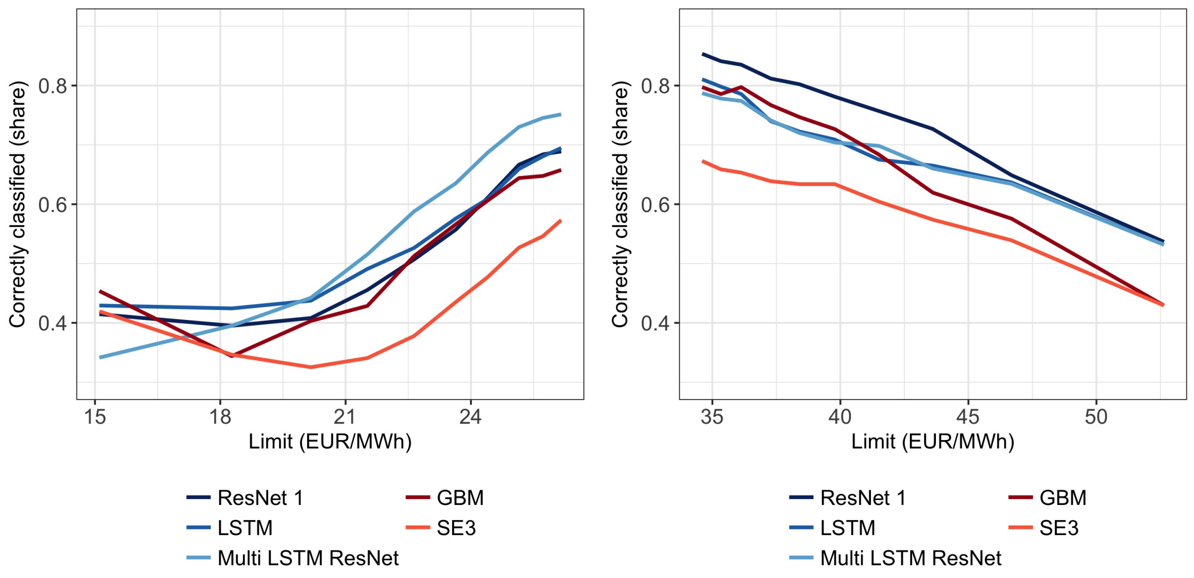


Figure 20: *Near* Elbas VWP predictions from using the *Multi LSTM ResNet* model (red) compared to the true values (blue), over a three-week period with at times quite volatile prices. The stippled bands mark the lower and upper limit of the test set *interquartile range* (IQR).

Figure 21 presents the degree to which the models are able to correctly predict that the *near* VWP would be lower (21a) or higher (21b) than a certain limit.[101] For the lower price limits, Figure 21a shows that when the the limit is set lower, the models struggle more to predict that the price will drop below it. If the limit is set around the lower band of the IQR — i.e. around 26 EUR/MWh — the *Multi LSTM ResNet* model correctly anticipates about 75% of price movements that drop below it, compared to about 69% for *LSTM* and *ResNet 1*, about 66% for gradient boosting, and about 57% for the SE3 heuristic. If the limit is instead set at around 20 EUR/MWh, the shares drop to below 45% for all models. The deep learning models generally tend to perform slightly better than the benchmarks — albeit the gradient boosting model is almost on the same level — until the limit is set below 17 EUR/MWh. Below this limit, the *Multi LSTM ResNet* model, in particular, does not perform as well as the other deep learning models, and is also worse than the simpler benchmarks, which on their side perform as well or better than the *LSTM* and *ResNet 1* models.

Figure 21b shows that the models' ability to anticipate that a price will exceed a given limit decreases as the limit is increased. For limits not much higher than the upper bound of the IQR — i.e. around 35 EUR/MWh — the *ResNet 1* model anticipates about 85% of cases,

[101] We let the limits for the higher price range vary from the 75th to the 97.5th percentile of the *near* VWP, and for the lower price range from the 25th to the 2.5th percentile.



(a) Lower prices (b) Higher prices

Figure 21: The share of correctly classified lower (21a) or higher (21b) prices for different levels of the *near* Elbas VWP.

followed by *LSTM*, gradient boosting, and *Multi LSTM ResNet* with shares around 80%. The SE3 heuristic predicts correctly 67% of the time, and generally performs worse than the other models regardless of where the limit is set. The *ResNet 1* model is overall best at anticipating exceed price movements over the higher price ranges, but the higher the limit is set, the closer the other deep learning models get to *ResNet 1*. Gradient boosting starts to struggle significantly, relative to deep learning, as the limit is set higher. If set above 50 EUR/MWh, gradient boosting is on the same level as the SE3 heuristic, with both anticipating about 43% of cases. In comparison, all of the deep learning models anticipate about 53% of cases — though they too struggle considerably more when the limit is much higher than the upper bound of the IQR.

In summary, the deep learning models generally tend to be better than gradient boosting at anticipating when the price will cross some upper or lower threshold. All soundly beat the SE3 heuristic, except when the limit is set below 17 EUR/MWh[102]. All models struggle more as the thresholds move further away from the IQR. There is, however, a trade-off in deciding where to set the limits; closer to the IQR means the models are more likely to anticipate movements that cross the thresholds, but if this is used to decide when to involve a human decision-maker, it may negate some of the gains in automation as it will also be triggered more often.

[102]This is likely a situation where day-ahead prices contain most of the predictive power for such events. Put differently, the deep learning models learn to build on the SE3 baseline, but never to take it completely at face value. This is one of the few situations where the latter is a disadvantage.

6.3 Conclusion

We find that it is possible to predict the *near* Elbas VWP six trade hours in advance with an accuracy of up to an MAE of 2.25 EUR/MWh on the validation set, and 2.66 EUR/MWh on the test set. In general, the deep learning models yield a lower predictive error than the benchmarks. For all deep learning models, the margin is approximately 0.33–0.44 EUR/MWh on the validation set and 0.30–0.36 on the test set, when comparing the RIE MAE for the deep learning models with the MAE of the gradient boosting model. This corresponds to improvements of up to 16% (validation) and 12% (test). Compared to the simplest baseline of predicting the SE3 spot price, the networks are approximately 21% more accurate on the validation data and 25% on the test data. Hence, using deep learning to predict the *near* Elbas VWP price may in fact yield significantly better performance.

The in-depth evaluation revealed that deep learning tends to perform better, relative to the other approaches, for delivery hours where the price is volatile and where trading activity is higher — although deep learning also struggles more during these periods. Among the deep learning models, *LSTM* and *Multi LSTM ResNet* yield the best results on the validation set, while *ResNet 1* is superior on the test set in terms of MAE. We also find that the higher RMSE values of *ResNet 1* stem from its worse performance for delivery hours with lower trading activity, in particular hours 1–7. All models struggle when the *near* Elbas VWP reaches unusually high or low levels. However, the deep learning models have generally shown a better ability than the benchmarks to anticipate that the price will deviate from the interquartile range.

In conclusion, the *LSTM* and the *Multi LSTM ResNet* models are difficult to distinguish in terms of MAE and RMSE. That said, we assert that the model with the best potential for predicting the *near* Elbas VWP is the *Multi LSTM ResNet* model, for multiple reasons. Firstly, it is designed to use all available data, including the weather data that we know from literature may have a significant influence on intraday prices. There is a limitation in that, due to technical limitations, we do not include comparable forecasts from the day-ahead market. If included, these may improve the performance of this particular model further. Secondly, the *Multi LSTM ResNet* model gains less from ensembling, but is still comparable to the other models, which suggests that if one were to take the time to train more diversified versions it would be even better. This model is, however, also more computationally expensive to fit.

7 Discussion

Although deep learning shows great promise, there are some non-negligible barriers to implementing it. Considering the numerous variables involved — such as how much to buy or how long one is willing to wait — and ways of utilising the forecasts, any attempt to precisely estimate the potential value-add of our deep learning models to a buyer in Elbas would arguably be futile or too specific to generalise. Similarly, while there are some largely universal prerequisites for successful deep learning projects, the need for — and therefore cost of — preparatory groundwork can vary greatly. Organisations that are already relatively data-driven will have fewer hurdles to overcome in implementing deep learning, whereas those without effective data curation or a culture of data driven decision making will likely need to incur greater costs to reap the benefits. As such, our focus here is on how our models can be improved, the wide range of other applications of deep learning in intraday electricity markets, and how deep learning — as a subset of artificial intelligence — fits into the greater picture of automation. Consequently, we aim to give the reader sufficient insight to envisage the potential value-adds of such projects under the specific conditions of his or her organisation.

7.1 The Potential of Deep Learning

Though the performance margin between our network and the simple SE3 heuristic is arguably relatively small — considering the vast difference in complexity — at 21–25% on the unseen data, the simple heuristic is already performing at full potential. In contrast, while our networks are already better, we assert that, if anything, they demonstrate promising potential, and that there is no reason to expect that we have exhausted the potential of deep learning on the problem we have defined. Furthermore, while our architectures can likely be improved further, there are also a myriad of other ways deep learning can be implemented in the Elbas market. Other methods, such as gradient boosting, could be coerced toward similar ends, but would arguably be much more cumbersome to use. In contrast, there are specialised types of deep learning appropriate to a wide range of applications that both perform better and are, arguably, easier to use (Chollet, 2018). These specialised variations also mean deep learning can open up new avenues of applications, or facilitate the use of data that is too unorthodox to be practically usable by more traditional techniques — such as, presumably, weather forecast maps[103].

Improving our neural networks

Due to time constraints, we had to make some simplifying choices when designing our neural networks. It is likely that more thoroughly tuning some of the hyperparameters and network topologies would improve their performances further. We performed much of the experimentation needed to identify promising avenues of possible network architectures, but since some of these have now been found, a more thorough search of the possible configurations will likely yield architectures that perform even better. In particular, we left the hyperparameters of the optimising algorithm at their default values. Fine-tuning these would likely improve the network's performance (Goodfellow et al., 2016) — especially in conjunction with checkpoint ensembling (H. Chen et al., 2017). Similarly, the window of preceding market- and weather data the network has access to when making predictions is an important decision, and can likely be adjusted further now that promising architectures have been identified. In the same way, training the networks for longer, introducing other forms of regularisation, or tuning other hyperparamters more extensively could all improve performance further.

Additionally, the networks will only become better with more data (Chollet, 2018), where an additional year increases the dataset our networks were trained on by approximately 16%. Crucially, we had to greatly simplify the weather forecast data due to time- and resource constraints; incorporating multiple forecasts for each hour of delivery to capture changes in expected weather conditions, both over time and after the day-ahead price is set, would presumably allow the networks to more reliably anticipate deviations between the day-ahead and intraday prices — especially as the importance of variable renewable energy sources, with their dependence on concurrent weather conditions, continues to increase (Scharff & Amelin, 2016).

Other applications of deep learning in Elbas

Our neural networks are superior on the problem we have defined. While this has merit in itself, we also assert that it suggests such techniques could be highly successful in other related applications as well. There are specialised types of deep learning for different purposes, such as reading in or outputting sequential data, or processing images — or combinations of both. For example, Section 6.2.4 shows that our final network tends to anticipate unusually high or low prices. Considering our choice of loss function did not particularly reward such behaviour, a

[103]We cannot reasonably assert that these have been unusable without deep learning as there are other ways to use them. However, forms of deep learning specialised for image processing greatly simplify the process, and are, arguably, better suited for that purpose.

network trained on a more customised loss function would likely be better at predicting major deviations in prices. These networks could be used in conjunction; our network automatically buys power in the intraday market[104], while the more customised network recognises when to involve a human decision-maker with domain expertise — e.g. when the situation deviates outside a learned norm, or if the automated network is insufficiently confident in its prediction. This would automate the laborious task of business-as-usual trading, and reserve decision makers' valuable attention for situations where their insight is best put to use.

A possible extension is therefore having the networks output some confidence level or probability that the error will lie within some margin — in addition to the intraday price itself. This could be used as above to involve decision-makers as needed, or facilitate better ensembling by giving an amalgamating model additional context with which to combine predictions from its constituent submodels. On this note, while we have used some ensembling, performance could likely be improved further by ensembling *deep and wide*, which combines deep learning with more shallow methods (Chollet, 2018). Another alternative is a network that outputs a sequence of price developments over the next, e.g. 4–8, hours for each hour of delivery. This would give decision makers continuously updated and refined forecasts of aggregated or trade-level prices as the delivery hour approaches, with which to better adjust their power-balancing tactic.

Gains from automation

Deep learning automates important parts of the model building that previously required deep domain expertise (Chollet, 2018). It therefore lowers the barriers to developing an artificial intelligence for a given domain. In addition, the largest challenge lies in developing the model. However, once built, it can be used to automate processes that previously required laborious and repetitive analyses by human experts.[105] Presumably, many participants in Elbas primarily trade during office hours due to the costs of employing workers outside these periods. Deep learning could be used to automate some of the work during office hours, or, since our network performs relatively well during off-peak hours (Figure 17b), handle some basic trading outside office hours. A detailed overview of the gains from automation is beyond the scope of this thesis, but if anything, we would argue that the effectiveness of our models demonstrate the potential of deep learning as a way to automate previously manual or, even, hitherto unhandled processes. As such, it can help take repetitive decisions out of the hands of individuals, both to reduce their laborious tasks so they can focus on what truly adds value, and to eliminate subjective

[104]Based on strategies or needs specified either manually or by another dedicated AI.

[105]Doing so over longer periods of time warrants intermittent re-training of the models, however.

and subconscious factors from what should arguably be objective and rational decisions.

7.2 Practical Barriers to Deep Learning

For all its promise, deep learning has a potentially daunting list of prerequisites. Deep learning is notoriously *"data hungry"* (Chollet, 2018), and requires extensive and meticulously curated data. Such models are only as good as the data they are fed[106], hence one must incur the cost of building robust and accurate data pipelines before operational deep learning is feasible. Furthermore, though deep learning automates part of what was traditionally domain-specific model-building necessarily reserved for experts in that domain, it still requires a certain level of technical proficiency to implement. An advantage is that such expertise is predominantly needed during the development of these models (Gu et al., 2018); supervising their operational use and keeping them up-to-date over time is less demanding. Similarly, while training the networks requires significant computational power, they need fewer — though not negligible — resources to make predictions once fit.[107] As such, the varying needs for technical infrastructure can be met by renting cloud-based computing power during development and training, and instead running the final models on simpler — potentially in-house — infrastructure for operational use.

Consequently, utilising AI techniques like deep learning requires certain types of technically adroit human capital. While these are relatively scarce and, therefore, may command a high price (Metz, 2017), it is not necessarily more expensive than the traditional approach of hiring domain experts. Electricity pricing is a very complicated market where, if you want forecasts, you either buy them at a premium from a provider or hire your own in-house experts. With deep learning, we argue that the need for expensive domain experts is reduced (Chollet, 2018), and — while there is an up-front cost of technical experts, data curation, and computational power — that these are generally more accessible and cost less in the long run — in part because deep learning can automate some of what would be a more repetitive and recurring process for domain experts. Another key consideration is that while the initial hurdles to implementing deep learning successfully are certainly notable, once those hurdles are overcome it is much easier to implement other projects that branch out — especially if they use much of the same infrastructure and data pipelines. The same holds if an organisation already intends to implement simpler machine learning projects, as much of the prerequisite groundwork is the

[106]See *"garbage in, garbage out"*: `https://en.wikipedia.org/wiki/Garbage_in,_garbage_out`

[107]On our relatively modest hardware, making predictions for a full year of data takes less than 10 seconds. Hence, individual predictions in a production setting would effectively be instantaneous.

same. However, it is important to not underestimate the potential for incurring significant upkeep of operationalised deep learning systems (Sculley et al., 2014).

7.3 The "Black Box" Critique

A common criticism of deep learning and other relatively complex forms of machine learning is that such models lack *interpretability* (Kumar, Taylor, & Wong, 2017; Shickel, Tighe, Bihorac, & Rashidi, 2017; Zhang & Zhu, 2018). The way they make predictions, the argument goes, is an opaque "black box" whose inner workings are indiscernible. Certainly, the desire for an unequivocal mapping of how such a model arrives at its conclusions is understandable; perhaps there are legal requirements for providing indubitable reasons for some outcome (Knight, 2017) — such as why a credit application was denied. Reverse-engineering why a complex neural network made the prediction or drew the conclusion it did can be difficult. Fortunately, there is a lot of promising recent research toward this end, such as *attention mechanisms* for recurrent networks (Brown, Tuor, Hutchinson, & Nichols, 2018). As such, the black box problem might therefore attenuate in time, although this could be offset by AI systems becoming more complex.

At the same time, we posit that some of these criticisms stem from outmoded ideas of artificial intelligence and a quixotic imperative that their decisions should always be perfectly transparent. In some situations there is a clear need, but in many others this is arguably wishful thinking and unnecessary. Does the patient really care[108] how an AI discovers a malignant tumour, if it does so more reliably than doctors (Pande, 2018)? Similarly, should an intraday electricity market participant elect not to utilise an AI that outperforms all domain experts simply because we cannot discern its reasoning? As such, while we certainly acknowledge the black box problem associated with deep learning, we also make the argument that this, like anything else, should be a pragmatic consideration undistracted by idealism.

7.4 Theoretical Limitations and Avenues for Further Research

For a model to be able to make accurate predictions of the intraday price, the market mechanisms that truly affect the price must be well captured by the input variables. In this regard, strategic behaviour among market participants poses a particular challenge. Contrary to physi-

[108]Of course, doctors and academia might find this insight invaluable in furthering their understanding of the disease or in developing a cure, but from the perspective of the "market participant", accuracy and accessibility are arguably all that matter.

cal market factors such as available intraday capacities, the individual bidding behaviour of one participant is difficult — if not impossible — for other participants to account for in a price model. Additionally, there may even be challenges related to physical market factors and their true impact on the price. The fact that not all participants would turn to Elbas trading if they experience an imbalance in their portfolio — for example, they may instead balance internally or leave the imbalance to the regulating market — can weaken the model's ability to detect patterns from the underlying data. Furthermore, some variables that, from a theoretical perspective, can impact the price — such as hydro reservoir content and inflow, updated consumption estimates, and transmission flow data — are not incorporated in our model due to a lack of data. Further research utilising deep learning may want to consider how these limitations might better be accounted for, given the available data.

We aggregate trading, both at a theoretical system-wide price for Nordic buyers and over the last six hours of delivery, partially due to high sparsity if the intraday prices are kept at their original grain of individual bidding areas or individual trades. There was a trade-off between aggregating to reduce gaps in the data, and keeping sufficient detail to both be informative and give buyers sufficient time to plan their strategy using the resulting forecasts. In aggregating trades across the Nordic countries, the model is applicable to a buyer from any Nordic bidding area — albeit under the assumptions that the bidding areas are in fact integrated into one single market, and that trades may happen between any two areas. Bottlenecks in the transmission grid system may limit the degree to which this is the case for every hour of power delivery (Fridolfsson & Tangerås, 2009). In particular, this model may not predict a price that is representative for the Nordic countries regardless of bidding area, in periods with high congestion and limited transmission capacity left for Elbas trading. Due to the potential influence of transmission grid bottlenecks in hindering market integration, further research may want to consider the volume-weighted price per buyer bidding area, and not for the Nordic region as a whole.

A way to solve the issue of missing outputs could be to develop a model that both predicts if trades will happen in a given bidding area, and if so, the price.[109] Also, if market liquidity increases in the future — for example due to the forthcoming cross-border intraday trading initiative[110] — the basis for modelling the price for each bidding area separately may be improved.

[109]There are some network types that can incorporate such gaps by reading variable-length input sequences, but given the time constraint for this thesis, we found these to be too complicated for our purpose.

[110]The Cross-Border Intraday Market Project (XBID) will enable continuous trading of electricity across 12 European countries, where orders submitted by market participants in one country can be matched by orders submitted by market participants in any other country within the project, given that transmission capacity between the two countries is available.

The extent to which Elbas may be considered as one integrated market with one price — and hence, whether price models may be constructed on an aggregated level or must be adapted for each separate bidding area or groups of bidding areas — can also be an interesting avenue for further research. Finally, the correlation between the spot prices and the intraday prices are high, as suggested by the spot price in SE3 being a relatively good prediction for the *near* Elbas VWP. Due to this high correlation, further research may consider modelling the *difference* between the spot and intraday prices, instead of the exact intraday price.

8 Conclusion

Trading power in Nord Pool's intraday market Elbas allows participants to modify their positions close to real-time. However, doing so profitably is not trivial, as prices can fluctuate significantly over time and vary greatly between trades for the same hour of power delivery. Hence, access to reliable predictions of the baseline price that is reasonable over some window of trading can be very valuable in adjusting tactical trading decisions. To this end, we develop models that, for each delivery hour, predict a volume-weighted average price over the last six hours of trading for Nordic buyers. Trading activity is usually concentrated in this window, which gives the buyer flexibility from many offers and sufficient time in which to utilise the predictions in making better trades. We use deep learning in an effort to comprehensively capture the market's complexity and intricacies — in part by exploiting as much of the available data as possible – and to demonstrate the broader potential of artificial intelligence in intraday electricity markets.

We find that deep learning provides relatively accurate and reliable predictions of hourly volume-weighted prices under normal conditions in Elbas. After developing and evaluating a range of such models, we conclude that the best neural network is that which combines sequences of comprehensive market data with sequences of weather forecast images. The average price across all delivery hours in the test dataset is 30.95 EUR/MWh, where the predictions of this network are, on average, off by a magnitude of 2.72 EUR/MWh. The best simple heuristic is to always predict that this price will equal the day-ahead spot price in the SE3 bidding area for the corresponding delivery hour. Our network soundly beats this baseline by 21–25%, albeit at significant added complexity. It also outperforms a range of benchmark models we develop from among more traditional methods — most of which require largely the same data preparation as deep learning. The best of these is gradient boosting, which our network beats by 12–16%.

In addition to outperforming all benchmarks on aggregate levels, the network also tends to perform better when prices are volatile or when trading activity is high. It also anticipates major fluctuations in prices with some consistency, which suggests that a specialised AI could be quite successful at it. Furthermore, while most benchmarks are already performing at full potential, there is no reason to believe that our network is; refining the architecture, fine-tuning hyperparameters, or including more market data once available could all improve performance. Importantly, due to time and resource constraints we only incorporate one weather forecast per delivery hour. While simple experiments of networks that only used these forecasts when predicting prices were surprisingly successful, including them in networks that were already using the market data had a negligible effect. Since weather is important for electricity intraday markets — and will only become more so as the share of vRES in the energy mix continues to rise — we expect networks that instead process sequences of multiple weather forecasts for each delivery hour to deliver notable gains, as those capture important updates in the forecasts.

While we find that deep learning is superior in predicting a theoretical system-wide aggregated intraday price, we also argue that, if anything, this thesis demonstrates the wider potential of deep learning in a range of applications in such markets. There are specialised types of deep learning — not to mention of artificial intelligence, more broadly — for a range of AI problems relevant to intraday electricity markets, such as predicting sequences of prices, predicting prices in specific bidding areas, forecasting other values such as consumption levels, or image processing. There are, however, some notable prerequisites for market participants to realistically develop operational deep learning systems. Among these is the need for comprehensive and reliable data curation, as well as technical expertise and significant computational power — at least during the development phase, though one should also be aware of the potential for accumulating high upkeep costs of such systems. Nevertheless, these up-front investments and upkeep costs are not necessarily greater than the long-term implications of relying on domain experts or persisting with wholly manual repetitive processes. Artificial intelligence is making impressive strides in a range of industries, and while it is important to have realistic expectations and to be cognisant of the salient pitfalls, there is little reason to think electricity intraday markets are different.

References

Alzahrani, A., Shamsi, P., Dagli, C., & Ferdowsi, M. (2017). Solar Irradiance Forecasting Using Deep Neural Networks. *Procedia Computer Science*, *114*, 304–313.

Andrade, J. R., Filipe, J., Reis, M., & Bessa, R. J. (2017). Probabilistic Price Forecasting for Day-Ahead and Intraday Markets: Beyond the Statistical Model. *Sustainability*, *9*, 1990–2019.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional Time Series Forecasting with Convolutional Neural Networks. *ArXiv e-prints*. (arXiv: 1703.04691v4)

Botnen Holm, T. (2017). *The future importance of short term markets: An analyse of intraday prices in the Nordic intraday market; Elbas* (Master's Thesis). Norwegian University of Life Sciences (NMBU).

Bourry, F., & Kariniotakis, G. (2009). Strategies for Wind Power Trading in Sequential Short-Term Electricity Markets. *European Wind Energy Conference (EWEC) 2009, Marseille: France (2009).*

Brown, A., Tuor, A., Hutchinson, B., & Nichols, N. (2018). Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection. *ArXiv e-prints*. (arXiv:1803.04967v1)

Brownlee, J. (2017). Long Short-Term Memory Networks With Python. *Machine Learning Mastery*. Retrieved from `https://machinelearningmastery.com/deep-learning-with-python/`.

Chen, H., Lundberg, S., & Lee, S.-I. (2017). Checkpoint Ensembles: Ensemble Methods from a Single Training Process. *ArXiv e-prints*. (arXiv:1710.03282v1)

Chen, Y., He, K., & Tso, G. K. F. (2017). Forecasting Crude Oil Prices: a Deep Learning based Model. *Procedia Computer Science*, *122*, 300–307.

Chollet, F. (2018). *Deep Learning with Python.* Shelter Island, New York: Manning Publications Co.

Cui, Z., Ke, R., & Wang, Y. (2018). Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction. *ArXiv e-prints*. (arXiv:1801.02143v1)

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Lecture Notes in Computer*

Science, *1857*, 1–15.

Engmark, E., & Sandven, H. (2017). *Stochastic multistage bidding optimisation for a Nordic hydro power producer in the post-spot markets* (Master's Thesis). Norwegian University of Technology and Science (NTNU).

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*, 654–669.

Fodstad, M., Aarlott, M. M., & Midthun, K. T. (2018). Value-Creation Potential from Multi-Market Trading for a Hydropower Producer. *Energies*, *11*, 16–31.

Fridolfsson, S.-O., & Tangerås, T. P. (2009). Market power in the Nordic electricity wholesale market: A survey of the empirical evidence. *Energy Policy*, *37*, 3681–3692.

Garnier, E., & Madlener, R. (2015). Balancing forecast errors in continuous-trade intraday markets. *Energy Systems*, *6*, 361–388.

George Mason University. (n.d.). Wind: u and v Components. *Virginia Weather and Climate Data Development Web Site*. Retrieved from `http://colaweb.gmu.edu/dev/clim301/lectures/wind/wind-uv`. (Accessed: 2018-06-10)

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation*, *12*, 2451–2471.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* Cambridge, Massachusetts: The MIT Press.

Graves, A., Jaitly, N., & Mohamed, A.-r. (2013). Hybrid Speech Recognition with Deep Bidirectional LSTM. *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on. IEEE*, 273–278.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., … Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354–377.

Hagemann, S. (2013). *Price Determinants in the German Intraday Market for Electricity: An Empirical Analysis* (EWL Working Paper). University of Duisburg-Essen.

Hagemann, S., & Weber, C. (2013). *An Empirical Analysis of Liquidity and its Determinants in the German Intraday Market for Electricity* (EWL Working Paper). University of Duisburg-Essen.

Hagemann, S., & Weber, C. (2015). *Trading Volumes in Intraday Markets – Theoretical Ref-*

erence Model and Empirical Observations in Selected European Markets (EWL Working Paper). University of Duisburg-Essen.

Hagfors, L. I., Bunn, D., Kristoffersen, E., Staver, T. T., & Westgaard, S. (2016). Modeling the UK electricity price distributions using quantile regression. *Energy*, *102*, 231–243.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *Elements of Statistical Learning – Data Mining, Inference, and Prediction* (2nd ed.). Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv e-prints*. (arXiv:1512.03385v1)

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity Mappings in Deep Residual Networks. *ArXiv e-prints*. (arXiv:1603.05027v3)

Hellström, J., Lundgren, J., & Haishan, Y. (2012). Why do electricity prices jump? Empirical evidence from the Nordic electricity market. *Energy Economics*, *34*, 1774–1781.

Henriot, A., & Glachant, J.-M. (2013). Melting-pots and salad bowls: The current debate on electricity market design for integration of intermittent RES. *Utilities Policy*, *27*, 57–64.

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

Holttinen, H., & Koreneff, G. (2012). Imbalance Costs of Wind Power for a Hydro Power Producer in Finland. *Wind Engineering*, *36*, 53–67.

International Energy Agency. (2016a). Denmark - Energy System Overview. *International Energy Agency*. Retrieved from `https://www.iea.org/media/countries/Denmark.pdf`.

International Energy Agency. (2016b). Finland - Energy System Overview. *International Energy Agency*. Retrieved from `https://www.iea.org/media/countries/Finland.pdf`.

International Energy Agency. (2016c). Germany - Energy System Overview. *International Energy Agency*. Retrieved from `https://www.iea.org/media/countries/Germany.pdf`.

International Energy Agency. (2016d). Norway - Energy System Overview. *International Energy Agency*. Retrieved from `https://www.iea.org/media/countries/Norway.pdf`.

International Energy Agency. (2016e). Sweden - Energy System Overview. *International Energy Agency*. Retrieved from `https://www.iea.org/media/countries/Sweden.pdf`.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*. (arXiv:1502.03167v3)

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning*. Springer.

Kiesel, R., & Paraschiv, F. (2017). Econometric analysis of 15-minute intraday electricity prices. *Energy Economics*, *64*, 77–90.

Kim, H. Y., & Won, C. H. (2018). Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models. *Expert Systems with Applications*, *103*, 25–37.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *ArXiv e-prints*. (arXiv:1412.6980v9)

Knapik, O. (2017). *Modeling and forecasting electricity price jumps in the Nord Pool power market* (CREATES Research Paper 2017-7). Aarhus University.

Knight, W. (2017). The Dark Secret at the Heart of AI. *MIT Technology Review*. Retrieved from `https://www.technologyreview.com`. (Accessed: 2018-06-10)

Kumar, D., Taylor, G. W., & Wong, A. (2017). Opening the Black Box of Financial AI with CLEAR-Trade: A CLass-Enhanced Attentive Response Approach for Explaining and Visualizing Deep Learning-Driven Stock Market Prediction. *ArXiv e-prints*. (arXiv:1709.01574v1)

Lago, J., De Ridder, F., & De Schutter, B. (2018). Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. *Applied Energy*, *221*, 386–405.

Lago, J., De Ridder, F., Vrancx, P., & De Schutter, B. (2018). Forecasting day-ahead electricity prices in Europe: The importance of considering market integration. *Applied Energy*, *211*, 890–903.

Lipton, Z. C., Kale, D. C., & Wetzel, R. (2016). Modeling Missing Data in Clinical Time Series with RNNs. *ArXiv e-prints*. (arXiv:1606.04130v5)

Liu, H., Mi, X., & Li, Y. (2018a). Smart multi-step deep learning model for wind speed forecasting based on variational mode decomposition, singular spectrum analysis, LSTM network and ELM. *Energy Conversion and Management*, *159*, 54–64.

Liu, H., Mi, X.-w., & Li, Y.-f. (2018b). Wind speed forecasting method based on deep learning strategy using empirical wavelet transform, long short term memory neural network and Elman neural network. *Energy Conversion and Management*, *156*, 498–514.

Lu, X., Dong, Z. Y., & Li, X. (2005). Electricity market price spike forecast with data mining techniques. *Electric Power System Research*, *73*, 19–29.

Mauritzen, J. (2013). *Now or Later? Trading wind power closer to real-time and how poorly designed subsidies lead to higher balancing costs* (Discussion Paper). Norwegian School of Economics (NHH).

Metz, C. (2017). Tech Giants Are Paying Huge Salaries for Scarce A.I. Talent. *The New York Times*. Retrieved from `https://www.nytimes.com`. (Accessed: 2018-06-10)

Monteiro, C., Ramirez-Rosado, I. J., Fernandez-Jimenez, L. A., & Conde, P. (2016). Short-Term Price Forecasting Models Based on Artificial Neural Networks for Intraday Sessions in the Iberian Electricity Market. *Energies*, *9*, 721–745.

Mosquera-López, S., Uribe, J. M., & Manotas-Duque, D. F. (2017). Nonlinear empirical pricing in electricity markets using fundamental weather factors. *Energy*, *139*, 594–605.

Nord Pool. (n.d.). The Nordic Electricity Exchange and The Nordic Model for a Liberalized Electricity Market. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/globalassets/download-center/rules-and-regulations/the-nordic-electricity-exchange-and-the-nordic-model-for-a-liberalized-electricity-market.pdf`.

Nord Pool. (2009). No. 113/2009 CORRECTION - Svenska Kraftnät's commitment to establish Bidding Areas in Sweden. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/message-center-container/newsroom/tso-news/2009/12/No-1132009-CORRECTION---Svenska-Kraftnats-commitment-to-establish-Bidding-Areas-in-Sweden-/`. (Accessed: 2018-05-20)

Nord Pool. (2016). 2016 Annual Report. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/globalassets/download-center/annual-report/annual-report_2016.pdf`. (Accessed: 2018-02-25)

Nord Pool. (2018a). Bidding Areas. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/the-power-market/Bidding-areas/`. (Accessed: 2018-02-18)

Nord Pool. (2018b). Day-ahead market. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/the-power-market/Day-ahead-market/`. (Accessed: 2018-03-09)

Nord Pool. (2018c). Elbas initial capacity. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/Market-data1/Intraday/Market-data13/Initial-capacity/Norway4/?view=table`. (Accessed: 2018-02-18)

Nord Pool. (2018d). Intraday trading. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/TAS/intraday-trading/`. (Accessed: 2018-02-15)

Nord Pool. (2018e). Order types. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/TAS/intraday-trading/order-types/`. (Accessed: 2018-02-25)

Nord Pool. (2018f). Producers. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/the-power-market/The-market-members/Producers/`. (Accessed: 2018-03-02)

Nord Pool. (2018g). Successful XBID launch. *Nord Pool*. Retrieved from `https://www.nordpoolgroup.com/message-center-container/newsroom/feature/2018/06/successful-xbid-launch/`. (Accessed: 2018-06-19)

NordREG. (2014). Nordic Market Report 2014 – Development in the Nordic Electricity Market. *NordREG - Nordic Energy Regulators*. Retrieved from `http://www.nordicenergyregulators.org/wp-content/uploads/2014/06/Nordic-Market-Report-2014.pdf`. (Accessed: 2018-03-02)

Pande, V. (2018). Artificial Intelligence's 'Black Box' Is Nothing to Fear. *The New York Times*. Retrieved from `https://www.nytimes.com`. (Accessed: 2018-06-10)

Pape, C., Hagemann, S., & Weber, C. (2016). Are fundamentals enough? Explaining price variation in the German day-ahead and intraday power market. *Energy Economics*, *54*, 376–387.

Peng, L., Liu, S., Liu, R., & Wang, L. (2018). Effective long short-term memory with differential evolution algorithm for electricity price prediction. *Energy*. (In Press, Accepted Manuscript)

Pires Ferreira, P. (2016). *Volume and Price in The Nordic Balancing Power Market* (Master's Thesis). Norwegian University of Technology and Science (NTNU).

Scharff, R. (2012). *On Distributed Balancing of Wind Power Forecast Deviations in Competitive Power Systems* (Licentiate Thesis). KTH School of Electrical Engineering.

Scharff, R., & Amelin, M. (2016). Trading behaviour on the continuous intraday market ELBAS. *Energy Policy*, *88*, 544–557.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Young, M. (2014). Machine Learning: The High-Interest Credit Card of Technical Debt. *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*.

Shen, X., Chen, Y.-C., Tao, X., & Jia, J. (2017). Convolutional Neural Pyramid for Image

Processing. *ArXiv e-prints*. (arXiv:1704.02071v1)

Shickel, B., Tighe, P., Bihorac, A., & Rashidi, P. (2017). Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis. *ArXiv e-prints*. (arXiv:1706.03446v2)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Srivastava, S., & Lessmann, S. (2018). A comparative study of LSTM neural networks in forecasting day-ahead global horizontal irradiance with satellite data. *Solar Energy*, *162*, 232–247.

Statnett. (2018). The power system right now. *Statnett*. Retrieved from `http://www.statnett.no/en/Market-and-operations/`. (Accessed: 2018-02-18)

Tangerås, T. P., & Mauritzen, J. (2014). *Real-time versus day-ahead market power in a hydro-based electricity market* (Discussion Paper). Norwegian School of Economics (NHH).

Voronin, S. (2013). *Price spike forecasting in a competitive day-ahead energy market* (PhD Dissertation). Lappeenranta University of Technology.

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

Wilson, D. R., & Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, *16*, 1429–1451.

Wolff, G., & Feuerriegel, S. (2017). Short-term dynamics of day-ahead and intraday electricity prices. *International Journal of Energy Sector Management*, *11*, 557–573.

Wooldridge, J. M. (2016). *Introductory Econometrics – A Modern Approach* (6th ed.). Boston, Massachusetts: Cengage Learning.

Zhang, Q., & Zhu, S.-C. (2018). Visual Interpretability for Deep Learning: a Survey. *ArXiv e-prints*. (arXiv:1802.00614v2)

Appendix

A Theory: Deep Learning

A.1 The Multi-Layer Perceptron

Given a vector of inputs $\mathbf{x}$ with a corresponding output y, the function f^* represents the true data-generating process, i.e. $y = f^*(\mathbf{x})$. The objective of neural networks is to approximate this target function with the function f. This is done by learning the network's *parameter values* $\boldsymbol{\theta}$ — which consist of potentially millions of *weights* and *biases* — through repeated exposure to the data, i.e. $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta}) \approx y$.

The typical example of a deep learning model is the *multi-layer perceptron* (MLP) — also called *fully-connected feed-forward networks* — which is effectively a chained set of functions that map input values to an output. They can be depicted as a directed flow of data from raw inputs to one — or a vector of multiple — output(s) via a chain of increasingly complex and abstract representations, by passing the data through the learned function f (Goodfellow et al., 2016). Each step in this chain is a *layer*, l. These define the *architecture* of the model, where layers are expressed as interlinked functions in a chain. They are the core constituent parts of the network, and transform the input data into representations more useful for solving the given problem (Chollet, 2018) — such as predicting a price. A network may consist of an arbitrary number L of such layers, so that $f(\mathbf{x}, \boldsymbol{\theta}) = f^{(L)}...(f^{(2)}(f^{(1)}(\mathbf{x}; \boldsymbol{\theta}^{(1)})))$.[111] The design and composition of these functions determine the *hypothesis space*, namely the set of functions the model is able to select as its solution (Goodfellow et al., 2016). Figure 22 depicts an example of a fully-connected feed-forward neural network.

A layer in the network can be represented as consisting of a number of units $i = 1, ..., M_l$ (Borovykh et al., 2017), also called *nodes* or *neurons* — hence *neural* network. Informally, the number of layers, L, defines the *depth* of the network (Goodfellow et al., 2016). There are three types of layers: *input*, *hidden*, and *output*. The input layer[112] does not transform the data, but simply passes the input values to the units in the next layer, and hence, it is here not counted as a layer in l. A sequence of hidden[113] layers $l = 1, ..., L-1$ follows the input

[111] $\boldsymbol{\theta}^{(1)}$ represents the parameters for layer 1 and $\boldsymbol{\theta}$ the parameters for the full network, i.e. $\boldsymbol{\theta} = \boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(L)}$.
[112] This layer is also called the *visible* layer, which refers to the variables being observable.
[113] The term *hidden* reflects that the values of the layers are not directly given by the data, but instead determined by the model.

layer. These transform the input data by extracting increasingly abstract representations that are more meaningful than their original form. The final hidden layer outputs a set of hidden features $\mathbf{h}^{(L-1)} = f^{(L-1)}(...(f^{(2)}(f^{(1)}(\mathbf{x}))))$ to the output layer, which makes a final prediction $\hat{y}$.[114] The connections between the units in one layer l and the units in the preceding layer are the layer's weights, i.e. the matrix $\mathbf{W}^{(l)}$. As we only have one unit in the output layer, this layer's weights is the vector $\mathbf{w}^{(L)}$.

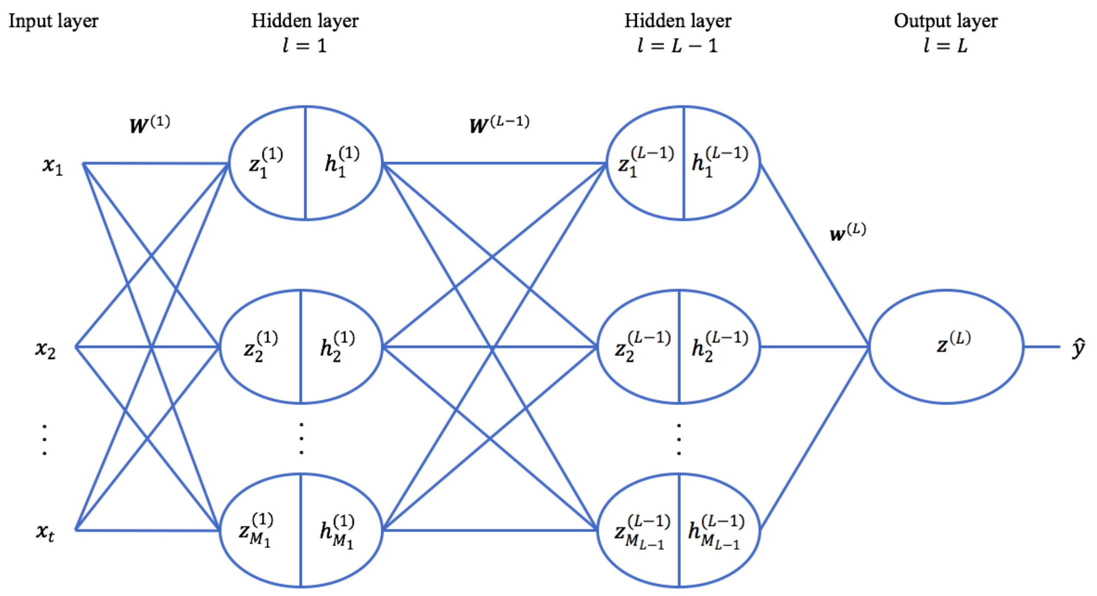


Figure 22: Multi-layer perceptron with L layers and M_l units in each layer. The weights of each layer is denoted as the matrix $\mathbf{W}^l$ for the hidden layers, and the vector $\mathbf{w}^L$ for the output layer as it consists of one single unit. A vector of inputs $\mathbf{x} = x_1, x_2, ..., x_t$ is propagated through the network to produce the prediction $\hat{y}$. An affine transformation $z_i^{(l)}$ is performed in each unit. For the hidden layers, a non-linear activation is applied on top of $z_i^{(l)}$ to produce $h_i^{(l)}$ which is fed forward in the network.

A.1.1 Activation Functions

Each neuron i in a hidden layer l accepts a vector of inputs, which is either $\mathbf{x}$ from the input layer to the first hidden layer or $\mathbf{h}^{(l-1)}$ from the preceding hidden layer, and performs an affine transformation which outputs $z_i^{(1)}$ (Goodfellow et al., 2016). For a unit in the first hidden layer, the transformation is $z_i^{(1)} = \mathbf{w}_i^{(1)\top}\mathbf{x} + b_i^{(1)}$ — that is, the inputs are multiplied with their respective weights, $\mathbf{w}_i^{(1)}$, and added a bias, $b_i^{(1)}$. In the following hidden layers, $\mathbf{h}^{(l-1)}$ is received as input, hence $z_i^{(l)} = \mathbf{w}_i^{(l)\top}\mathbf{h}^{(l-1)} + b_i^{(l)}$. To map non-linear representations, $z_i^{(l)}$ is followed by applying a

[114] In our case, since we are predicting a continuous numeric price, we do not need to apply a non-linear transformation in the output layer.

non-linear *activation function*[115] g, such that the resulting activation $h_i^{(l)}$ passed forward to the units in the next layer in the network, is $h_i^{(l)} = g(z_i^{(l)})$. In fully-connected networks, most hidden layers are distinguished only by the choice of activation function g. While this is still a highly active area of research with few guiding theoretical principles, *Rectified Linear Units* (ReLU) are a good default choice (Goodfellow et al., 2016). These apply $g(z_i^{(l)}) = max\{0, z_i^{(l)}\}$, which replaces all negative activations with zero. This means that the first derivative of the rectifying operation is 1 wherever the neuron is active, while the second derivative is almost 0 everywhere. As such, the gradients are large and consistent wherever the neuron is active, which mitigates the problem of vanishing gradients (more on that later).

A.1.2 Gradient-Based Learning

The network's weights are *randomly initialised* as small, random values, hence the network initially out performing poorly (Chollet, 2018). After propagating a *batch* of N samples, randomly drawn from the training data, through the network, the network's loss function $C(\hat{\mathbf{y}}, \mathbf{y}) = C(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})$ computes a distance score between the resulting predictions $\hat{\mathbf{y}}$ and the true targets $\mathbf{y}$. By iteratively adjusting the parameters $\boldsymbol{\theta}$ based on repeated exposure to the training data, the network aims to minimise the loss. Each such round is referred to as an *epoch*, and it is completed when the algorithm has seen a number of samples corresponding to the full training data set.[116] Choosing the number of epochs involves a trade-off between opportunities for the network to learn, and computational cost.[117]

Similarly to how input data $\mathbf{x}$ is *fed forward* through the hidden layers to produce a prediction $\hat{y}$, the resulting error must be *fed backward* to compute each parameter's contribution to the loss. As neural networks are designed so that all constituent functions are differentiable, the *gradient* ∇ of the loss function C with respect to the parameters $\boldsymbol{\theta}$ — i.e. $\nabla_{\boldsymbol{\theta}} C$ — can be calculated by applying the *chain rule*.[118] The gradient contains all partial derivatives of the loss function with respect to each network parameter, which an optimiser algorithm uses in combination with a *learning rate* to adjust the parameters so that it descends the loss function.[119] It then repeats this until some stopping condition is met.

[115]Sometimes referred to as a *transfer function*.

[116]Each sample may be drawn more than one time, hence each *unique* observation in the training set may not have been propagated through the network at the completion of one epoch.

[117]Despite a trade-off, there is an upper limit to how many epochs are desirable, as the network will start overfitting to the training data the higher the number of epochs.

[118]The chain rule computes the derivative of composite functions as $\nabla_{\mathbf{x}} z = (\frac{\partial \mathbf{y}}{\partial \mathbf{x}})^{\top} \nabla_{\mathbf{y}} z$.

[119]The approach is therefore commonly referred to as *gradient descent*.

A.1.3 Optimisation Algorithms

Optimising algorithms in machine learning usually decompose the loss function as a sum over training examples, where weight updates are computed by estimating the expected value of the loss over a subset of the training data. Most algorithms used in deep learning are *mini-batch* methods, often referred to as *stochastic* methods, which use more than one observation but less than the entire set at a time. A larger *batch size* provides a more accurate estimate of the gradient and requires less computational time per parameter update, while a smaller batch can offer a regularising effect (Wilson & Martinez, 2003). Though a larger batch is generally preferable, it also requires more hardware as more data needs to fit alongside the model parameters in memory. When training networks on a GPU, it is common to select a power of 2 as the batch size — usually between 32 and 256 (Goodfellow et al., 2016).

The *stochastic gradient descent* (SGD) family of optimiser algorithms draw a mini-batch of B samples from the training set, then computes the gradient $\hat{g}$ of the loss with respect to the parameters for that mini-batch as $\hat{g} = \frac{1}{B}\nabla_\theta \sum_{b=1}^{B} C(f(\mathbf{x}^{(b)}, \boldsymbol{\theta}), y^{(b)})$. Each iteration k has a learning rate ϵ_k. To update the parameters $\boldsymbol{\theta}$, $\hat{g}$ is multiplied by the learning rate, and the sum is subtracted from the previous parameter values, i.e. $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} - \epsilon_k \hat{g}$. Setting an appropriate learning rate is paramount to model convergence, which is consistently among the hardest design choices (Goodfellow et al., 2016). *Adaptive* learning rate methods use separate rates for each parameter and adjust these during training, in addition to incorporating concepts like *decay* and *momentum*. Among these, *ADAM* (Kingma & Ba, 2014) is a popular choice. The default values are generally a safe choice (Goodfellow et al., 2016), hence we use the default ADAM throughout. In addition, we specify that the learning rate should be halved when the training loss plateaus.

A.1.4 Fighting Overfitting

As with any other machine learning method, neural networks entail a trade-off between *optimisation*, i.e. performing optimally on the training data, and *generalisation*, i.e. performing well on unseen data (Chollet, 2018). *Overfitting* occurs when the network starts using rote memorisation to reduce its training loss, rather than capturing the underlying patterns of the data-producing process. The best remedy is procuring additional training data such that the model will generalise better (Chollet, 2018). Alternatively, *regularisation* methods constrain how much and/or what information the network can store, which forces it to prioritise the more salient patterns in the data. This should presumably help the network generalise better, and ideally not come at the expense of higher training loss (Goodfellow et al., 2016).

Introducing weight regularisation, such as L^1 and L^2 *norm penalties*,[120] is one traditional approach.[121] An alternative is to reduce the *capacity* of the network, i.e. the number of learnable parameters, though there is a risk of the network under-performing. Unfortunately, there are few theoretical guiding principles (Chollet, 2018), so the most reliable approach is experimentation. Another method is *early stopping*, where the algorithm records the network parameters whenever validation loss improves, and uses those that generalised best rather than the latest when training ends (Goodfellow et al., 2016). We use this method, which is sometimes also referred to as the *minimum validation loss* model selection method (H. Chen et al., 2017).

Dropout is a computationally cheap technique applied to hidden layers during training, where it randomly *drops* a share — usually between 0.2, and 0.5 — of neurons on the forward and backward pass for each set of inputs (Chollet, 2018). The network is forced to not rely excessively on a few features, but rather disperse its weights and learn redundant mappings. A network with dropout is also effectively smaller, which constrains its capacity and indirectly regularises it. Using dropout is also analogous to *bagging*[122] a plethora of neural networks, which improves generalisation but would usually be extremely expensive (Goodfellow et al., 2016). Dropout has been empirically found to outperform alternative cheap regularisation methods, can be combined with other methods, and does not limit the possible architectures (N. Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). As such, we use it extensively in our networks.

A.1.5 Residual Connections

While adding depth increases a network's capacity, it also exacerbates the problem of *vanishing gradients*: the gradient can explore or become infinitesimally small through repeated multiplication as it is back-propagated through a deep sequence of layers, which hampers training and

[120] This approach adds a parameter norm penalty $\Omega(\theta)$ to the loss function (Goodfellow et al., 2016). L^2 adds the norm penalty $\Omega(\theta) = \frac{1}{2}||w||_2^2$ which drives the weights towards zero, while L^1 adds $\Omega(\theta) = ||w||_1$ which may cause a subset of weights to become zero, resulting in higher sparsity than L^2 regularisation.

[121] After comprehensive experimentation, we found that these excessively constrained optimisation without a proportionate improvement in generalisation, hence we do not employ them.

[122] *Bootstrap aggregation*, or simply *bagging*, consists of repeated random resampling of the data, and fitting a model on each such subsample. For a given set of inputs, the predictions from these separate models are tallied up, and a final aggregated prediction made — for instance by majority vote or averaging (Hastie et al., 2009).

impedes convergence. *Residual connections* make the outputs from earlier layers available to later layers, by adding the input to a given stack of consecutive layers to its output (He et al., 2015). This effectively creates a shortcut in the network that facilitates the propagation of both forward and backward signals (Chollet, 2018). Also, rather than hoping the stack accurately fits an underlying ideal mapping $H(\mathbf{x})$, we instead let it fit a residual function $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$.[123] He et al. (2015) demonstrate that deep networks find learning such perturbations relative to identity mappings easier than having to learn an entirely new mapping.

Deep residual networks have become popular for their impressive performance and training convergence (He et al., 2016). Residual connections add no extra parameters and therefore come at no added computational cost, making them even more attractive for practitioners (He et al., 2015). The residual block — the stack of similar hidden layers at the end of which the stack's input signal is added — proposed by He et al. (2015) applies batch normalisation and ReLU after each layer except for the last, where ReLU is instead applied after the shortcut has been added. In contrast, Ioffe and Szegedy (2015) apply batch normalisation after ReLU is applied, hence there seems to be no consistent guideline for their ideal ordering. He et al. (2016) propose an alternative residual block design where batch normalisation and ReLU activations are applied before each constituent layer's activations, rather than after. They find that such *pre-activation* makes the network converge around a slightly higher training loss, but also achieves a lower test loss — presumably due to the regularising effect of batch normalisation.

A.1.6 Batch Normalisation

Although we centre input data around zero with unit variance to help the networks learn and generalise better, such *standardisation* should be a concern after every transformation; there is no reason to expect that a layer's output is Gaussian because its input is (Chollet, 2018). Many activation functions are *saturating* in that they coerce values of a linear input beyond some threshold(s) to a confined range, in which case the differential is virtually zero, hence gradients could vanish. Changes in the distribution of activations due to changing weights during training, known as *covariate shift*, are likely to move many activations into saturating regions, which slows training and hampers convergence (Ioffe & Szegedy, 2015). ReLUs are a popular remedy since they are only saturating in the negative direction, but do not entirely eliminate the probability

[123]If the dimensions of $F(\mathbf{x})$ and $\mathbf{x}$ are different, we can simply perform a linear projection with no activation function or bias to ensure the same number of dimensions.

of the optimiser becoming stuck in a saturation region.

Batch normalisation, implemented in Keras[124] as a type of layer, records an exponential moving average of the feature-wise means and variances seen during training (Chollet, 2018), and applies a transformation at the mini-batch level that centres the preceding layers' activations around zero with unit variance. The layer standardises each feature independently, and introduces pairs of parameters that scale and shift the standardised values to maintain the network's representational power (Ioffe & Szegedy, 2015). This makes the distribution of inputs to the subsequent layer more stable, which helps gradient propagation and accelerates training — thereby enabling deeper networks (Chollet, 2018). Reducing covariate shift via batch normalisation has been found to mitigate vanishing gradients and accelerate training (Ioffe & Szegedy, 2015).

A.1.7 Checkpoint Ensembling

The complex hypothesis space of neural networks (Chollet, 2018) allows them to capture a great diversity of interactions between features, but this also makes them susceptible to mapping suboptimal relationships between features due to sampling noise or insufficient data (H. Chen et al., 2017). In addition to regularisation, another popular approach is to use *ensembling* methods, where different models[125] are combined to make predictions that are more accurate than those of any of its constituents (Dietterich, 2000). The simplest ensembling method is arguably *random initialisation ensembling* (RIE), where the same architecture is trained multiple times with different random weight initialisations, and the predictions simply averaged. RIE is less effective than ensembling different classes of models as it is hard for a single architecture to capture the same range of feature representations, though neural networks' stochastic nature attenuates this limitation (H. Chen et al., 2017; Goodfellow et al., 2016).

As an alternative, *checkpoint ensembles* (CE) combine predictions from the best *checkpoints* of weights from all epochs of a single model's training run, as measured on the validation set (H. Chen et al., 2017). As the network traverses its parameter space, the combination of the gradient and learning rate sends it meandering around local optima. The model corresponding to the weights at the end of each epoch may be particularly adept at making predictions in its respective subspace of prediction problems. By combining some of the best such checkpoints, one might get a model that generalises better to the complete space of all possible prediction

[124]See documentation: `https://keras.io/layers/normalization/#batchnormalization`

[125]Ideally, these combine different classes of models with very different hypothesis spaces in order to capture a more diverse set of efficacious feature representations with relatively independent errors.

problems (H. Chen et al., 2017). Checkpoint ensembling has been found to improve performance over selecting only the best weights (H. Chen et al., 2017), by capturing parts of the gains from traditional ensembling techniques, without the computational cost of repeated training runs.

A.2 Recurrent Neural Networks

Feed-forward networks treat time series as a single temporal point by collapsing time steps of the same features into separate features. They also cannot be shown a series of successive samples in sequence, as each sample is processed independently (Chollet, 2018). In contrast, recurrent networks are specialised to process sequential input data, and can incorporate much longer sequences than would be practical with less specialised network types (Goodfellow et al., 2016). Due to *parameter sharing*, recurrent networks also share the same parameter values across time steps, which makes them able to handle variable-length input or output sequences. It also enables the network to temporally generalise its mappings, as a recognised pattern will be picked up no matter where in the sequence it appears (Chollet, 2018). In contrast, a fully-connected network will only pick up a pattern if it occurs in the exact same place in the input sequence as before.

Recurrent networks iterate through a given sequence while keeping an internal *state* of information on what it has seen (Chollet, 2018). Ideally, this state is reset after each sequence is processed, so that a single sequence is treated as one sample. In practice, the use of mini-batches means state is reset after each batch. Hence, there is a certain trade-off between representational power and computational cost. The network is in other words *recurrent* in that it has connections that make available the state from previous activations in the sequence when making a prediction. In a sense, the network has memory.

For a given window length, or *lookback*, a recurrent network considers a sequence of $\tau = lookback + 1$ input vectors $\mathbf{x}^{(t)}$, where $t = 1, ..., \tau$. The *unfolded* recurrence after t steps, i.e. the state at time t, can be represented by a function $g^{(t)}$ that takes the whole preceding sequence of inputs $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, ..., \mathbf{x}^{(1)})$, i.e. $\mathbf{h}^{(t)} = g^{(t)}(\mathbf{x}^{(t)}, ..., \mathbf{x}^{(1)}) = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$, where $\mathbf{h}$ denotes that the state is the network's hidden units (Goodfellow et al., 2016). To predict a price given a preceding sequence of input values, the network learns to use the state $\mathbf{h}^{(t)}$ as a lossy summary[126] of what the network deems to be the relevant parts of this history. The function $g^{(t)}$

[126]The variable-length sequence of preceding inputs $(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)})$ is reduced to a fixed-size vector $\mathbf{h}^{(t)}$, hence some information is necessarily lost.

takes the whole preceding sequence of inputs and would amount to fitting a separate such transformation for each time step, but unfolding it makes it equivalent to the repeated application of some function f. Hence, the network uses the same transition function f between states, rather than relying on a history of states. This means the network can trivially generalise to arbitrary sequence lengths, and is also much easier to train.

Computing the gradient through a recurrent network is done much the same way by applying the traditional back-propagation algorithm to the unrolled computational graph, called *back-propagation through time* (BPTT). Recurrent networks can be designed for a wide variety of cases, but we use them to take a fixed-length input sequence to make a single prediction — for each sample in the dataset. Like fully-connected networks, recurrent networks can sometimes generalise better with the added representational capacity afforded by stacking additional layers, but is less certain to do so.

A.2.1 Bidirectional Recurrent Networks

Most recurrent networks are causal in that they evaluate sequences of inputs in chronological order. *Bidirectional* networks also iterate in the opposite direction, which can improve performance on problems where the entire sequence is relevant or where information from two distant points may be needed in conjunction to disambiguate some signal (Goodfellow et al., 2016). Put simply, they combine a traditional recurrent network that steps chronologically through time with another that steps in the opposite direction. This has significantly improved performance on problems such as reading handwriting and speech recognition (Brownlee, 2017). We try bidirectional recurrent networks in case they pick up some valuable patterns by also evaluating the inputs in the reverse chronological order. In our case, this means stepping backwards from time τ to 1 rather than evaluating the complete temporal sequence, as that would incorrectly incorporate information from the future.

A.2.2 Long Short-Term Memory

Some of the most successful contemporary sequence models are *gated recurrent neural networks*, such as the *long short-term memory* (LSTM) networks and networks using *gated recurrent units* (GRU). Such networks were designed with paths whose derivatives do not vanish in an attempt to address the problem of vanishing gradients in recurrent networks. They do this by introducing connection weights that may change at each time step, so that the network can learn to choose when to reset its state to zero, rather than it having to be defined manually (Goodfellow et al., 2016).

LSTM introduces *self-loops* which facilitate propagation of the gradient in very long sequences (Hochreiter & Schmidhuber, 1997), and where the weight on this self-loop is contextual rather than constant (Gers, Schmidhuber, & Cummins, 2000). This weight is *gated*, i.e. controlled by another unit, so that its value can be changed dynamically (Goodfellow et al., 2016). In a sense, the LSTM adds another track to *carry* information, so that older signals do not gradually vanish as the network continues iterating down the sequence (Chollet, 2018). While simpler recurrent networks have the outer state recurrence, LSTM "smart" nodes (Brownlee, 2017) therefore also have the internal self-loop, more parameters, and a set of gating units that contextually adapt information flow (Goodfellow et al., 2016).

A.3 Image Processing: Convolutions and Pooling

Similarly to how recurrent networks are specialised for sequential input data, *convolutional* networks are specialised for grids of values, such as 1-dimensional time series or 2-dimensional images (Goodfellow et al., 2016). Compared to fully-connected layers that learn global patterns, convolutional layers learn local patterns (Chollet, 2018). Rather than having every input unit interact with every output unit, convolutional layers use fewer operations. In addition to reduced memory usage and computational cost, convolution layers also have other desirable properties: they learn *translation invariant* patterns, meaning it learns to recognise a pattern in an image no matter where it appears; they learn spatial hierarchies of patterns, from initially small, local patterns to larger, more abstract and increasingly complex patterns made up of the underlying ones (Goodfellow et al., 2016).

An image consists of two spatial dimensions (height and width) and a depth of some *channels*; a grey-scale image has one channel, whereas a colour image has three: red, green, and blue. A convolution operation slides some *kernel*, also called a *receptive field*, across the input grid, and computes some transformations at every possible location. Typical kernel dimensions are 3×3 or 5×5 with a *stride* — the number of steps the convolution takes when extracting the next patch — of 1[127] (Chollet, 2018). The key is that these transformations are the same for each

[127]With no stride (stride of 1), successive receptive fields will overlap somewhat, which is usually desirable when searching for local patterns.

extracted patch, i.e. that the convolution looks for the same learned patterns in each spot. The resulting vectors are applied a non-linear activation function, then spatially reassembled into a *feature map* where the "pixel" values are now these activations (Chollet, 2018).

While the original image has some given channel axis depth, the resulting feature maps from each convolutional layer can be arbitrarily deep. The width and height can shrink depending on the *padding* and kernel dimensions, but the depth depends on the number of *filters* specified. Filters encode different aspects of the input map or image, the number of which is specified as one of the layer's parameters alongside the kernel dimensions, type of padding, and activation function — among others. As such, each filter learns to look for a certain pattern, and outputs a map with its activations at all possible locations in the input map (Chollet, 2018). The stack of these maps constitute the layer's output feature map.

After a non-linear activation function has been applied to the output of the convolutions, it is typically followed by *pooling* to further refine the map to a form of summary statistic (Goodfellow et al., 2016). A popular method is *max pooling*, which outputs the maximum value found in each channel of rectangular grids in the input map. It is typically used with a 2×2 kernel with stride 2, to effectively downsample the feature map in both dimensions by a factor of two (Chollet, 2018). This reduces the number of feature map coefficients to compute, and more directly promotes hierarchical filters by making successive convolutions span increasingly larger spatial areas (Chollet, 2018).

A.3.1 Convolutional Pyramids

New record-setting convolutional architectures are announced with such frequency that defining a generally "best" architecture is impractical (Goodfellow et al., 2016). Instead, we design convolutional networks following the guiding principle of *convolutional pyramids*: large receptive fields are often necessary for the network's structured understanding of low-level image processing tasks, but stacking sufficient layers or using adequately wide kernels can become cost-prohibitive (Shen, Chen, Tao, & Jia, 2017). Starting with the spatially wide (and high) input images with few channels, we stack successive convolution and pooling layers that progressively downsample the spatial dimensions while adding filters, thereby resulting in a *pyramid* of decreasing spatial footprint and increasing depth. This enables a spatial-filter hierarchy of increasingly wide fields, without sacrificing computational efficiency (Shen et al., 2017).

B Additional Information about Elbas

B.1 Prices in the Regulating Market

Figure 23 shows how the regulating price is derived. If there is a need for more power to be delivered in a given hour, the TSOs set the price so that up-regulation occurs (price above the market/Elspot price), which is the situation shown in the figure. If, on the other hand, there is a need for less power to be delivered in a given hour, the price is set so that down-regulation occurs (price below the market/Elspot price). Thus, the regulating prices are used to incentivise more or less power being produced in each hour of delivery. If consumption exceeds planned production, a higher price incentivises more producers to deliver power in a given hour. If production exceeds consumption, a lower price incentivises more producers to buy cheap power from other producers instead of generating their own, and as such, reducing overall power production.

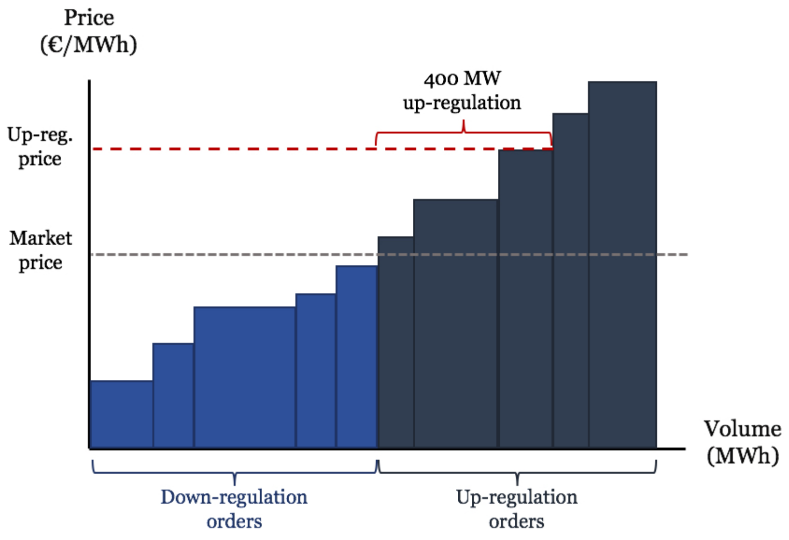


Figure 23: Up- and down-regulation in the regulating market. The illustration is based on: Nord Pool (n.d.).

Table 7, which is based on Table 4 in Scharff and Amelin (2016), shows how production and consumption imbalances are settled.[128] For consumption imbalances, one price is used for set-

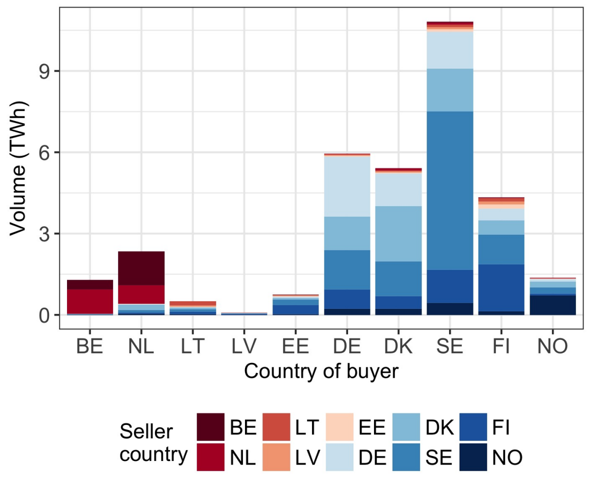
[128]Production imbalances are calculated as the difference between the latest submitted production plan after settlements in the Elspot market and the observed production. Consumption imbalances are calculated as the latest submitted production/consumption plan after Elspot, adjusted for trading in Elbas, and observed consumption (Scharff & Amelin, 2016). The binding plan may be updated until 45 minutes before power delivery. Any changes to the production plan that is not corrected through Elbas trading, goes into the consumption imbalance, which implies that the production imbalance only reflects the deviations during the delivery hour.

tlement and it reflects the dominating direction of regulating power in a given hour. That is, it equals the up-regulating price if the dominating direction is up, and the down-regulating price if the dominating direction is down. For production imbalances, on the other hand, a two-price system is applied in settlements. Hence, if the dominating direction is up, producers with a production deficit pay the up-regulation price, while producers with a production surplus receive the day-ahead price. Conversely, if the dominating direction is down, the producer with a deficit pay the day-ahead price while the producers with a surplus receive the down-regulation price. The purpose of the two-price system is to ensure that no market participants may profit from balancing in real-time compared to trading the corresponding power in the day-ahead market (Scharff & Amelin, 2016).

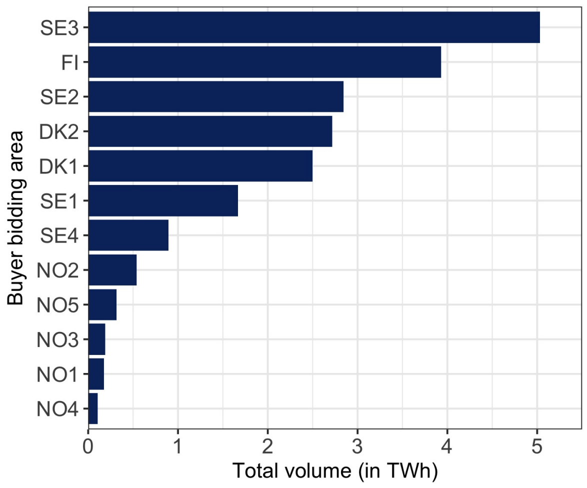
Own position in regulating market	**Up-regulation**	**No regulation**	**Down-regulation**
Production deficit (= prod. purchase)	Pay: $P_{up} \times E_{deficit}$	Pay: $P_{da} \times E_{deficit}$	Pay: $P_{da} \times E_{deficit}$
Production excess (= prod. sale)	Receive: $P_{da} \times E_{excess}$	Receive: $P_{da} \times E_{excess}$	Receive: $P_{down} \times E_{excess}$
Consumption deficit	Pay: $P_{up} \times E_{deficit}$	Pay: $P_{da} \times E_{deficit}$	Pay: $P_{down} \times E_{deficit}$
Consumption excess	Receive: $P_{up} \times E_{excess}$	Receive: $P_{da} \times E_{excess}$	Receive: $P_{down} \times E_{excess}$

Table 7: Imbalance price settlement for production imbalances (two-price system) and consumption imbalances (one-price system). Source: Scharff and Amelin (2016).

B.2 Elbas Trade Volumes per Area and Country



(a) Trade between Elbas countries



(b) Total volume per buyer bidding area

Figure 24: Volume traded over the period November 2, 2011 to December 31, 2017.

B.3 Trading Activity across Trade Hours

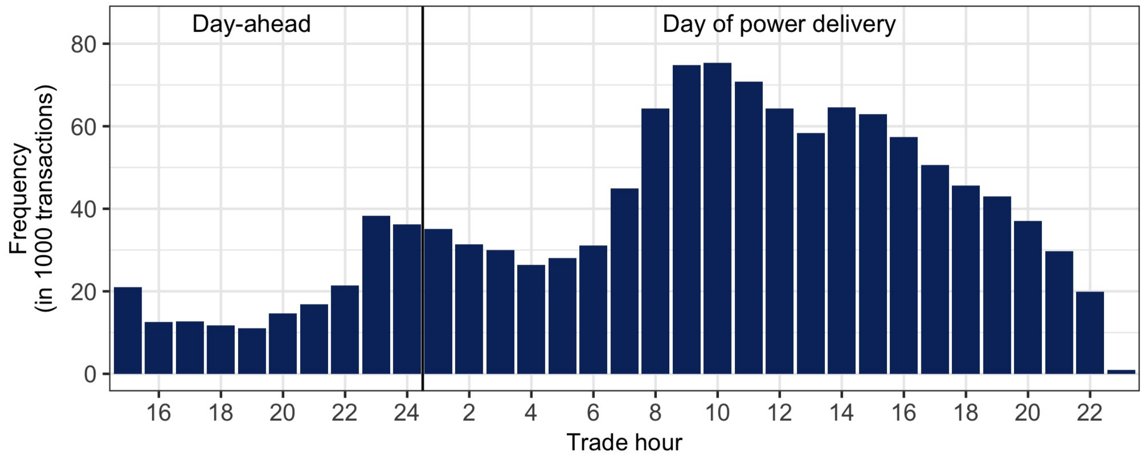


Figure 25: Number of trades (measured in 1000 transactions) per trade hour. The data includes trades over the period November 2, 2011 to December 31, 2017 with a buyer from a Nordic bidding area.

C Standardisation or Normalisation?

Many methods in machine learning — in particular neural networks — have been found to perform better when continuous numeric variables have similar scales (Hastie et al., 2009). Two common approaches are, based on the training data, to either scale variables to be in the range $[0, 1]$ or to centre each variable around zero mean and with unit variance. The nomenclature is not consistent in academia, but we refer to these methods as *normalisation* and *standardisation*, respectively. Specifically, each value $x_{i,j}$ in observation i of variable j can be standardised by utilising the mean and standard deviation of j in the training set: $s_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j}$. Similarly, they can instead be normalised via the minimum and maximum values of j in the training set: $n_{i,j} = \frac{x_{i,j} - min_j}{max_j - min_j}$. *Normalising* is simpler and makes no assumptions on the distribution of each variable, but is very sensitive to outliers as these could determine the minimum and maximum values used to scale all values between 0 and 1. Similarly, one would like to know a priori that no values of greater magnitude appear in the out-of-sample set(s); though not a requirement, if this is not generally met the point of scaling is somewhat defeated. In contrast, *standardising* is much less susceptible to outliers, but makes the strong assumption that each variable follows a Gaussian distribution. It still works if not, but model performance might deteriorate if highly non-normal variables are standardised (Brownlee, 2017). A possible approach is to combine normalisation and standardisation, determining on a per-variable basis which to use depending on the variable's distribution and/or presence of outliers.

C.1 Distribution Plots

These plots show the training set distribution for some examples of the continuous numeric variables, with a superimposed Gaussian probability density function (black curve) using the corresponding variable's test set mean and standard deviation. If the curve perfectly envelops the actual distribution, then the variable is perfectly Gaussian, and standardising it is exactly the right approach. If the variable is clearly not normally distributed, then standardising it may be detrimental to performance, and it is perhaps better to normalise it.

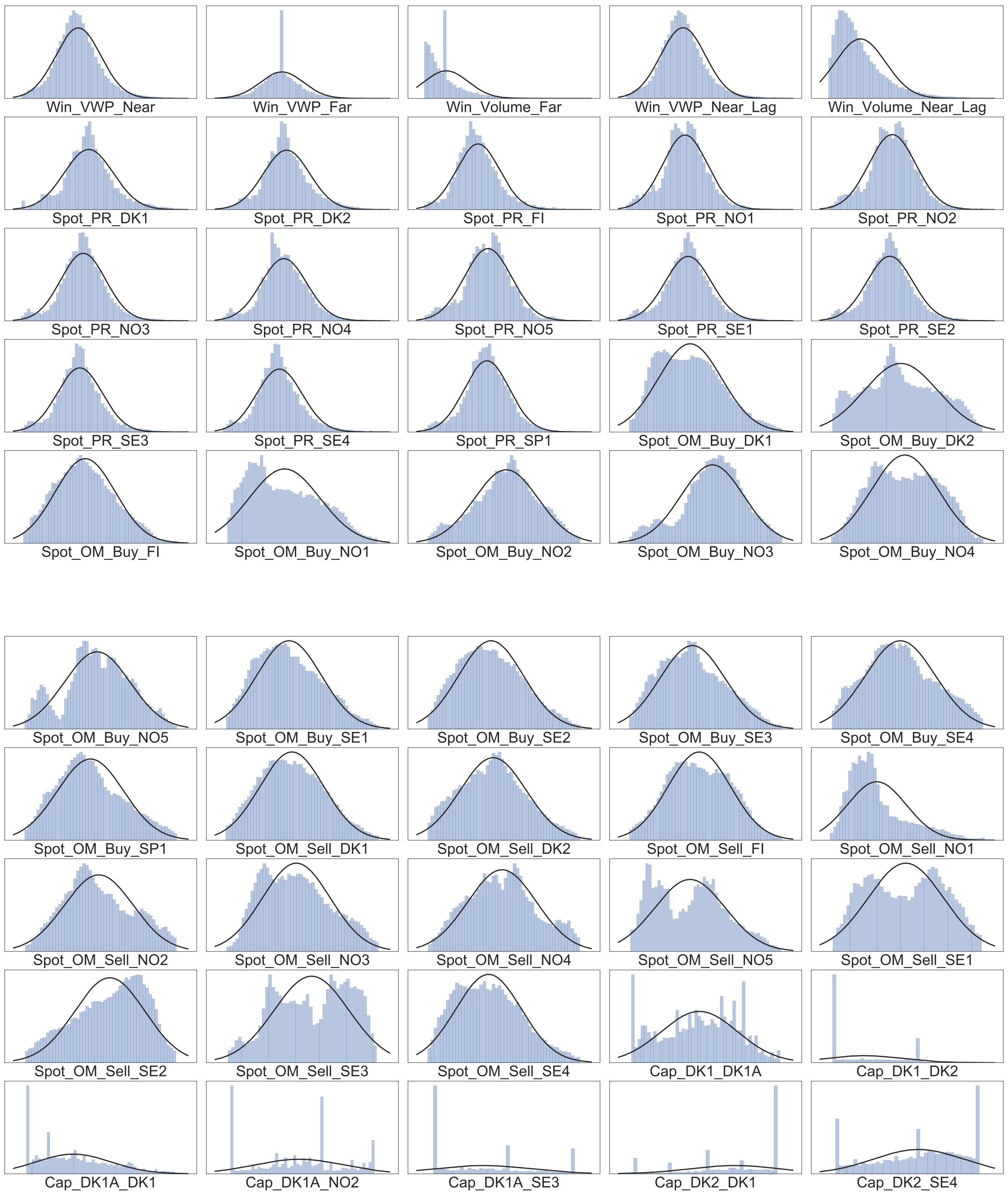
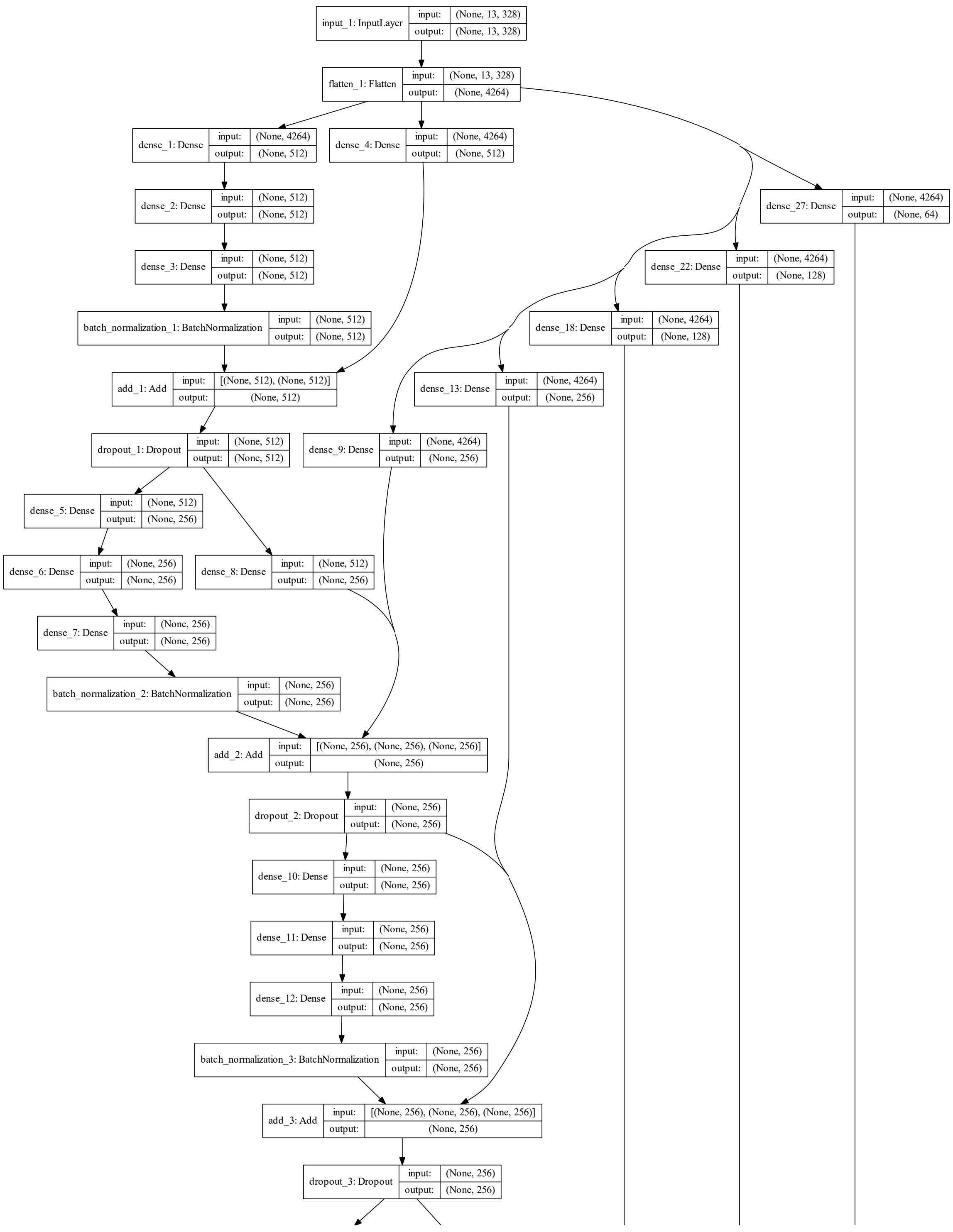


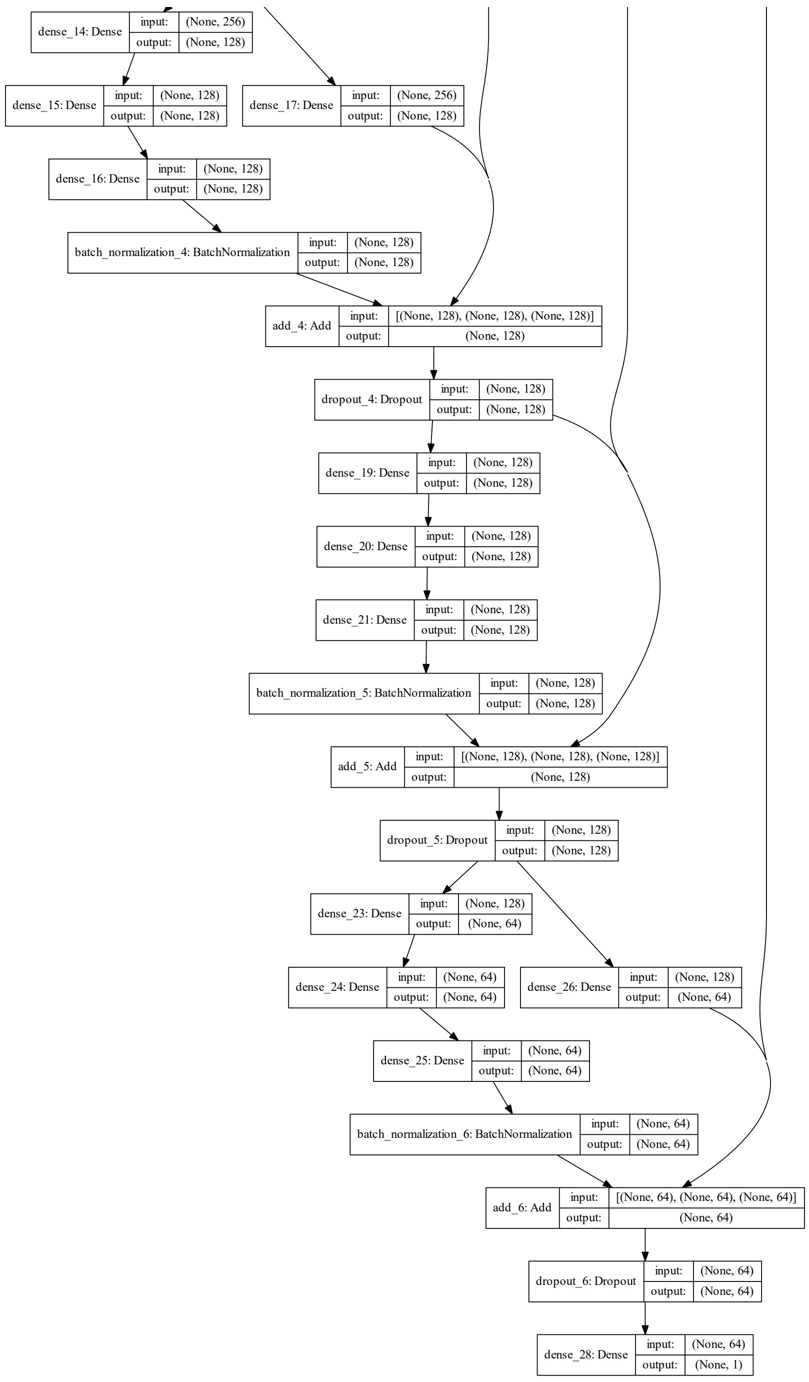
Figure 26: Plots showing the training set distribution for some examples of the continuous numeric variables, with a superimposed Gaussian probability density function (black curve) using the corresponding variable's test set mean and standard deviation.

D Deep Learning Models

D.1 ResNet 1

Model topology


input_1: InputLayer | input: (None, 13, 328) | output: (None, 13, 328)
flatten_1: Flatten | input: (None, 13, 328) | output: (None, 4264)
dense_1: Dense | input: (None, 4264) | output: (None, 512)
dense_4: Dense | input: (None, 4264) | output: (None, 512)
dense_27: Dense | input: (None, 4264) | output: (None, 64)
dense_2: Dense | input: (None, 512) | output: (None, 512)
dense_22: Dense | input: (None, 4264) | output: (None, 128)
dense_3: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_1: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_18: Dense | input: (None, 4264) | output: (None, 128)
add_1: Add | input: [(None, 512), (None, 512)] | output: (None, 512)
dense_13: Dense | input: (None, 4264) | output: (None, 256)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
dense_9: Dense | input: (None, 4264) | output: (None, 256)
dense_5: Dense | input: (None, 512) | output: (None, 256)
dense_6: Dense | input: (None, 256) | output: (None, 256)
dense_8: Dense | input: (None, 512) | output: (None, 256)
dense_7: Dense | input: (None, 256) | output: (None, 256)
batch_normalization_2: BatchNormalization | input: (None, 256) | output: (None, 256)
add_2: Add | input: [(None, 256), (None, 256), (None, 256)] | output: (None, 256)
dropout_2: Dropout | input: (None, 256) | output: (None, 256)
dense_10: Dense | input: (None, 256) | output: (None, 256)
dense_11: Dense | input: (None, 256) | output: (None, 256)
dense_12: Dense | input: (None, 256) | output: (None, 256)
batch_normalization_3: BatchNormalization | input: (None, 256) | output: (None, 256)
add_3: Add | input: [(None, 256), (None, 256), (None, 256)] | output: (None, 256)
dropout_3: Dropout | input: (None, 256) | output: (None, 256)



dense_14: Dense | input: (None, 256) | output: (None, 128)
dense_15: Dense | input: (None, 128) | output: (None, 128)
dense_17: Dense | input: (None, 256) | output: (None, 128)
dense_16: Dense | input: (None, 128) | output: (None, 128)
batch_normalization_4: BatchNormalization | input: (None, 128) | output: (None, 128)
add_4: Add | input: [(None, 128), (None, 128), (None, 128)] | output: (None, 128)
dropout_4: Dropout | input: (None, 128) | output: (None, 128)
dense_19: Dense | input: (None, 128) | output: (None, 128)
dense_20: Dense | input: (None, 128) | output: (None, 128)
dense_21: Dense | input: (None, 128) | output: (None, 128)
batch_normalization_5: BatchNormalization | input: (None, 128) | output: (None, 128)
add_5: Add | input: [(None, 128), (None, 128), (None, 128)] | output: (None, 128)
dropout_5: Dropout | input: (None, 128) | output: (None, 128)
dense_23: Dense | input: (None, 128) | output: (None, 64)
dense_24: Dense | input: (None, 64) | output: (None, 64)
dense_26: Dense | input: (None, 128) | output: (None, 64)
dense_25: Dense | input: (None, 64) | output: (None, 64)
batch_normalization_6: BatchNormalization | input: (None, 64) | output: (None, 64)
add_6: Add | input: [(None, 64), (None, 64), (None, 64)] | output: (None, 64)
dropout_6: Dropout | input: (None, 64) | output: (None, 64)
dense_28: Dense | input: (None, 64) | output: (None, 1)

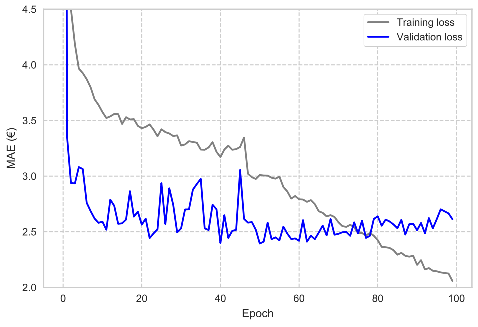
Figure 27: Model topology *ResNet 1*.

Results per run

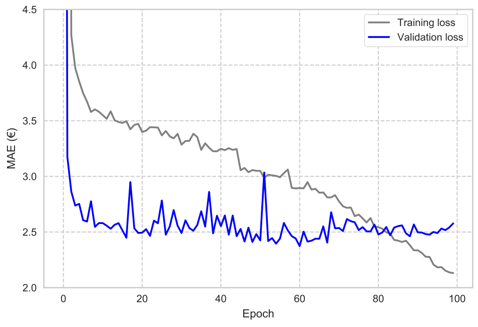
Run	Validation				Test			
	MVL		CE		MVL		CE	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	2.3942	4.4517	2.3623	4.4401	2.7122	**4.7409**	**2.6922**	4.6670
2	**2.3738**	4.4256	2.3642	4.4196	2.6793	4.7478	2.6894	4.7581
3	2.3797	**4.4474**	**2.3636**	**4.4267**	2.7477	4.7217	2.7108	**4.7222**
4	2.3710	4.4964	2.3654	4.4788	2.7074	4.7887	2.6942	4.7970
5	2.3530	4.4014	2.3403	4.4190	**2.7082**	4.5967	2.6671	4.6220
Mean	2.3743	4.4445	2.3592	4.4368	2.7110	4.7192	2.6907	4.7133
Std.	0.0149	0.0352	0.0106	0.0249	0.0244	0.0727	0.0156	0.0699

Table 8: MAE and associated RMSE for five runs of *ResNet 1*. The median performer is highlighted. All values are in EUR/MWh.

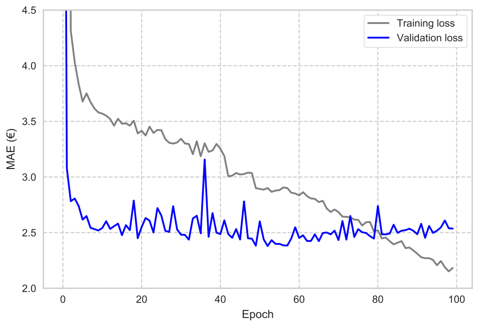
Training plots



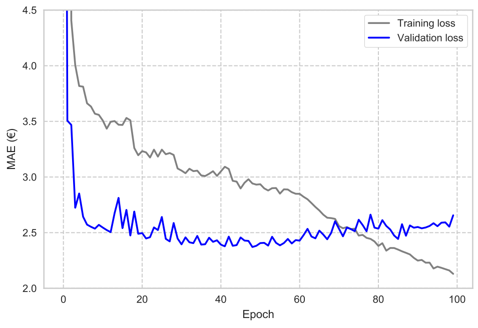
(a) Run 1



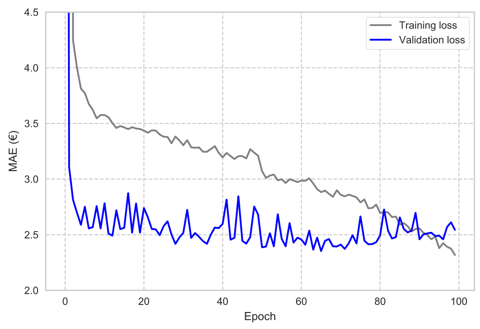
(b) Run 2



(c) Run 3



(d) Run 4

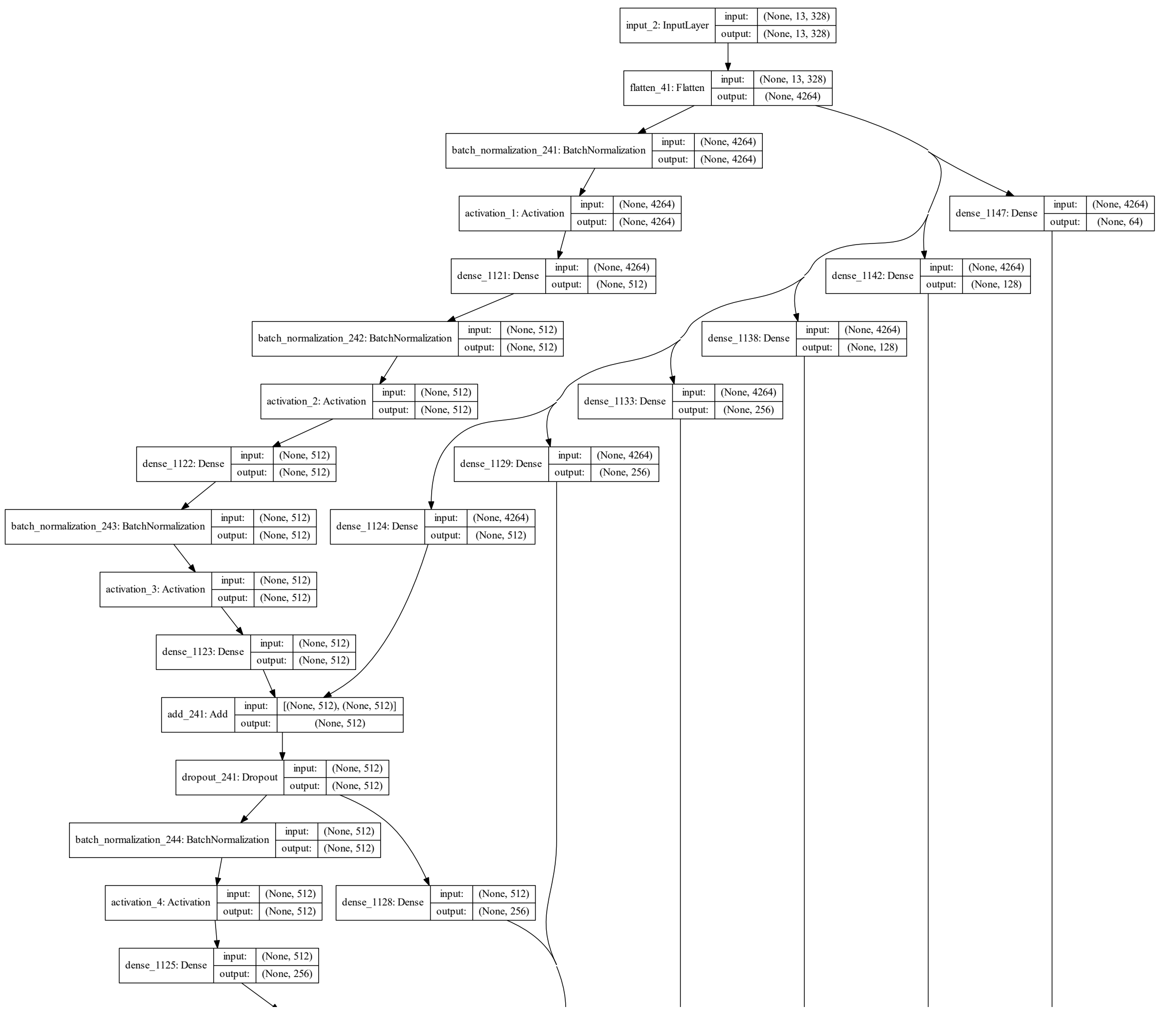


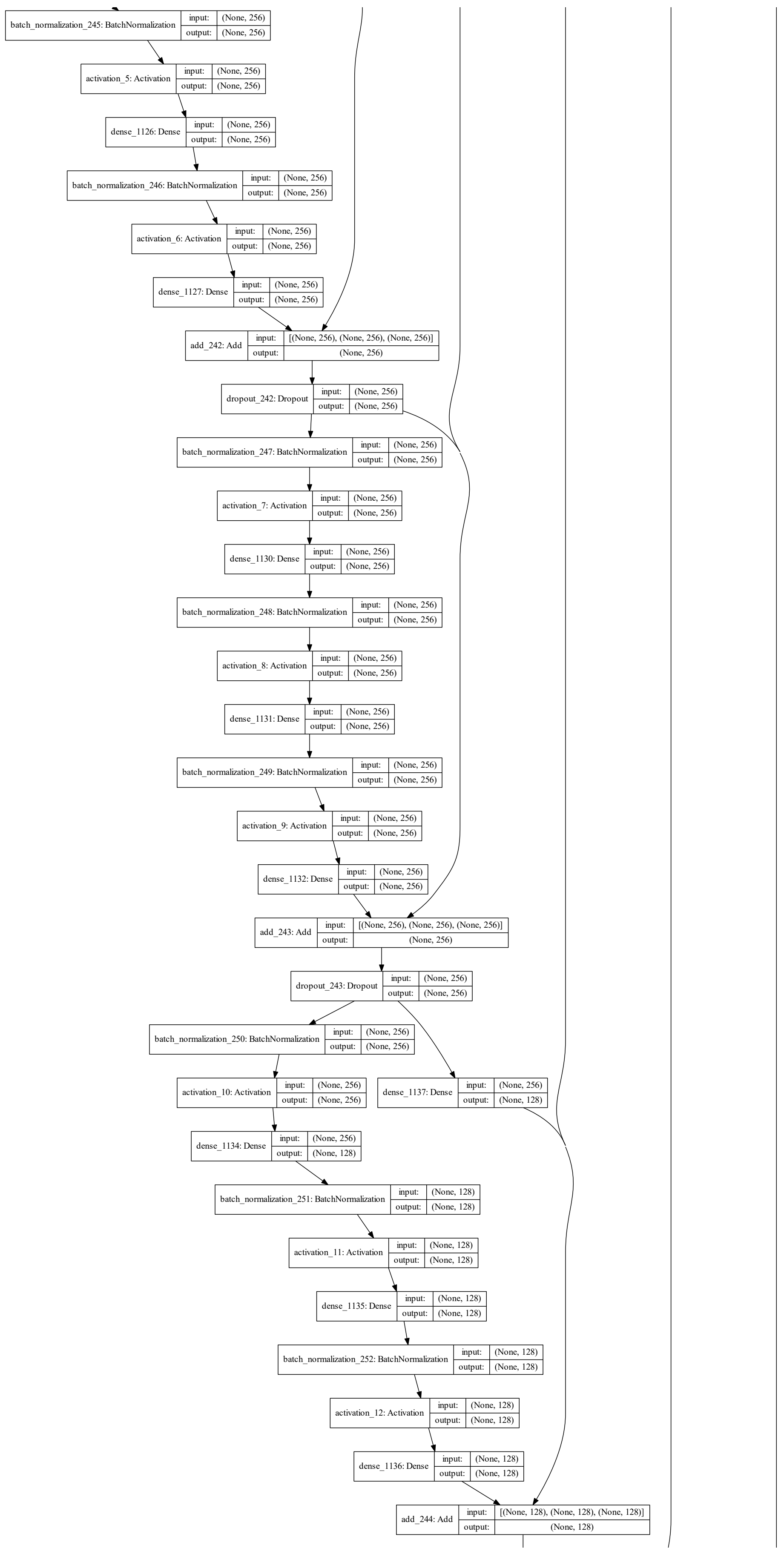
(e) Run 5

Figure 28: Training plots *ResNet 1*.

D.2 ResNet 2

Model topology


input_2: InputLayer | input: (None, 13, 328) | output: (None, 13, 328)
flatten_41: Flatten | input: (None, 13, 328) | output: (None, 4264)
batch_normalization_241: BatchNormalization | input: (None, 4264) | output: (None, 4264)
activation_1: Activation | input: (None, 4264) | output: (None, 4264)
dense_1147: Dense | input: (None, 4264) | output: (None, 64)
dense_1121: Dense | input: (None, 4264) | output: (None, 512)
dense_1142: Dense | input: (None, 4264) | output: (None, 128)
batch_normalization_242: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_1138: Dense | input: (None, 4264) | output: (None, 128)
activation_2: Activation | input: (None, 512) | output: (None, 512)
dense_1133: Dense | input: (None, 4264) | output: (None, 256)
dense_1122: Dense | input: (None, 512) | output: (None, 512)
dense_1129: Dense | input: (None, 4264) | output: (None, 256)
batch_normalization_243: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_1124: Dense | input: (None, 4264) | output: (None, 512)
activation_3: Activation | input: (None, 512) | output: (None, 512)
dense_1123: Dense | input: (None, 512) | output: (None, 512)
add_241: Add | input: [(None, 512), (None, 512)] | output: (None, 512)
dropout_241: Dropout | input: (None, 512) | output: (None, 512)
batch_normalization_244: BatchNormalization | input: (None, 512) | output: (None, 512)
activation_4: Activation | input: (None, 512) | output: (None, 512)
dense_1128: Dense | input: (None, 512) | output: (None, 256)
dense_1125: Dense | input: (None, 512) | output: (None, 256)


batch_normalization_245: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_5: Activation | input: (None, 256) | output: (None, 256)
dense_1126: Dense | input: (None, 256) | output: (None, 256)
batch_normalization_246: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_6: Activation | input: (None, 256) | output: (None, 256)
dense_1127: Dense | input: (None, 256) | output: (None, 256)
add_242: Add | input: [(None, 256), (None, 256), (None, 256)] | output: (None, 256)
dropout_242: Dropout | input: (None, 256) | output: (None, 256)
batch_normalization_247: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_7: Activation | input: (None, 256) | output: (None, 256)
dense_1130: Dense | input: (None, 256) | output: (None, 256)
batch_normalization_248: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_8: Activation | input: (None, 256) | output: (None, 256)
dense_1131: Dense | input: (None, 256) | output: (None, 256)
batch_normalization_249: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_9: Activation | input: (None, 256) | output: (None, 256)
dense_1132: Dense | input: (None, 256) | output: (None, 256)
add_243: Add | input: [(None, 256), (None, 256), (None, 256)] | output: (None, 256)
dropout_243: Dropout | input: (None, 256) | output: (None, 256)
batch_normalization_250: BatchNormalization | input: (None, 256) | output: (None, 256)
activation_10: Activation | input: (None, 256) | output: (None, 256)
dense_1137: Dense | input: (None, 256) | output: (None, 128)
dense_1134: Dense | input: (None, 256) | output: (None, 128)
batch_normalization_251: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_11: Activation | input: (None, 128) | output: (None, 128)
dense_1135: Dense | input: (None, 128) | output: (None, 128)
batch_normalization_252: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_12: Activation | input: (None, 128) | output: (None, 128)
dense_1136: Dense | input: (None, 128) | output: (None, 128)
add_244: Add | input: [(None, 128), (None, 128), (None, 128)] | output: (None, 128)

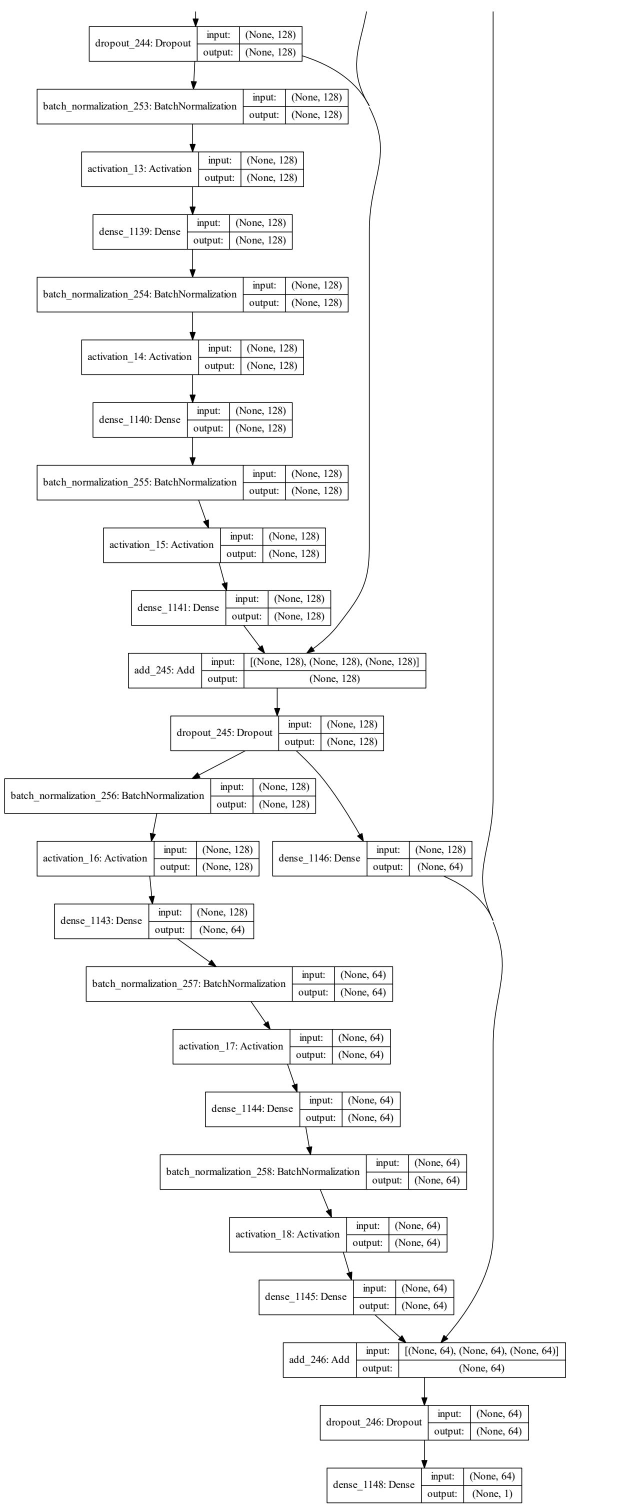

dropout_244: Dropout | input: (None, 128) | output: (None, 128)
batch_normalization_253: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_13: Activation | input: (None, 128) | output: (None, 128)
dense_1139: Dense | input: (None, 128) | output: (None, 128)
batch_normalization_254: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_14: Activation | input: (None, 128) | output: (None, 128)
dense_1140: Dense | input: (None, 128) | output: (None, 128)
batch_normalization_255: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_15: Activation | input: (None, 128) | output: (None, 128)
dense_1141: Dense | input: (None, 128) | output: (None, 128)
add_245: Add | input: [(None, 128), (None, 128), (None, 128)] | output: (None, 128)
dropout_245: Dropout | input: (None, 128) | output: (None, 128)
batch_normalization_256: BatchNormalization | input: (None, 128) | output: (None, 128)
activation_16: Activation | input: (None, 128) | output: (None, 128)
dense_1146: Dense | input: (None, 128) | output: (None, 64)
dense_1143: Dense | input: (None, 128) | output: (None, 64)
batch_normalization_257: BatchNormalization | input: (None, 64) | output: (None, 64)
activation_17: Activation | input: (None, 64) | output: (None, 64)
dense_1144: Dense | input: (None, 64) | output: (None, 64)
batch_normalization_258: BatchNormalization | input: (None, 64) | output: (None, 64)
activation_18: Activation | input: (None, 64) | output: (None, 64)
dense_1145: Dense | input: (None, 64) | output: (None, 64)
add_246: Add | input: [(None, 64), (None, 64), (None, 64)] | output: (None, 64)
dropout_246: Dropout | input: (None, 64) | output: (None, 64)
dense_1148: Dense | input: (None, 64) | output: (None, 1)

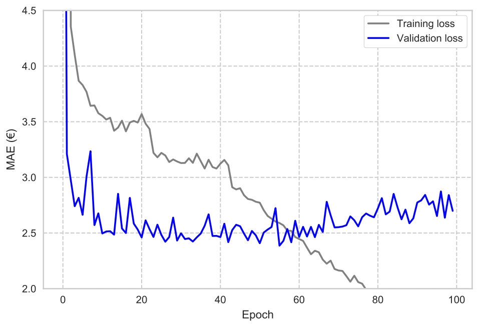
Figure 29: Model topology *ResNet 2*.

Results per run

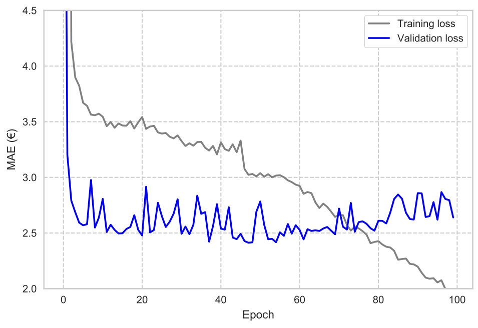
Run	Validation				Test			
	MVL		CE		MVL		CE	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	2.3863	4.3225	2.3476	4.3791	2.7721	4.6800	2.6972	**4.7074**
2	2.4126	**4.4930**	2.3699	4.4675	**2.7029**	**4.7459**	**2.6776**	4.7783
3	2.4244	4.5616	2.3440	4.4171	2.7446	4.8454	2.6721	4.7382
4	2.3864	4.4851	2.3606	**4.4284**	2.6857	4.7842	2.7052	4.6529
5	**2.4089**	4.5622	**2.3549**	4.4701	2.6902	4.7088	2.6647	4.6453
Mean	2.4037	4.4849	2.3554	4.4324	2.7191	4.7529	2.6834	4.7044
Std.	0.0169	0.0978	0.0103	0.0379	0.0377	0.0649	0.0172	0.0565

Table 9: MAE and associated RMSE for five runs of *ResNet 2*. The median performer is highlighted. All values are in EUR/MWh.

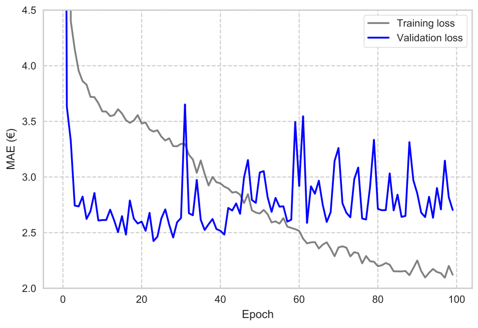
Training plots



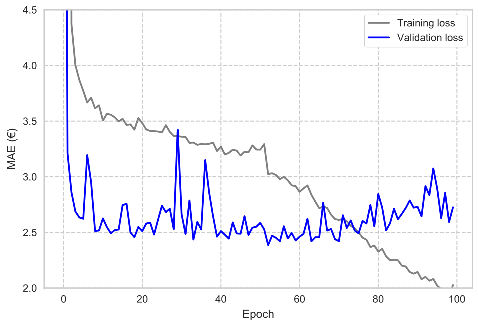
(a) Run 1



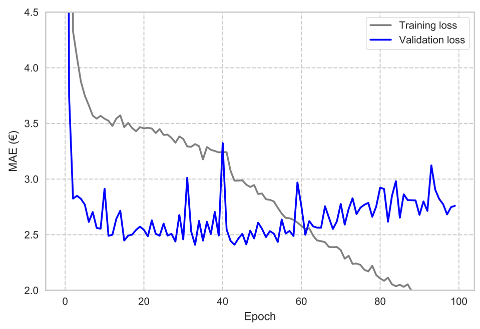
(b) Run 2



(c) Run 3



(d) Run 4



(e) Run 5

Figure 30: Training plots *ResNet 2*.

D.3 LSTM

Model topology

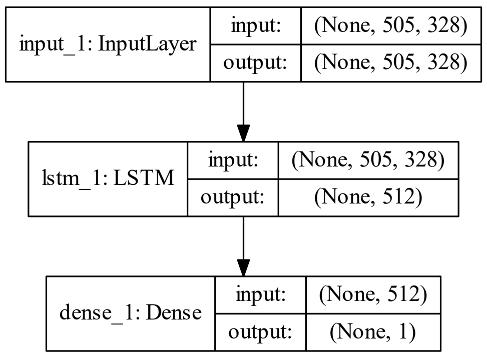


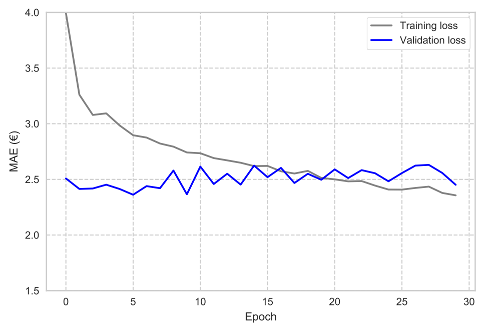
Figure 31: Model topology *LSTM*.

Results per run

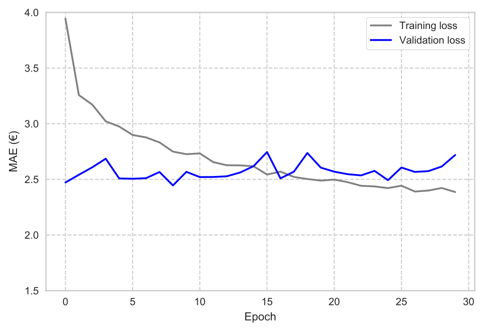
Run	Validation				Test			
	MVL		CE		MVL		CE	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	2.3620	3.4672	2.2765	**3.3728**	2.8084	4.3270	**2.7414**	4.2589
2	2.4457	3.5324	2.3361	3.4574	**2.8278**	**4.3563**	2.7421	**4.2803**
3	**2.3553**	**3.4757**	2.2829	3.3613	2.7772	4.2948	2.7217	4.2369
4	2.3545	3.4842	2.2416	3.3471	2.8324	4.4001	2.7370	4.2815
5	2.3249	3.4516	**2.2475**	3.3809	2.8521	4.3718	2.7731	4.3210
Mean	2.3685	3.4822	2.2769	3.3839	2.8196	4.3500	2.7431	4.2757
Std.	0.0455	0.0306	0.0376	0.0430	0.0283	0.0406	0.0187	0.0312

Table 10: MAE and associated RMSE for five runs of *LSTM*. The median performer is highlighted. All values are in EUR/MWh.

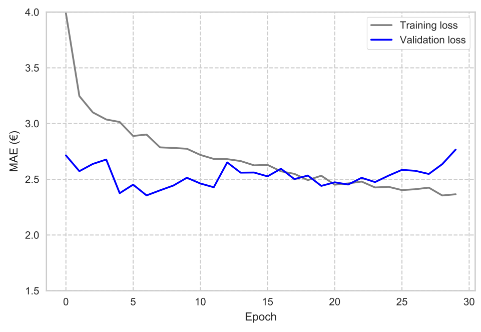
Training plots



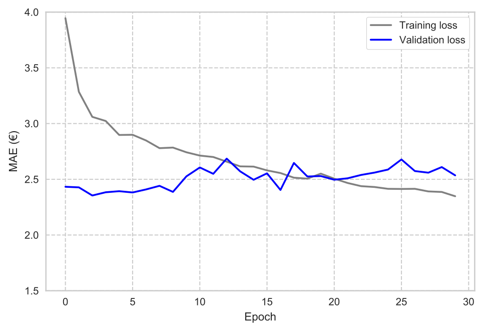
(a) Run 1



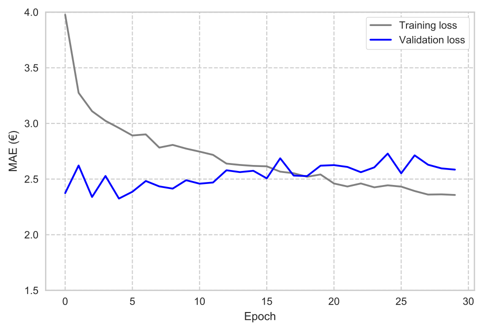
(b) Run 2



(c) Run 3



(d) Run 4



(e) Run 5

Figure 32: Training plots *LSTM*.

D.4 BD-LSTM

Model topology

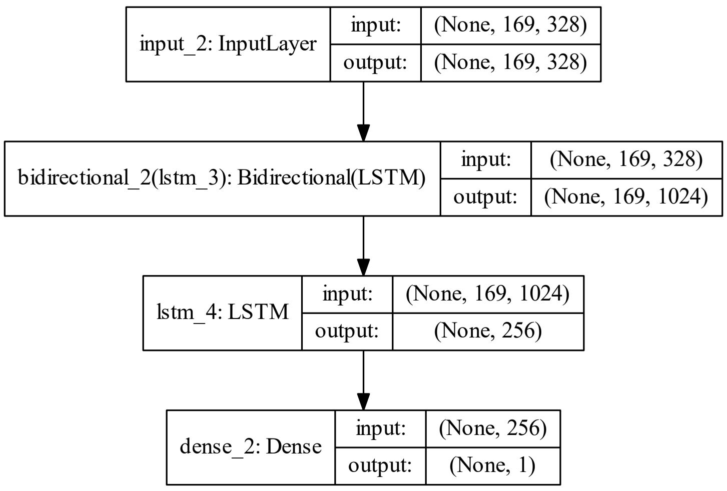


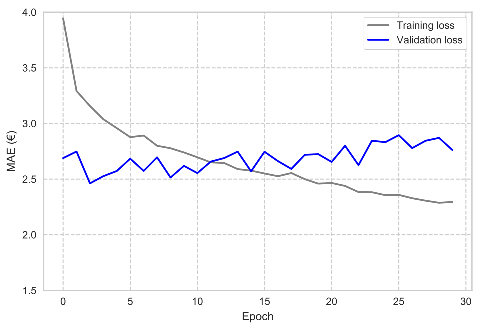
Figure 33: Model topology *BD-LSTM*.

Results per run

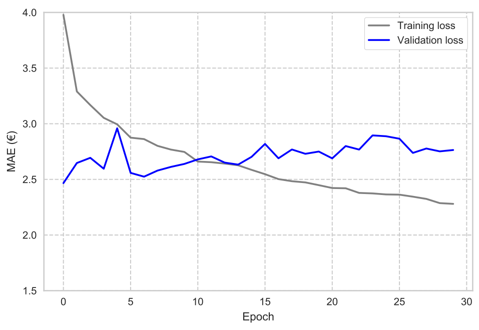
Run	Validation				Test			
	MVL		CE		MVL		CE	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	2.4622	**3.9569**	2.3826	3.7868	2.7471	4.3706	2.7223	**4.3147**
2	2.4663	4.0374	2.4263	3.8505	3.0154	4.6683	2.7663	4.3642
3	2.5453	3.9105	2.3819	3.7579	**2.8510**	4.3758	2.7189	4.3082
4	2.5448	3.9375	2.4589	**3.8373**	2.8325	**4.3781**	**2.7439**	4.2886
5	**2.5423**	4.0348	**2.4168**	3.8624	2.8548	4.4886	2.7500	4.3573
Mean	2.5122	3.9754	2.4133	3.8190	2.8602	4.4563	2.7403	4.3266
Std.	0.0438	0.0578	0.0324	0.0447	0.0972	0.1284	0.0198	0.0327

Table 11: MAE and associated RMSE for five runs of *BD-LSTM*. The median performer is highlighted. All values are in EUR/MWh.

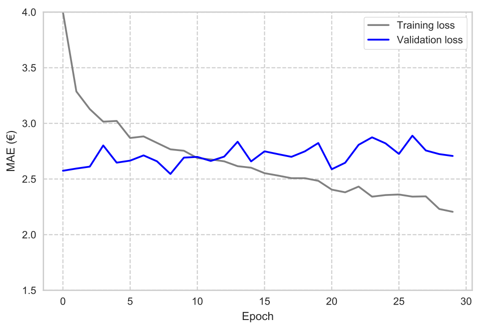
Training plots



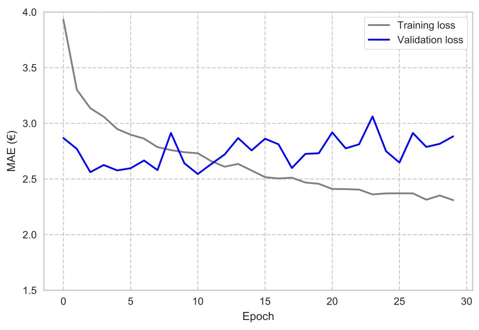
(a) Run 1



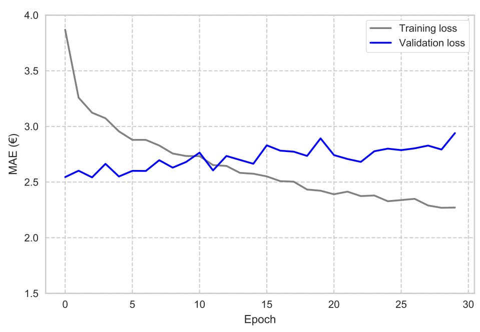
(b) Run 2



(c) Run 3



(d) Run 4

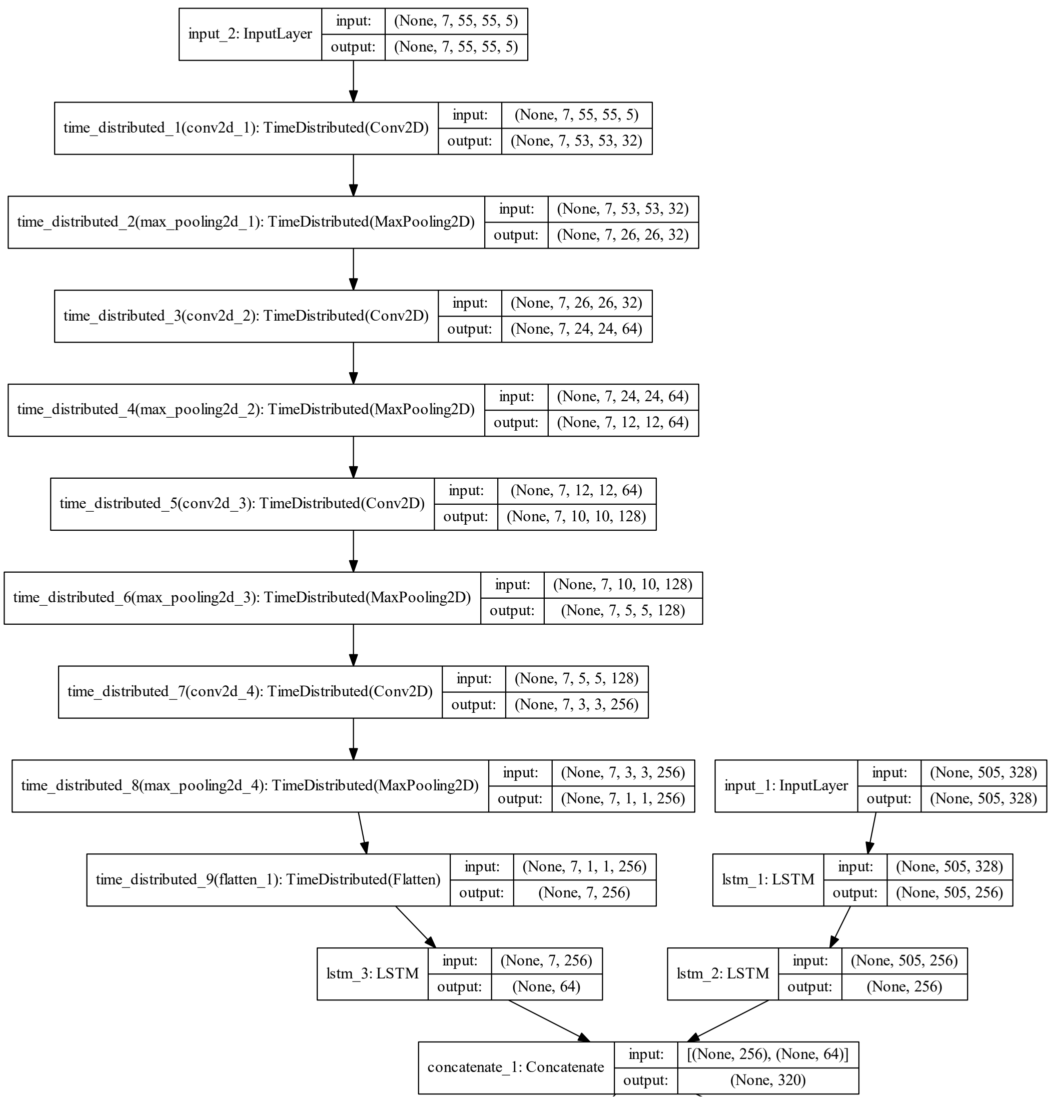


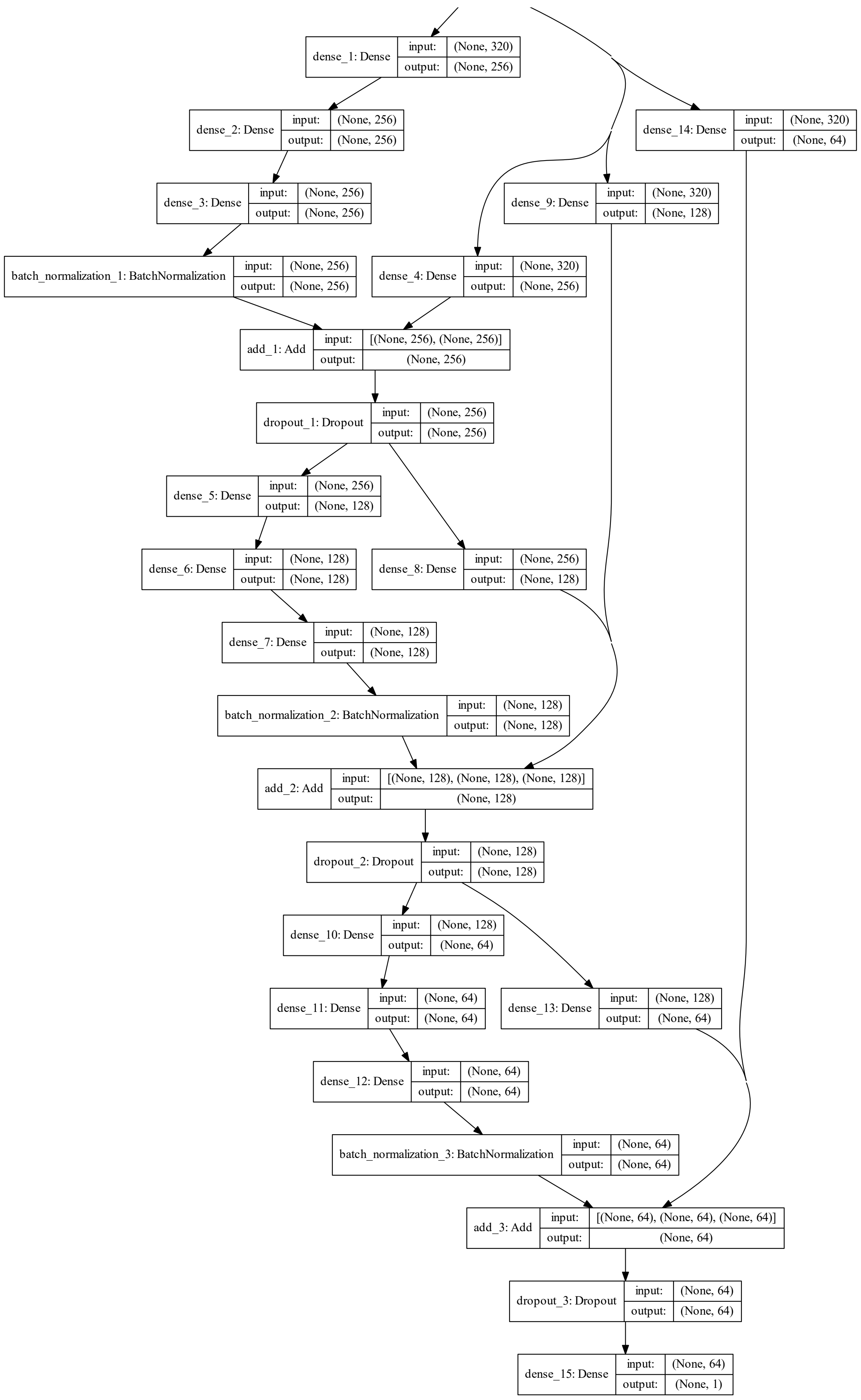
(e) Run 5

Figure 34: Training plots *BD-LSTM*.

D.5 Multi LSTM ResNet

Model topology


input_2: InputLayer | input: (None, 7, 55, 55, 5) | output: (None, 7, 55, 55, 5)
time_distributed_1(conv2d_1): TimeDistributed(Conv2D) | input: (None, 7, 55, 55, 5) | output: (None, 7, 53, 53, 32)
time_distributed_2(max_pooling2d_1): TimeDistributed(MaxPooling2D) | input: (None, 7, 53, 53, 32) | output: (None, 7, 26, 26, 32)
time_distributed_3(conv2d_2): TimeDistributed(Conv2D) | input: (None, 7, 26, 26, 32) | output: (None, 7, 24, 24, 64)
time_distributed_4(max_pooling2d_2): TimeDistributed(MaxPooling2D) | input: (None, 7, 24, 24, 64) | output: (None, 7, 12, 12, 64)
time_distributed_5(conv2d_3): TimeDistributed(Conv2D) | input: (None, 7, 12, 12, 64) | output: (None, 7, 10, 10, 128)
time_distributed_6(max_pooling2d_3): TimeDistributed(MaxPooling2D) | input: (None, 7, 10, 10, 128) | output: (None, 7, 5, 5, 128)
time_distributed_7(conv2d_4): TimeDistributed(Conv2D) | input: (None, 7, 5, 5, 128) | output: (None, 7, 3, 3, 256)
time_distributed_8(max_pooling2d_4): TimeDistributed(MaxPooling2D) | input: (None, 7, 3, 3, 256) | output: (None, 7, 1, 1, 256)
input_1: InputLayer | input: (None, 505, 328) | output: (None, 505, 328)
time_distributed_9(flatten_1): TimeDistributed(Flatten) | input: (None, 7, 1, 1, 256) | output: (None, 7, 256)
lstm_1: LSTM | input: (None, 505, 328) | output: (None, 505, 256)
lstm_3: LSTM | input: (None, 7, 256) | output: (None, 64)
lstm_2: LSTM | input: (None, 505, 256) | output: (None, 256)
concatenate_1: Concatenate | input: [(None, 256), (None, 64)] | output: (None, 320)



dense_1: Dense input: (None, 320) output: (None, 256)
dense_2: Dense input: (None, 256) output: (None, 256)
dense_14: Dense input: (None, 320) output: (None, 64)
dense_3: Dense input: (None, 256) output: (None, 256)
dense_9: Dense input: (None, 320) output: (None, 128)
batch_normalization_1: BatchNormalization input: (None, 256) output: (None, 256)
dense_4: Dense input: (None, 320) output: (None, 256)
add_1: Add input: [(None, 256), (None, 256)] output: (None, 256)
dropout_1: Dropout input: (None, 256) output: (None, 256)
dense_5: Dense input: (None, 256) output: (None, 128)
dense_6: Dense input: (None, 128) output: (None, 128)
dense_8: Dense input: (None, 256) output: (None, 128)
dense_7: Dense input: (None, 128) output: (None, 128)
batch_normalization_2: BatchNormalization input: (None, 128) output: (None, 128)
add_2: Add input: [(None, 128), (None, 128), (None, 128)] output: (None, 128)
dropout_2: Dropout input: (None, 128) output: (None, 128)
dense_10: Dense input: (None, 128) output: (None, 64)
dense_11: Dense input: (None, 64) output: (None, 64)
dense_13: Dense input: (None, 128) output: (None, 64)
dense_12: Dense input: (None, 64) output: (None, 64)
batch_normalization_3: BatchNormalization input: (None, 64) output: (None, 64)
add_3: Add input: [(None, 64), (None, 64), (None, 64)] output: (None, 64)
dropout_3: Dropout input: (None, 64) output: (None, 64)
dense_15: Dense input: (None, 64) output: (None, 1)

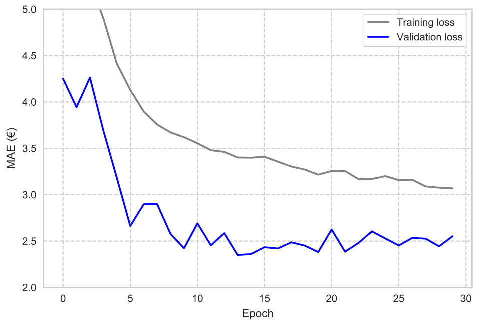
Figure 35: Model topology *Multi LSTM ResNet*.

Results per run

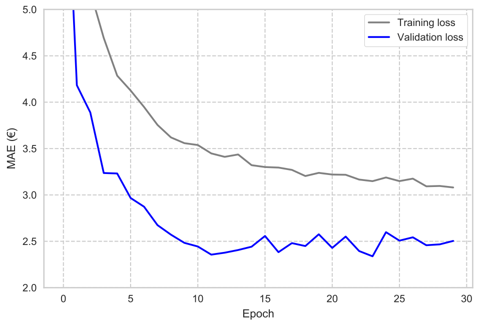
Run	Validation				Test			
	MVL		CE		MVL		CE	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
1	2.3498	3.5415	2.3031	**3.4525**	2.7740	4.3489	**2.7359**	4.3236
2	**2.3383**	3.4533	**2.2854**	3.4003	2.8253	**4.3390**	2.7241	4.2574
3	2.3984	3.5398	2.3277	3.5034	2.8569	4.3914	2.7589	4.3072
4	2.3246	3.4578	2.2688	3.3844	2.7746	4.2580	2.7360	4.2315
5	2.2898	**3.5277**	2.2842	3.4897	**2.7824**	4.3288	2.7269	**4.2672**
Mean	2.3402	3.5040	2.2938	3.4461	2.8026	4.3332	2.7364	4.2774
Std.	0.0396	0.0446	0.0225	0.0527	0.0370	0.0483	0.0137	0.0375

Table 12: MAE and associated RMSE for five runs of *Multi LSTM ResNet*. The median performer is highlighted. All values are in EUR/MWh.

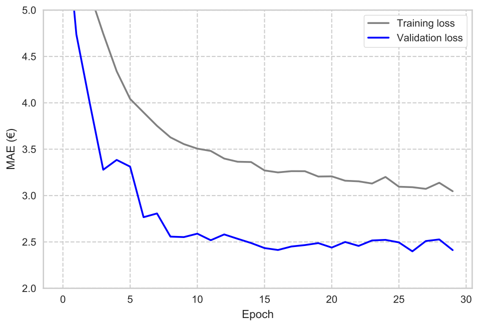
Training plots



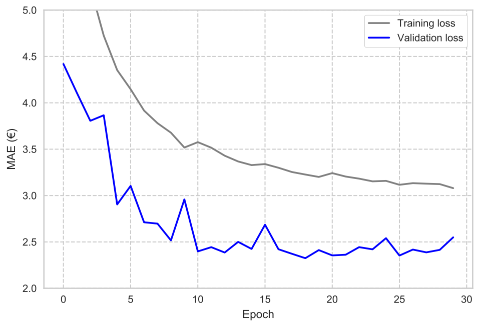
(a) Run 1



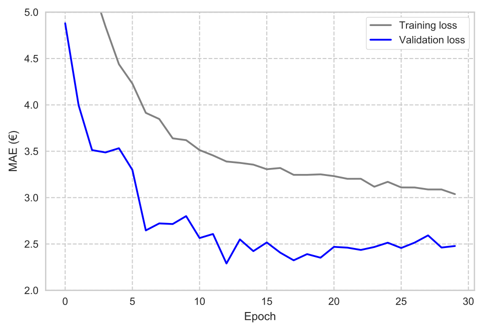
(b) Run 2



(c) Run 3



(d) Run 4



(e) Run 5

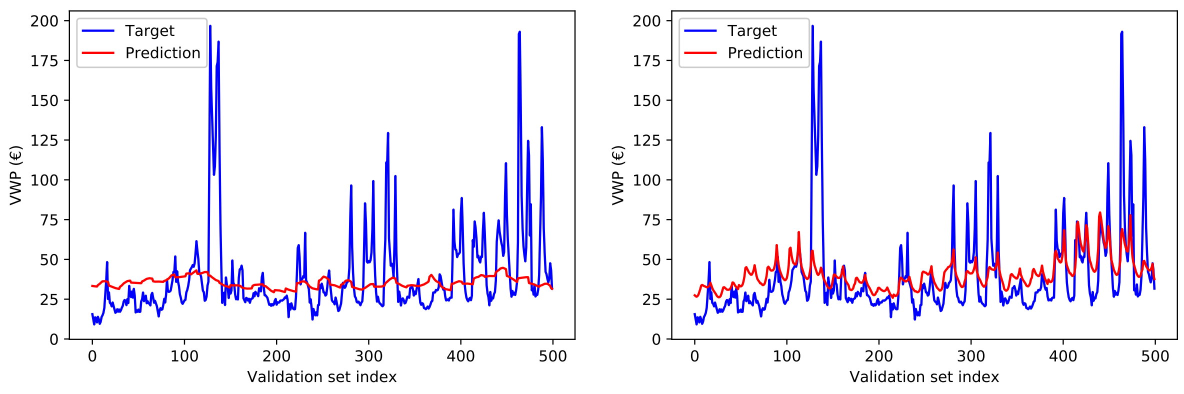
Figure 36: Training plots *Multi LSTM ResNet*.

D.6 Weather Models

To assess whether weather data can at all be used to predict intraday electricity prices, we perform some experiments with models using only the weather data. Naturally, these neural networks face a challenging task, as this is essentially akin to placing a camera in front of a weather report and asking it to give us a price in Euros. The only market data available to these models is the output itself, which can really only be captured at an aggregate level, either explicitly by bias terms or implicitly in nodes' weights. We might therefore expect that networks with access only to weather forecasts will tend to very nearly or exactly predict the mean or median price from the training set in most cases, perhaps with some adjustments to match cyclical patterns for instance in time of day or the season of the year. Since the only available market data in this setting is the output price, the only comparable simple heuristic baseline is the mean volume weighted price from the training set, with corresponding MAEs of 9.00 EUR/MWh and 6.68 EUR/MWh respectively on the validation and test sets.

The convolutiopnal networks we experiment with fit well to the training set, but the performance on the validation set is rather volatile, and it begins overfitting almost immediately. Efforts to mitigate this using additional dropout, reducing capacity, or introducing weight shrinkage did not help noticeably. Similarly, the simpler multi-layer perceptrons successfully fits on the training data, but its validation set loss is also unstable. The best validation set MAEs achieved by these simple CNNs and MLPs are EUR 7.65 and EUR 7.68, respectively. Though very similar, the models' validation set loss are so erratic that it is hard to conclude anything definitive with regards to their ability to generalise to unseen data. Nevertheless, their best MAEs are, somewhat surprisingly, a fair amount better than the EUR 9.00 of always predicting the average price from the training set (Table 3, Section 6.1.1). That said, these networks might have to spend most of their representational ability on deciphering the delivery hour's temporal position in some cyclical pattern, rather than any deviations or patterns in the data compared to such norms. Presumably, the networks might primarily be picking up on seasonal (in terms of winter and summer) and time-of-day patterns using solar radiation and temperature.

In this regard, fully-connected networks can easily be given some additional context with which to map representations of the weather forecast data that better capture deviations from cyclical patterns. Since such networks flatten the input data and treat each combination of time step and input variable as a separate feature, we can simply append to the end of that tensor a set of indicator variables for the corresponding hour of the day, week day, and month. We would expect a network that is given such context to be able to more accurately predict prices at any given delivery hour — even if it only has access to weather data, in addition to these indicators — as it will not have to waste as much representational capacity on inferring cyclical patterns.



(a) Without seasonal indicators.

(b) Giving the same network explicit seasonal indicator variables inreases accuracy by a healthy margin.

Figure 37: Price predictions by a simple MLP on a subset of the validation set, using only weather forecast images and, in one case, seasonal indicators.

Figure 37 (left) shows the individual predictions of a very simple MLP in a subset of the validation set time frame. The exact same architecture is then trained again on an extension of the weather forecast data with added indicator variables, and the corresponding plot is shown in Figure 37 (right). Compared to the MAE of EUR 7.65 achieved without any explicit context, including these indicator variables reduce this to minimum validation loss to 7.20. This latter network can more easily use temporal context to detect patterns in weather properties relative to the delivery hour's temporal position. For instance, it might be more confident that a forecasted temperature is abnormally low if it already "knows" that the corresponding hour is in the middle of the day during summer — otherwise it could simply be night-time or the middle of the winter.

Somewhat surprisingly, networks without the added explicit context provided by indicator variables do not simply resort to predicting some constant or near-constant price, but in fact seem to vary their predictions based on recognised weather patterns. In Figure 37a, we see that the MLP is able to not only discern some cyclical intraday pattern, but also has some longer-term fluctuations. That being said, the network seems to revert to some global average, as it evidently lacks the confidence to make large steps outside this average when predicting prices based solely on weather data. The inclusion of explicit indicator variables in Figure 37b clearly increases the network's willingness to take such risks[129], as the magnitude of the cyclical fluctuations is greatly increased, as well as the deviations from the loosely discernible global tendency. Comparing Figure 37a and Figure 37b, it is impressive the extent to which the network is in fact able to predict the near Elbas VWP based solely on forecasted weather *at* the corresponding delivery hour — not even the weather preceding it, or the difference between predicted and actual weather — along with some very simple contextual temporal information.

Consequently, convolutional neural networks perform better than we might expect considering their translation invariance, and are not necessarily beat by simpler fully-connected neural networks using the same data. However, it is trivial to give the latter additional temporal context using indicator variables, which improves their performance considerably. As such, it seems that neural networks given only a single forecast of five weather bands are able to discern some pattern that can help it predict volume-weighted prices better than simply guessing some global measure — especially if also given some simple temporal context. Nevertheless, it is not clear that including this weather data will improve the performance of networks that already have access to market data, as any information from weather forecasts might already be captured by that data. That being said, the fully-connected networks that combine weather forecasts and temporal context evidently do more than simply predict using only the latter, which would suggest that the weather data includes some information with predictive value beyond simply conveying seasonal or cyclical effects.

[129]Predicting a price that is very different from some global average is a risk in that being more wrong is penalised more by the loss function, hence it needs sufficient confidence in having recognised some pattern in the data to justify taking such steps outside the norm.

D.7 Results LSTM vs. GRU

Model specifications				LSTM			GRU		
Model	**Look-back (days)**	**Hidden layer 1**	**Hidden layer 2**	**MAE**	**RMSE**	**s/ epoch**	**MAE**	**RMSE**	**s/ epoch**
1	7	48		3.0492	5.0532	70	2.9851	4.9471	55
2	14	48		2.9826	4.7248	135	2.9644	4.8112	107
3	7	72	72	2.646	4.4994	128	2.8146	5.0104	100
4	14	72 (0.2, 0.3)	72 (0.2, 0.3)	2.5771	4.1661	259	2.6931	4.9741	200
5	7	256 (0.2, 0.3)		2.5872	4.2477	83	2.7102	4.8597	65
6	14	256 (0.2, 0.3)		2.5990	4.1915	158	2.7249	4.7481	128
7	14	256 (0.2, 0.3)	256 (0.2, 0.3)	2.6063	4.1749	304	2.7265	4.9212	240
8	21	256 (0.2, 0.3)		2.5925	3.9643	238	2.5823	4.0816	193
9	14	512 (0.2, 0.3)		2.6200	4.2375	199	2.6831	4.6738	156
10	7	256 (0.2, 0.3)	256 (0.2, 0.3)	2.6199	4.2241	161	2.8084	5.1020	121
11	14	512 (0.2, 0.3)	512 (0.2, 0.3)	2.6827	4.2760	504	2.8127	4.7704	331
12	14	512 (0.2, 0.3)	512 (0.2, 0.3)	2.7574	4.3825	513	3.0478	5.3720	337
13	21	512 (0.2, 0.3)		2.4665	3.8010	307	2.5911	4.1012	232
14	21	512 (0.2, 0.3)	512 (0.2, 0.3)	2.4595	3.8225	942	2.5471	4.0585	655
15	30	512 (0.2, 0.3)		2.5397	3.8538	545	2.6018	4.0580	371

Table 13: Performance of models with respectively LSTM and GRU layers. Model 12 is exactly the same as model 11, except it is stateful. All other models are stateless. MAE and RMSE are in EUR/MWh.

E Theory on Benchmark Models

E.1 Shrinkage Methods

Shrinkage methods add a regularisation term, $\Omega(\beta)$, to the loss function, $C(\hat{y}, y)$ so that the regularised loss function, $\tilde{C}(\hat{y}, y)$, is:

$$\tilde{C}(\hat{y}, y) = C(\hat{y}, y) + \lambda\Omega(\beta) \tag{4}$$

where $\hat{y}$ is a linear model $f(\mathbf{x}, \beta) = \beta_0 + \sum_{j=1}^{p} \beta_j \mathbf{x}_j$ with $j = 1, ..., p$ inputs in $\mathbf{x}$ and β the estimated coefficients. The values of β are chosen so that the expression in Equation 4 is minimised. The tuning parameter, λ, controls the extent of regularisation imposed on the loss function, and it is usually selected through cross-validation.

Ridge regression

In ridge regression, the regularisation term is the sum of the squared coefficients, β, that is, $\Omega(\beta) = \sum_{j=1}^{p} \beta_j^2$. Ridge regression can also be expressed as an optimisation problem where the loss function, $C(\hat{y}, y)$, is minimised subject to the constraint $\sum_{j=1}^{p} \beta_j^2 \leq s$. When s approaches infinity, λ approaches zero. At zero, the restriction is no longer binding and no shrinkage is imposed. Alternatively, the closer s gets to zero, the stronger the shrinkage imposed on the linear model.

LASSO

In the LASSO, or the *Least Absolute Shrinkage and Selection Operator*, method, the regularisation term is set to equal the sum of the absolute values of the coefficients, β, that is $\Omega(\beta) = \sum_{j=1}^{p} |\beta_j|$. Expressed as an optimisation problem, LASSO minimises the loss function, $C(\hat{y}, y)$, subject to the constraint $\sum_{j=1}^{p} |\beta_j| \leq s$. When the tuning parameter, λ, is sufficiently large, LASSO sets some coefficients to zero. Thus, LASSO performs feature selection, in contrast to ridge that may shrink coefficients considerably but never sets them exactly equal to zero. Hence, in a sparse dataset, where not all variables are relevant, LASSO would perform best, while in a dense dataset, where all variables have relevance, ridge would be preferred.

Elastic net

Elastic nets are hybrids between ridge regression and LASSO, where the regularisation term is expressed as the weighted sum of the ridge and LASSO terms: $\Omega(\beta) = \sum_{j=1}^{p}(\alpha|\beta_j| + (1-\alpha)\beta_j^2)$.

The value of α is used to indicate the weight between ridge regression ($\alpha = 0$) and LASSO ($\alpha = 1$) — hence, for elastic nets that are a hybrid between ridge and LASSO, $\alpha = (0, 1)$.

E.2 Feature Selection

Random Forest

Random forest is a bootstrap aggregation, or *bagging*, technique that combines the results from multiple trees grown from random data samples. Hastie et al. (2009) describe how the random forest algorithm is implemented to make predictions in regression problems. To grow each random forest tree, T_b, where $b = 1, ..., B$, the algorithm draws a bootstrap sample, $Z*$, with N observations from the training data. The tree, T_b, is then grown by randomly selecting m of p variables, identifying the best split point among the m variables, and then split the node into two sub-nodes at that point. For each terminal node of the tree, this procedure is repeated until reaching the minimum node size, $nmin$. The ensemble of trees over $b = 1, ..., B$ is $\{T_b\}_1^B$. In regression, the algorithm uses this ensemble, $\{T_b\}_1^B$, to make a prediction $\hat{y}$ at point x by taking the average of the ensemble:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \tag{5}$$

The variables that have the highest importance in the random forest model can then be extracted and used to fit a linear model.

Multivariate Adaptive Regression Splines (MARS)

Another approach for feature selection is to extract the most important features from a multivariate adaptive regression splines (MARS) model. MARS splits a function, at the value $x = t$, into two piecewise linear basis functions, called a *reflected pair*:

$$(x-t)_+ = \begin{cases} x - t & \text{if } x > t \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad (t-x)_+ = \begin{cases} t - x & \text{if } x < t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where + means the positive part (Hastie et al., 2009). Then, the MARS model takes a form that resembles a linear model as shown in Equation 7 below. The exception is that instead of using the variables, x, directly as inputs, the MARS model uses $h_m(x)$, where each $h_m(x)$ is a function in the collection of basis functions or the product of two or more of the functions in (Hastie et al., 2009). The variables that have the highest importance when fitting the MARS model in Equation 7 can then be extracted and used to fit a linear model.

$$f(x) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(x) \tag{7}$$

E.3 Feature Extraction

Principal Component Analysis

Principal component analysis (PCA) performs an orthogonal linear transformation of a set of variables, X. Hence, PCA is not a feature selection method, as the components are derived from combinations of all the original variables (James, Witten, Hastie, & Tibshirani, 2017). The first principal component, z_1, has the direction v_1 resulting in the transformation $z_1 = Xv_1$ having the largest variance amongst all linear combinations of the set of variables, X. Then, each subsequent principal component, $z_i = Xv_i,$ for $i = 1, ..., M$, is derived so that the variance is maximised subject to being orthogonal to the earlier components (Hastie et al., 2009).

The n first principal components of a set of variables, X, can be extracted and used to fit a model, instead of using the original variables (James et al., 2017). The components are uncorrelated, hence, PCA removes potential weakness of one or more of the original variables, X, being strongly correlated.

E.4 Gradient Boosting

Boosting methods combine many weak learners[130] so as to achieve a higher predictive accuracy (Hastie et al., 2009). To minimise the loss, new learners are added one at a time. In the *gradient boosting* algorithm, decision trees are used as the weak learners. Decision trees are found to have the potential of outperforming classical approaches when the relationship between input variables and the dependent variable is a highly non-linear and complex (James et al., 2017). Hastie et al. (2009) describes each step in the gradient boosting algorithm. First, the algorithm

[130]A weak learner is one that performs only slightly better than random guessing.

is initialised with a single terminal node tree outputting the prediction $f_0(x)$ as:

$$f_0(x) = \text{arg min}_\gamma \sum_{i=1}^{N} C(y_i, \gamma) \tag{8}$$

where $i = 1, 2, ..., N$ are the training observations, y_i their corresponding true targets, C a loss function selected for the regression problem, and γ a constant assigned to the single terminal node of the initial tree. Furthermore, γ is assigned so that the sum of the loss for all observations i is minimised. To improve predictive accuracy, new trees are added using a gradient descent procedure. For each iteration $m = 1, 2, ..., M$ and for each of the training observations i, the negative gradient of the loss function, r_{im}, is computed as:

$$r_{im} = -[\frac{\partial C(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}} \tag{9}$$

where $f(x_i)$ is the prediction from iteration $m - 1$. The negative gradient gives the direction of the steepest descent, and adding a new tree in this direction will reduce the loss. This is done by fitting a tree with the objective to predict the negative gradient r_{im}, as r_{im} reflects the prediction error of the the previous trees in the sequence. The terminal nodes of the new tree are $j = 1, 2, ..., J_m$, and in iteration m they represent the regions r_{jm}. For each $j = 1, 2, ..., J_m$, γ_{jm} — the constant assigned to node j in iteration m — is computed as:

$$\gamma_{jm} = \text{arg min}_\gamma \sum_{x_i \in R_{jm}} C(y_i, f_{m-1}(x_i) + \gamma) \tag{10}$$

Hence, γ_{jm} is assigned so that the the loss when predicting the true value y_i from $f_{m-1}(x_i) + \gamma$ is minimised. Finally, the update from iteration m is:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{j_m} \gamma_{jm} I(x \in R_{jm}) \tag{11}$$

where $f_{m-1}(x)$ is the prediction from iteration $m - 1$ and $\sum_{j=1}^{j_m} \gamma_{jm} I(x \in R_{jm})$ the new tree. Finally, after the last iteration M, the resulting prediction $\hat{y}$ is:

$$\hat{y} = f_M(x) \tag{12}$$

F Market Data Variables

F.1 Extracted Variables

Dataset	**Variable**	**Unit**	**Level of aggregation**	**Lagged**
Elbas ticker	Near Elbas VWP (output)	EUR/MWh	Nordic	
	Lagged near Elbas VWP	EUR/MWh	Nordic	$h-6$
	Lagged near Elbas volume	MWh	Nordic	$h-6$
	Far Elbas VWP	EUR/MWh	Nordic	
	Far Elbas volume	MWh	Nordic	
Elspot file	Elspot prices	EUR/MWh	Per area	
	Elspot volumes	MWh	Per area	
Regulating prices	Up-regulation price	EUR/MWh	Per area	$h-8$
	Down-regulation price	EUR/MWh	Per area	$h-8$
	Dominating direction	Up = 1; No = 0; Down = -1	Per area	$h-8$
	Consumption imbalance price	EUR/MWh	Per area	$h-8$
	Production imbalance prices (sales/purchases)	EUR/MWh	Per area	$h-8$
Elspot capacity	Elspot transmission capacities	MW	Per area	
Elbas capacity	Elbas initial transmission capacities	MW	Per area	
Operating data	Consumption prognosis	MW	Per area	
	Production prognosis	MW	Per area	
UMM data	Total number of UMMs	Count	Nordic	
	Number of production UMMs	Count	Nordic	
	Volume reported in production UMMs	MWh	Nordic	
	Number of consumption UMMs	Count	Nordic	
	Volume reported in consumption UMMs	MWh	Nordic	
Seasonal dummies	Hour-of-day	$1-24$		
	Day-of-week	$1-7$		
	Month	$1-12$		

Table 14: Extracted variables per market dataset. All (except day-of-week and month dummies) are observed on an hourly frequency.

F.2 Summary Statistics

Summary statistics are calculated on the complete dataset.

Elbas ticker variables

Variable	Minimum	1st Qu.	Median	Mean	3rd Qu.	Maximum	Stdev
N of trades	1.00	13.00	20.00	23.02	29.00	986.00	16.48
Price per trade	-150.00	25.60	31.20	32.88	38.00	555.00	13.58
Volume per trade	0.00	5.00	10.00	16.82	20.00	890.00	24.64

Table 15: Summary statistics for the Elbas ticker data.

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Win_VWP_Near	-75.68	25.00	30.90	31.93	37.47	259.55	12.41	0
Win_VWP_Far	-54.16	25.00	30.74	31.54	36.88	253.90	12.02	7313
Win_Volume_Far	0.10	33.00	80.00	130.03	170.00	3355.60	155.59	7313
Win_VWP_Near_Lag	-75.68	25.00	30.91	31.93	37.47	259.55	12.42	5
Win_Volume_Near_Lag	0.10	120.40	216.10	276.44	363.70	3180.50	228.48	5

Table 16: Summary statistics for the *near* and *far* Elbas VWP and volume variables.

Elspot price variables

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Spot_PR_DK1	-200.00	24.09	30.63	31.16	37.11	2000.00	22.60	0
Spot_PR_DK2	-200.00	25.36	31.57	32.77	38.85	253.92	13.93	0
Spot_PR_FI	0.32	27.52	33.53	34.97	40.57	300.01	13.64	0
Spot_PR_NO1	0.59	23.30	28.38	28.56	33.39	234.38	10.19	0
Spot_PR_NO2	0.59	23.26	28.28	28.23	33.06	210.00	9.23	0
Spot_PR_NO3	1.14	25.12	30.13	30.48	35.21	253.92	10.27	0
Spot_PR_NO4	1.46	23.86	28.27	28.98	33.89	253.92	10.03	0
Spot_PR_NO5	0.59	22.88	28.30	28.17	33.08	210.00	9.59	0
Spot_PR_SE1	0.32	25.04	30.45	30.79	35.86	253.92	10.93	0
Spot_PR_SE2	0.32	25.04	30.47	30.80	35.87	253.92	10.93	0
Spot_PR_SE3	0.32	25.10	30.62	31.23	36.22	253.92	11.67	0
Spot_PR_SE4	0.32	25.42	31.01	32.11	37.41	253.92	12.44	0
Spot_PR_SP1	1.14	24.54	29.34	29.64	34.50	224.97	9.90	0

Table 17: Summary statistics for the Elspot price variables.

Elspot volume variables

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Spot_OM_Buy_DK1	515.70	1432.50	1911.10	1963.18	2403.60	4499.90	660.84	0
Spot_OM_Buy_DK2	673.90	1337.00	1646.90	1690.61	2048.10	3034.50	468.33	0
Spot_OM_Buy_FI	3171.70	5411.60	6189.20	6270.52	7032.82	11163.40	1182.38	0
Spot_OM_Buy_NO1	1529.70	2812.00	3769.30	3978.39	5039.95	8299.00	1400.01	0
Spot_OM_Buy_NO2	1529.60	3731.10	4303.10	4259.90	4811.10	6839.60	826.40	0
Spot_OM_Buy_NO3	856.10	2150.20	2490.00	2445.31	2796.90	3860.30	525.46	0
Spot_OM_Buy_NO4	905.70	1478.30	1738.80	1746.04	2008.80	2662.90	332.44	0
Spot_OM_Buy_NO5	534.50	1525.20	1829.75	1835.07	2199.62	3361.40	525.11	0
Spot_OM_Buy_SE1	754.20	1054.40	1175.00	1195.03	1322.00	1946.20	184.40	0
Spot_OM_Buy_SE2	955.40	1419.10	1631.15	1654.81	1866.80	2845.50	309.32	0
Spot_OM_Buy_SE3	4658.20	7706.30	9092.25	9246.99	10621.60	16353.20	2012.12	0
Spot_OM_Buy_SE4	1172.10	2504.80	3075.90	3136.54	3741.10	5632.90	866.45	0
Spot_OM_Buy_SP1	22634.00	34284.65	39086.10	39980.50	45187.55	65311.10	7850.75	0
Spot_OM_Sell_DK1	142.70	1372.20	1942.90	1989.91	2566.72	4916.00	827.95	0
Spot_OM_Sell_DK2	68.50	677.00	1015.70	1020.01	1323.70	2799.40	463.99	0
Spot_OM_Sell_FI	1946.90	3836.10	4620.45	4664.38	5496.50	8799.70	1114.26	0
Spot_OM_Sell_NO1	902.80	1831.30	2211.70	2434.57	2784.90	5932.30	873.75	0
Spot_OM_Sell_NO2	771.60	4271.98	5308.15	5547.16	6833.75	9975.50	1782.97	0
Spot_OM_Sell_NO3	243.60	1284.58	1747.65	1799.47	2236.10	3542.20	631.26	0
Spot_OM_Sell_NO4	626.30	1950.40	2436.55	2450.19	2888.50	4120.10	677.66	0
Spot_OM_Sell_NO5	406.10	1877.40	3236.75	3197.29	4226.12	7397.80	1481.71	0
Spot_OM_Sell_SE1	433.90	1677.28	2586.55	2555.66	3375.10	4884.60	1006.38	0
Spot_OM_Sell_SE2	656.40	3461.40	4681.05	4470.35	5593.95	7734.20	1413.37	0
Spot_OM_Sell_SE3	4462.10	7466.68	8939.55	8886.13	10329.90	12353.40	1668.99	0
Spot_OM_Sell_SE4	67.60	491.70	734.50	761.16	990.12	2265.50	337.36	0

Table 18: Summary statistics for the Elspot volume variables.

Elbas capacity variables

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Cap_DK1_DK1A	0.00	1161.00	2141.00	2053.36	2989.00	4484.50	1152.71	456
Cap_DK1_DK2	0.00	0.00	65.00	217.77	459.23	1190.00	260.85	336
Cap_DK1A_DK1	0.00	483.00	1203.00	1373.43	2120.00	4578.00	1093.38	456
Cap_DK1A_NO2	0.00	717.00	1550.00	1462.48	1954.62	3064.00	954.04	216
Cap_DK1A_SE3	0.00	0.00	391.05	490.31	740.00	1420.00	486.19	288
Cap_DK2_DK1	0.00	600.00	895.55	836.50	1190.00	1200.00	370.08	336
Cap_DK2_SE4	0.00	1029.75	1860.45	1729.72	2511.00	3000.00	935.94	744
Cap_FI_SE1	0.00	1813.08	2491.00	2162.82	2600.00	2600.00	579.90	360
Cap_FI_SE3	0.00	1351.92	2025.00	1827.37	2400.00	2700.00	647.62	456
Cap_NO1_NO2	0.00	1733.90	2669.00	2610.67	3554.80	5450.00	1192.89	600
Cap_NO1_NO3	0.00	0.00	0.00	0.14	0.00	200.00	4.00	720
Cap_NO1_NO5	0.00	944.00	1527.00	1696.57	2393.45	4400.00	970.55	768
Cap_NO1_SE3	0.00	0.00	387.00	907.63	1708.28	4240.00	1082.27	504
Cap_NO2_DK1A	0.00	0.00	212.25	673.16	1222.03	3264.00	844.41	216
Cap_NO2_NO1	0.00	634.48	1498.65	1531.20	2271.00	5300.00	1078.12	600
Cap_NO2_NO5	0.00	100.00	250.00	289.56	400.00	900.00	239.73	792
Cap_NO3_NO1	0.00	0.00	0.00	0.17	0.00	250.00	5.23	744
Cap_NO3_NO4	0.00	291.70	500.55	495.46	729.00	1100.00	284.13	672
Cap_NO3_SE2	0.00	195.00	600.00	655.01	1096.00	1600.00	500.35	504
Cap_NO4_NO3	0.00	0.00	77.00	239.53	436.00	1050.00	297.81	672
Cap_NO4_SE1	0.00	0.00	235.00	329.13	649.00	1200.00	343.93	480
Cap_NO4_SE2	0.00	0.00	0.00	80.58	150.00	1000.00	117.68	504
Cap_NO5_NO1	0.00	214.00	700.00	1042.84	1682.10	4200.00	1008.05	768
Cap_NO5_NO2	0.00	115.00	300.00	273.64	400.00	800.00	187.40	816
Cap_SE1_FI	0.00	0.00	0.00	306.25	552.75	1998.00	463.40	360
Cap_SE1_NO4	0.00	364.68	548.00	536.52	750.00	1150.00	311.52	480
Cap_SE1_SE2	0.00	2074.00	2799.00	2664.01	3300.00	5474.00	881.01	360
Cap_SE2_NO3	0.00	217.00	762.00	723.88	1131.00	1600.00	523.30	504
Cap_SE2_NO4	0.00	100.00	250.00	223.09	400.00	1000.00	146.65	504
Cap_SE2_SE1	0.00	3300.00	3555.00	3673.91	4295.23	6600.00	891.57	360
Cap_SE2_SE3	0.00	1524.78	3086.50	3311.02	4988.55	9748.00	2169.81	432
Cap_SE3_DK1A	0.00	157.85	680.00	639.70	1028.65	1420.00	474.12	288
Cap_SE3_FI	0.00	0.00	39.00	365.66	689.83	2700.00	499.34	456
Cap_SE3_NO1	0.00	1287.62	2245.00	2259.82	3339.93	4240.00	1260.76	504
Cap_SE3_SE2	0.00	8788.73	10643.30	10366.12	12260.00	14600.00	2427.29	432
Cap_SE3_SE4	0.00	593.15	1575.40	1848.30	2918.10	6582.00	1471.61	576
Cap_SE4_DK2	0.00	149.00	707.10	842.70	1331.00	3000.00	735.76	744
Cap_SE4_SE3	0.00	3650.45	4970.85	4755.16	5964.00	7400.00	1524.23	576

Table 19: Summary statistics for the Elbas capacity variables.

Elspot capacity variables

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Spot_UE_DK1_DK1A	246.00	1690.00	1740.00	1786.73	2272.00	2372.00	424.52	0
Spot_UE_DK1_DK2	0.00	590.00	590.00	558.54	590.00	600.00	121.39	0
Spot_UE_DK1A_DK1	250.00	1610.00	1630.00	1705.97	2212.00	2312.00	413.50	0
Spot_UE_DK1A_NO2	0.00	950.00	1000.00	1115.27	1532.00	1632.00	369.19	0
Spot_UE_DK1A_SE3	0.00	355.00	740.00	560.39	740.00	740.00	218.25	0
Spot_UE_DK2_DK1	0.00	600.00	600.00	566.61	600.00	600.00	126.20	0
Spot_UE_DK2_SE4	27.00	1050.00	1700.00	1396.81	1700.00	1700.00	416.09	0
Spot_UE_FI_SE1	0.00	1065.00	1110.00	1076.16	1150.00	1230.00	159.58	0
Spot_UE_FI_SE3	0.00	1200.00	1200.00	1092.79	1200.00	1350.00	254.40	0
Spot_UE_NO1_NO2	400.00	1700.00	1700.00	1672.36	1900.00	2900.00	294.70	0
Spot_UE_NO1_NO3	-350.00	-50.00	50.00	78.61	200.00	550.00	162.54	0
Spot_UE_NO1_NO5	0.00	300.00	300.00	419.25	650.00	650.00	161.12	0
Spot_UE_NO1_SE3	0.00	1050.00	1747.50	1565.89	2105.00	2145.00	568.97	0
Spot_UE_NO2_DK1A	0.00	950.00	950.00	1073.36	1432.00	1632.00	353.65	0
Spot_UE_NO2_NO1	700.00	2100.00	2600.00	2528.94	3100.00	3500.00	657.94	0
Spot_UE_NO2_NO5	0.00	100.00	250.00	217.09	300.00	400.00	100.67	0
Spot_UE_NO3_NO1	-550.00	-200.00	-50.00	-78.61	50.00	350.00	162.54	0
Spot_UE_NO3_NO4	0.00	0.00	0.00	7.78	0.00	200.00	28.14	0
Spot_UE_NO3_SE2	0.00	600.00	600.00	578.70	600.00	600.00	89.87	0
Spot_UE_NO4_NO3	0.00	600.00	900.00	737.64	900.00	1000.00	217.46	0
Spot_UE_NO4_SE1	0.00	350.00	650.00	503.22	650.00	700.00	188.19	0
Spot_UE_NO4_SE2	0.00	100.00	150.00	121.76	150.00	250.00	50.71	0
Spot_UE_NO5_NO1	0.00	700.00	2600.00	2343.82	3600.00	3900.00	1312.67	0
Spot_UE_NO5_NO2	0.00	300.00	300.00	352.16	500.00	600.00	118.70	0
Spot_UE_SE1_FI	50.00	1440.00	1470.00	1419.60	1510.00	1580.00	208.42	0
Spot_UE_SE1_NO4	0.00	400.00	450.00	375.53	450.00	550.00	118.96	0
Spot_UE_SE1_SE2	1100.00	3000.00	3300.00	3115.35	3300.00	3300.00	294.23	0
Spot_UE_SE2_NO3	0.00	600.00	900.00	813.27	1000.00	1000.00	218.91	0
Spot_UE_SE2_NO4	0.00	100.00	250.00	185.41	250.00	250.00	82.86	0
Spot_UE_SE2_SE1	2900.00	3300.00	3300.00	3299.82	3300.00	3300.00	8.48	0
Spot_UE_SE2_SE3	3100.00	6200.00	6600.00	6551.92	7000.00	7300.00	561.49	0
Spot_UE_SE3_DK1A	0.00	680.00	680.00	580.92	680.00	680.00	187.14	0
Spot_UE_SE3_FI	0.00	1200.00	1200.00	1115.76	1200.00	1350.00	225.73	0
Spot_UE_SE3_NO1	0.00	1158.00	1850.00	1613.54	2095.00	2095.00	546.29	0
Spot_UE_SE3_SE2	800.00	7300.00	7300.00	7296.79	7300.00	7300.00	142.67	0
Spot_UE_SE3_SE4	2600.00	4400.00	4900.00	4688.48	5100.00	5600.00	595.41	0
Spot_UE_SE4_DK2	0.00	1300.00	1300.00	1195.42	1300.00	1300.00	264.32	0
Spot_UE_SE4_SE3	0.00	2000.00	2000.00	1996.42	2000.00	2000.00	81.74	0

Table 20: Summary statistics for the Elspot capacity variables.

Regulating price variables (1): Down- and up-regulating prices

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Reg_RN_DK1	-217.02	21.00	27.81	27.53	34.03	183.40	13.32	0
Reg_RN_DK2	-217.02	22.00	28.69	29.06	35.22	253.92	13.53	0
Reg_RN_FI	-1000.00	23.68	29.85	30.62	36.82	300.01	16.16	0
Reg_RN_NO1	-25.55	20.70	26.00	26.00	31.42	234.38	9.84	0
Reg_RN_NO2	-25.55	20.66	25.96	25.74	31.21	149.10	9.10	0
Reg_RN_NO3	-25.55	22.17	27.84	27.58	33.00	253.92	10.14	0
Reg_RN_NO4	-25.55	21.00	26.17	26.34	31.70	253.92	9.86	0
Reg_RN_NO5	-25.55	20.41	25.96	25.73	31.29	203.08	9.41	0
Reg_RN_SE1	-66.89	21.88	27.88	27.62	33.28	253.92	10.60	0
Reg_RN_SE2	-66.89	21.95	27.90	27.66	33.30	253.92	10.59	0
Reg_RN_SE3	-66.89	22.00	28.00	28.03	33.59	253.92	11.20	0
Reg_RN_SE4	-66.89	22.13	28.25	28.62	34.19	253.92	11.86	0
Reg_RO_DK1	-199.99	25.80	31.99	34.45	39.00	2000.00	26.42	0
Reg_RO_DK2	-199.99	26.97	32.90	36.88	40.86	1999.00	25.47	0
Reg_RO_FI	0.32	28.85	35.00	40.22	43.39	3000.00	41.41	0
Reg_RO_NO1	0.59	24.12	29.58	30.37	34.75	1999.00	17.12	0
Reg_RO_NO2	0.59	24.07	29.48	29.88	34.36	500.00	12.42	0
Reg_RO_NO3	1.14	26.14	31.41	32.79	37.04	500.00	15.38	0
Reg_RO_NO4	1.46	24.50	29.57	31.02	35.47	500.00	14.70	0
Reg_RO_NO5	0.59	23.87	29.42	29.77	34.40	500.00	12.72	0
Reg_RO_SE1	0.32	26.09	31.50	32.99	37.22	1999.00	18.66	0
Reg_RO_SE2	0.32	26.09	31.54	33.03	37.25	1999.00	18.78	0
Reg_RO_SE3	0.32	26.21	31.73	33.86	37.82	1999.00	20.77	0
Reg_RO_SE4	0.32	26.54	32.15	35.38	39.06	1999.00	23.54	0

Table 21: Summary statistics for the down- and up-regulating price variables. All variables have 53,400 observations.

Regulating price variables (2): Imbalance prices

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Reg_RC_DK1	-217.02	22.32	29.17	30.85	36.10	500.00	19.73	0
Reg_RC_DK2	-217.02	23.26	30.00	33.23	37.44	1999.00	25.88	0
Reg_RC_FI	-1000.00	24.65	31.14	35.93	39.58	3000.00	42.82	0
Reg_RC_NO1	-25.55	21.43	27.04	27.82	32.99	1999.00	17.19	0
Reg_RC_NO2	-25.55	21.39	27.00	27.39	32.73	500.00	12.65	0
Reg_RC_NO3	-25.55	23.00	28.93	29.90	35.00	500.00	15.71	0
Reg_RC_NO4	-25.55	21.71	27.29	28.36	33.46	500.00	14.92	0
Reg_RC_NO5	-25.55	21.10	27.00	27.34	32.79	500.00	12.89	0
Reg_RC_SE1	-66.89	22.53	28.87	29.84	35.00	1999.00	18.83	0
Reg_RC_SE2	-66.89	22.63	28.91	29.92	35.00	1999.00	18.95	0
Reg_RC_SE3	-66.89	22.88	29.03	30.69	35.37	1999.00	20.93	0
Reg_RC_SE4	-66.89	23.00	29.40	31.95	36.00	1999.00	23.72	0
Reg_RP_DK1	-200.00	25.73	31.97	34.42	39.00	2000.00	26.43	0
Reg_RP_DK2	-200.00	26.94	32.88	36.85	40.83	1999.00	25.49	0
Reg_RP_FI	0.32	28.82	35.00	40.16	43.31	3000.00	41.39	0
Reg_RP_NO1	0.59	24.09	29.55	30.35	34.73	1999.00	17.12	0
Reg_RP_NO2	0.59	24.04	29.46	29.85	34.35	500.00	12.42	0
Reg_RP_NO3	1.14	26.10	31.38	32.76	37.01	500.00	15.36	0
Reg_RP_NO4	1.46	24.46	29.52	30.97	35.43	500.00	14.68	0
Reg_RP_NO5	0.59	23.85	29.40	29.75	34.38	500.00	12.72	0
Reg_RP_SE1	0.32	26.07	31.50	32.96	37.20	1999.00	18.65	0
Reg_RP_SE2	0.32	26.07	31.52	33.01	37.24	1999.00	18.77	0
Reg_RP_SE3	0.32	26.19	31.70	33.83	37.79	1999.00	20.77	0
Reg_RP_SE4	0.32	26.51	32.13	35.35	39.04	1999.00	23.52	0
Reg_RS_DK1	-217.02	21.00	27.89	27.59	34.11	183.40	13.35	0
Reg_RS_DK2	-217.02	22.03	28.77	29.15	35.36	253.92	13.59	0
Reg_RS_FI	-1000.00	23.77	29.94	30.74	36.94	300.01	16.22	0
Reg_RS_NO1	-25.55	20.74	26.02	26.03	31.45	234.38	9.85	0
Reg_RS_NO2	-25.55	20.70	25.99	25.77	31.25	149.10	9.12	0
Reg_RS_NO3	-25.55	22.23	27.89	27.62	33.04	253.92	10.16	0
Reg_RS_NO4	-25.55	21.00	26.20	26.37	31.74	253.92	9.87	0
Reg_RS_NO5	-25.55	20.45	25.99	25.76	31.31	203.08	9.42	0
Reg_RS_SE1	-66.89	21.92	27.90	27.67	33.32	253.92	10.62	0
Reg_RS_SE2	-66.89	21.99	27.96	27.71	33.33	253.92	10.61	0
Reg_RS_SE3	-66.89	22.02	28.02	28.09	33.65	253.92	11.25	0
Reg_RS_SE4	-66.89	22.19	28.32	28.70	34.30	253.92	11.93	0

Table 22: Summary statistics for the imbalance price variables.

Consumption and production prognosis variables

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
Op_CE_DK1	1184.00	1853.00	2205.00	2255.64	2649.00	3687.00	481.52	0
Op_CE_DK2	725.00	1261.00	1519.00	1519.50	1749.00	2545.00	325.81	0
Op_CE_FI	0.00	8328.00	9265.00	9404.42	10468.00	15280.00	1483.53	0
Op_CE_SE1	548.00	964.00	1120.00	1146.77	1290.00	2145.00	232.34	48
Op_CE_SE2	878.00	1628.00	1873.00	1911.16	2170.00	3613.00	377.61	48
Op_CE_SE3	5039.00	8275.00	9786.00	9967.78	11494.00	18215.00	2196.13	24
Op_CE_SE4	1345.00	2283.00	2748.00	2800.99	3260.00	5165.00	672.92	24
Op_PE_DK1	71.00	1051.00	1518.00	1607.10	2081.00	5014.00	774.15	0
Op_PE_DK2	60.00	555.00	846.00	866.95	1160.00	2437.00	392.14	0
Op_PE_FI	667.00	6398.00	7416.00	7451.49	8469.00	12328.00	1440.25	0
Op_PE_NO1	988.00	1937.00	2331.00	2614.67	3002.00	6309.00	1000.70	0
Op_PE_NO2	875.00	4438.00	5640.00	5830.58	7278.00	10016.00	1867.47	0
Op_PE_NO3	358.00	1481.00	1982.00	1974.16	2441.25	3788.00	633.73	0
Op_PE_NO4	1000.00	2285.00	2769.00	2798.18	3278.00	4601.00	717.20	0
Op_PE_NO5	686.00	2312.00	3102.00	3288.93	4062.00	7256.00	1245.16	0
Op_PE_SE1	351.00	1610.00	2514.00	2481.45	3295.00	4839.00	1001.60	0
Op_PE_SE2	873.00	3703.00	4991.00	4791.51	5984.00	8265.00	1499.19	0
Op_PE_SE3	4644.00	7932.00	9741.50	9566.08	11176.00	13570.00	1868.54	0
Op_PE_SE4	192.00	531.00	796.00	862.35	1130.00	2342.00	406.32	0

Table 23: Summary statistics for the consumption and production prognosis variables.

Urgent market messages (UMM)

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max	Stdev	NAs
UMM_Tot_Cnt	0.00	0.00	1.00	1.85	3.00	15.00	2.16	0
UMM_PE_Vol	0.00	0.00	0.00	74.51	0.00	8835.00	278.50	0
UMM_PE_Cnt	0.00	0.00	0.00	0.30	0.00	14.00	0.82	0
UMM_CE_Vol	0.00	0.00	0.00	4.86	0.00	3000.00	50.50	0
UMM_CE_Cnt	0.00	0.00	0.00	0.03	0.00	6.00	0.22	0

Table 24: Summary statistics for the urgent market message (UMM) variables.