Chair of Energy Network Technology

Master's Thesis

Time-Series Forecasting of an Electric Steel Mill's Power Demand - A Neural Network Approach

Sebastian Halbwirth, BSc

May 2022

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 08.05.2022

S Halbwirth

Unterschrift Verfasser/in
Sebastian Halbwirth

ABSTRACT

When wanting to lower the atmospheric carbon dioxide concentration, it is of interest to look at the iron and steel industry, as especially the blast furnace process with its large coal consumption is a significant emittent. Electric steel mills, on the other hand, use electricity and natural gas to melt scrap metal into steel. Demand side management can thereby help to integrate renewable energy sources and increase energy efficiency. In this context, load forecasting is an important tool to know the future energy demand of such an industrial process.

The purpose of this work is to develop a machine learning model that can predict the power demand of an electric steel mill's primary aggregates as accurately as possible. Due to the high power demand of the electric arc furnace, a focus is laid on this aggregate.

In the course of this thesis, literature research was conducted about machine learning algorithms, their advantages and disadvantages, and their use cases. In particular, machine learning methods used in energy system modelling were researched. A suitable method was then chosen, and based on this method, multiple models were created to forecast the aggregates' power demand in a time-resolved manner.

The method chosen for predicting the power demand was a neural network. Two types of neural networks were compared: long short-term memory networks and standard feedforward networks. Altogether six models were created, of which five are based on long short-term memory networks.

The results show that long short-term memory networks can be used to predict the power demand of an electric arc furnace. By stochastically generating input parameters and realigning the predicted and actual batches of the electric arc furnace process, the model can be implemented in and used for demand side management applications.

KURZFASSUNG

Wenn die Kohlendioxidkonzentration in der Atmosphäre gesenkt werden soll, muss die Eisen- und Stahlindustrie berücksichtigt werden, da diesbezüglich insbesondere der Hochofenprozess mit seinem hohen Kohleverbrauch ein bedeutender Emittent ist. Elektrolichtbogenöfen hingegen verwenden Strom und Erdgas um Metallschrott zu Stahl zu schmelzen. Dabei kann Demand Side Management helfen, erneuerbare Energiequellen zu integrieren und die Energieeffizienz zu erhöhen. In diesem Zusammenhang ist die Lastprognose ein wichtiges Instrument zur Ermittlung des künftigen Energiebedarfs.

Ziel dieser Arbeit ist es, ein Machine-Learning-Modell zu entwickeln, das den Leistungsbedarf der Primäraggregate eines Elektrostahlwerks so genau wie möglich prognostizieren kann. Aufgrund des hohen Leistungsbedarfs des Elektrolichtbogenofens wird ein Schwerpunkt auf dieses Aggregat gelegt.

Im Rahmen dieser Arbeit wurde eine Literaturrecherche über Algorithmen des maschinellen Lernens, deren Vor- und Nachteile und deren Einsatz durchgeführt. Insbesondere wurden Machine-Learning Verfahren recherchiert, die in der Energiesystemmodellierung eingesetzt werden. Ein geeignetes Verfahren wurde ausgewählt, mit dem mehrere Modelle zur zeitaufgelösten Vorhersage des Leistungsbedarfs der Aggregate erstellt wurden.

Als Prognosemethode wurde ein neuronales Netz ausgewählt. Es wurden dabei zwei Arten von neuronalen Netzen verglichen: Long Short-Term Memory Netze und Feedforward-Netze.

Die Ergebnisse zeigen, dass Long Short-Term Memory Netze zur Vorhersage des Leistungsbedarfs eines Lichtbogenofens verwendet werden können. Durch die stochastische Generierung von Inputparametern und die zeitliche Synchronisation der vorhergesagten und tatsächlichen Chargen des Lichtbogenofens kann das Modell als Basis für eine Optimierung in Demand Side Management Anwendungen eingesetzt werden.

CONTENTS

NOMENCLATURE

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ASHA	Asynchronous Successive Halving
CNN	Convolutional Neural Network
CO_2	Carbon Dioxide
D	Dedusting
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
dCor	Distance Correlation
DNN	Deep Neural Network
DSM	Demand Side Management
EAF	Electric Arc Furnace
FI	Feature Importance
GB	Gigabyte
GJ	Gigajoule
KS	Kolmogorov-Smirnov
LF	Ladle Furnace
LH	Ladle Heater
LSTM	Long Short-Term Memory
LTLF	Long-Term Load Forecasting
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
min	Minute
MLP	Multilayer Perceptron
MSE	Mean Squared Error

MTLF	Medium-Term Load Forecasting
MW	Megawatt
MWh	Megawatt Hour
PI	Permutation Importance
PLS	Partial Least Squares
r	Correlation Coefficient
R^2	Coefficient of Determination
RAGS	Random Approximated Greedy Search
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SHAP	Shapley Additive Explanations
SSR	Regression Sum of Squares
SSTO	Total Sum of Squares
STLF	Short-Term Load Forecasting
SVM	Support Vector Machine
t	Ton

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Since the Industrial Revolution, temperatures have been rising and the scientific community agrees that human activities are primarily responsible for this trend. [1] In the past decade, decarbonisation has therefore been a focus of climate researchers with the goal of lowering the worldwide carbon dioxide (CO_2) emissions. The path towards decarbonisation can be achieved in different ways by, for example, using synthetic fuels or decarbonised electricity and increasing energy efficiency in industrial settings. [2]

According to the Austrian Federal Environment Agency [3], the production of iron and steel contributed to 12.9% of Austria's total greenhouse gas emissions in 2019, which makes this sector an essential lever for decreasing CO_2 emissions and reducing the industrial impact on climate. While the larger part of Austrian steel production is concentrated at two sites using blast furnaces, about 10% [3] of the total steel production is accounted for by electric arc furnaces (EAFs). The specific energy expenditure of EAFs (3.6 -5.9 GJ/t [4]) is lower than that of other processes, with the majority of it being used to melt steel scrap using electricity. Due to the limited use of coal, the CO_2 emissions of EAFs are also lower compared to those of blast furnaces. [3, 4] This presents EAFs as a crucial alternative to the blast furnace process with regard to decarbonising the metallurgy sector. One limitation of the EAF process is the high demand for electricity. [4]

Regarding this demand for electricity, load forecasts are an essential tool of industrial plants in the metallurgy sector to better manage their energy allocation, optimise their energy sources, and ensure their economic operation. A high-quality energy demand forecast can directly impact the economics and reliability of and for companies. Additionally, demand side management (DSM) can increase the operational flexibility of industrial production processes, reinforce the integration of renewable energies, and improve energy efficiency. [5]

Historically, conventional methods like statistical analysis and regression-based approaches have been used for forecasting purposes. Despite them still being relevant today, these models often cannot capture complex dependencies in non-linear data. In more recent times, Artificial Intelligence (AI) based methods have established themselves when dealing with complex forecasting problems [6] and have outperformed conventional methods [7]. AI has been utilised to forecast loads in the energy industry, including smart grids and buildings, next day load demands and loads in distributed systems. [8]

The work of this thesis was done as part of the project DSM_OPT, carried out by the Chair of Energy Technology at Montanuniversität Leoben. DSM_OPT focuses on the optimisation of industrial energy systems through demand side management. In this project, the aim is to develop a decision support system toolbox to increase energy efficiency, incorporate time-based tariffs, and integrate the fast-changing markets.

1.1 Objective

This Master’s thesis aimed to create machine learning models that are able to predict the power demand of the primary aggregates in the electric steel mill Marienhütte, located in Graz. The models created should be able to predict multiple time steps into the future. The time steps should thereby cover at least the length of one batch, of which the median time is 41 min, and ideally one week.

The research question was thus to find out what kind of model, more precisely method, could be used to accurately determine the future power usage of different steel mill aggregates. Using parameters measured at Marienhütte, the task was to create a machine learning model for the EAF that can predict the power usage as accurately as possible. The EAF was chosen due to it being the largest energy consumer of the steel mill; however, the chosen methods for the EAF were also applied to other aggregates including the ladle furnace, ladle heaters, and dedusting to see whether power forecasts using AI are possible.

1.2 Methodology

To answer the research question, it was essential to get an overview of the available methods of modelling data and forecasting energy demand. For this, literature research has been conducted, where different machine learning methods were described. Additionally, different use cases of energy system models in the iron and steel and energy industry were examined to get a more detailed overview of the industry’s state of the art.

Based on the findings of the literature research, adequate forecasting methods were adapted and used to predict the power demand of the primary aggregates. Previously measured data at Marienhütte was prepared, where, for example, missing values and outliers were detected and removed, and data was labelled. This preparation depended on the aggregate modelled and the forecasting method used.

The processed data was then used to forecast the power demand of the primary aggregates by creating different models in the programming language Python. The approach of preparing the data and creating these models was documented in this thesis.

1.3 Steel and Rolling Mill Marienhütte

Marienhütte is a company located in Graz and is the only producer of concrete steel both in the form of billets and wire rods in Austria. The plant consists of a steel and a rolling mill. [9]

The focus in this thesis lies on the steel mill and its main aggregates that are used during steel production: the electric arc furnace (EAF), ladle furnace (LF), ladle heaters (LHs), and dedusting (D). As the power demand of the continuous casting system (CC) was stable and relatively small, this aggregate was not modelled. Relevant data of these aggregates is consistently measured. For the purpose of this thesis, data has been extracted from past measurements of over 200 parameters between May and July 2021. These parameters can be either of continuous nature (e.g. melting temperature) or of logical nature (e.g. whether a valve is closed or not). Chapter 3 contains further information on these parameters.

2 Theoretical Background – Modelling Methods for Forecasting

Forecasting makes the fundamental assumption that the future shares a similar pattern or distribution to historical data. AI has been used for energy forecasting for several decades; however, due to its complexity and algorithm speed, it has seen a surge in recent years with advancements in computing power. Load forecasting is a vital subset of energy forecasting, and has been used for investigating the development of power demand for over a century (long-term load forecasts), or to decide when, where and how much electricity demand will grow (spatial load forecasting), or to pursue operational excellence (short-term load forecasting). [10]

The following sections describe different basic methods used in machine learning and statistical analysis to model future energy systems. Additionally, use cases in the iron and steel and energy industry of these and more advanced methods will be summarised. This literature research serves as a foundation for the subsequent practical part, in which models are created to forecast the power demand of the steel mill Marienhütte using previously collected and prepared data.

2.1 Machine Learning

With advances in computer technology, it is currently possible to store and process large amounts of data. For a computer to carry out any task requires a set of specific instructions, in other words, the implementation of an algorithm that defines rules which need to be followed. Machine learning, an umbrella term for these algorithms that help describe or make decisions or predictions about data, is a kind of AI. A prerequisite for intelligence is that a system can learn, especially in a changing environment. The advantage of this is that the system designer does not need to foresee the future and provide solutions for all possible situations because the system is able to learn independently. Nowadays, this trend is helping people worldwide in numerous ways, whether that is in vision, speech recognition, or robotics. [11]

2.1.1 Operating Principle

Machine learning is a field of expertise that has grown enormously in the past decade and will continue to do so in the foreseeable future. Therefore, it is crucial to get an overview of existing types of machine learning methods, different kinds of models, how to validate one's models and problems that arise when working with machine learning algorithms.

Machine learning methods can be organised based on the outcome of the algorithm. The three most well-known and acknowledged algorithm types are *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In supervised learning, the algorithm creates a function that maps inputs to desired outputs by looking at several input-output examples of the function. Algorithms that fall into this group are decision trees, regression analysis, support vector machines (SVMs), Naïve Bayes classifiers and neural networks. Unsupervised learning models a set of inputs by extracting structural information from the data samples and finding patterns; however, labelled examples are not given. It includes algorithms such as different kinds of clustering and dimensionality reduction. In reinforcement learning, the algorithm learns a policy of how to act, and its environment provides feedback. This machine learning method is expanded further in section 2.1.8. [11, 12]

To develop well-functioning machine learning models, proper evaluation is key. This evaluation generally occurs by splitting the available data into training, validation, and test sets. A common practice is using approximately 70% of the data points for training, 15% for validation, and 15% for testing. While the training set can then be used for model training, the validation set can be used for tuning the model's parameters, and the test set can be used for model evaluation. Because of this, it is crucial that the test set must not be exposed to the model before the final testing. [13]

Machine learning, in general, aims to correctly approximate a function that associates input elements with output elements; therefore, it is essential to consider the so-called *bias-variance trade-off*. A training set used in supervised learning methods represents the global distribution, but it is not exhaustive. Hence, it is vital to consider fitting the function to the data while still being flexible to unknown inputs. With this comes the danger of underfitting or overfitting the data. Underfitting means that the model cannot capture dynamics from the training set; overfitting means that the model is fit (almost) perfectly to the training data. However, in the latter case, with an unknown input, the prediction error can be high. Figure 2-1 shows a graphical comparison between the terms over- and underfitting, where the orange line represents the test or validation data, and the blue line represents the fitted model. [14] Overfitting relates to high variance in the model. To fight it, the variance has to be reduced by increasing regularisation, obtaining larger data sets, decreasing the number of features and more. On the other hand, underfitting relates to having a high bias, which can be reduced by decreasing regularisation and using more features. [15]

Fighting overfitting does generally not help in fighting underfitting. A balance must be reached; hence it is being called a trade-off. A generic rule of thumb is that a residual error, meaning a difference between the actual and predicted data points, is always necessary for avoiding overfitting, while a model that shows a validation accuracy of 99.99% is certainly overfitted. Because of these reasons, it is crucial to analyse the model's performance by looking at the training error and cross-validation error simultaneously. The training error signifies the amount of bias and is determined by whether the model fits the training data itself. Cross-validation error can signify high variance and is usually determined by modelling the validation dataset. [14, 15]

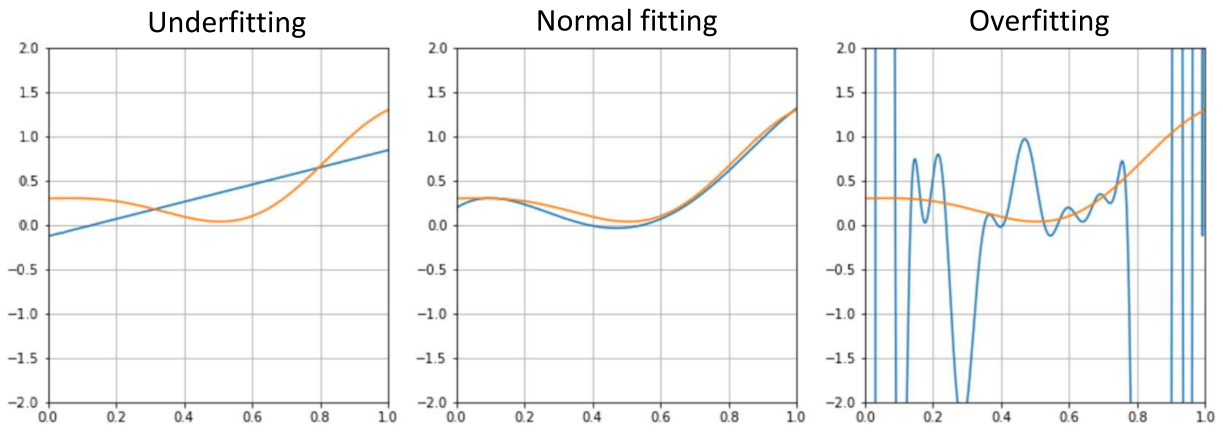

Figure 2-1: Comparison between underfitting and overfitting machine learning algorithms, with the orange line showing the actual data and the blue line showing the predictions [14]

Different methods are often combined or used in parallel to improve the accuracy of machine learning models. Two major approaches in this sense are hybrid and ensemble models. Hybrid models integrate machine learning methods with other machine learning or soft computing/optimisation methods, where the output of one model is essentially fed to the next. The models are dependent on each other. In a similar sense, ensemble models also combine weaker models through various grouping techniques such as bagging or boosting; however, these models work independently and in parallel of one another and generally make use of more than one machine learning algorithm. [16] The following sections outline different algorithms used in the space of machine learning.

2.1.2 Regression Analysis

Usually, when comparing two variables in a dataset, questions arise about the relationships between these variables. These questions can be answered using regression (analysing whether there is a relationship) and correlation (how strong the relationship is). [17]

2.1.2.1 Simple Linear Regression

Linear models are the simplest parametric methods, and even nonlinear problems can be solved using them, by recasting them through a process called linearisation. A parametric approach assumes that the sample obeys a given model, defined by a small number of parameters. [14, 18] The simplest linear model involves only one independent variable x (2-1). While the dependent variable y is the variable to be described by data, variables which are thought to predict, explain and provide information on this dependent variable are called independent. An example of a simple linear regression is:

$$\hat{y}(x) = \alpha_0 + \alpha \cdot x \tag{2-1}$$

where α_0 is the y -intercept and α is the coefficient for the regression line $\hat{y}(x)$. To fit the data to the model, the parameters are frequently found using the *least squares approach*, where the function L (2-2) is minimised:

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

with $e_i = (y_i - \hat{y}_i)$ being the observed residual for the i^{th} observation. The least squares estimation uses the criterion that the solution results in the smallest possible sum of squared deviations of the observed y_i from the estimates of their true means. [19]

An important metric used in regressions is the coefficient of determination, denoted as R^2. It is the proportion of the regression sum of squares (SSR) to the total sum of squares (SSTO):

$$R^2 = \frac{SSR}{SSTO} = 1 - \frac{SSE}{SSTO} = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \overline{y_i})^2} \tag{2-3}$$

SSE is the error sum of squares, which defines how much the data points y_i vary around the estimated regression line $\hat{y}_i$. $\overline{y_i}$ is the sample mean. [14, 19]

R^2 measures the variation on the prediction, which is explained by the dataset, and always ranges from 0 to 1. The higher the correlation of determination, the better; an R^2 of 1 would mean that the independent variables explain 100% of the dependent data. In the case of a simple linear regression, R^2 is also the square of the correlation coefficient r. [19]

2.1.2.2 Multiple Linear Regression

In a dataset of not just a single scalar value x but a vector with multiple parameters x_i, each vector is associated with a value y. The regression $\hat{y}$ then takes this form:

$$\hat{y} = \alpha_0 + \sum_{i=1}^{m} \alpha_i \cdot x_i \tag{2-4}$$

with α_i being the coefficients of the resulting hyperplane. The least squares estimation is also used in multiple linear regression, and while it is the most common loss function, there are also others like the Ridge, Lasso and ElasticNet function. [14, 19] The reason for not always using the least squares method is that it sometimes is not a perfectly robust (meaning effective) solution. A common problem is collinearity, which describes the phenomenon in which multiple independent variables are (almost) exactly correlated with each other. Collinearity is a problem as it inflates the variance of regression parameters and can potentially wrongly identify relevant predictors in a statistical model. [20]

2.1.2.3 Polynomial Regression

The least squares approach is not limited to linear models but can also be used on polynomials, as long as the coefficients enter the model linearly. If there is certain know-how of a domain, that a model is quadratic or periodic, this knowledge should be used. [21]

In a polynomial regression model with one variable (a quadratic model):

$$y = \alpha_0 + \alpha_1 \cdot x + \alpha_2 \cdot x^2 \tag{2-5}$$

the coefficients $\alpha_1$and α_2 are called linear effect parameter and quadratic effect parameter, respectively. Polynomial models explain more complex nonlinear relationships and are Taylor series expansions of unknown nonlinear functions. [22]

The regression techniques, linear and polynomial, that have been discussed so far, also go by the name of *curve fitting* in common literature. Curve fitting describes techniques to fit curves between discrete values to obtain (future) estimates. [23]

Polynomial models can be extended to the fitting of two or more variables. A second-order polynomial is, for example, more often used in practice:

$$y = \alpha_0 + \alpha_1 \cdot x_1 + \alpha_2 \cdot x_2 + \alpha_{11} \cdot {x_1}^2 + \alpha_{22} \cdot {x_2}^2 + \alpha_{12} \cdot x_1 x_2 \tag{2-6}$$

where $\alpha_1$and α_2 are called linear effect parameters, α_{11} and α_{22} are called quadratic effect parameters, and α_{12} is the interaction parameter. This function is also called the *response surface*. In practice, response surface methodology is used by researchers to design experiments in order to obtain an optimal response. [22, 24]

Significant problems of multivariate polynomial regressions are collinearity and the fact that higher degree terms in the equation do not contribute majorly to the regression equation. [24] One should always maintain a sense of parsimony, meaning to use the simplest possible model by, for example, transforming to keep the model first order. This will prevent issues of overfitting the model. [25]

When building polynomial models, one can use two procedures, which do not always lead to the same model: forward selection and backward elimination. In forward selection, models of increasing order are successively fitted until the t-test for the highest order term is not significant. The t-test checks the null hypothesis and concludes whether α_i is significantly different from 0 (hypothesis rejected) or not (hypothesis accepted). [26] Backward elimination is about fitting the highest order model and then deleting terms, one at a time, starting with the highest order. This is done until the highest order remaining term has a significant t statistic. [25]

2.1.2.4 Nonparametric Regression

In nonparametric regression methods, there is no need to specify the form of the function, as data is used directly to make predictions. The goal is thus to estimate the function itself. Kernel regression and locally weighted regression (Loess) are discussed. [25]

Kernel regression extends the weights of an ordinary regression using the kernel function K to predict $\hat{y}$ at a specific location x_0:

$$\hat{y} = \sum_j w_j \cdot y_j \,, where\ w_j = \frac{K(x_j - x_0)}{\sum_k K(x_k - x_0)}\ and \sum_j w_j = 1 \tag{2-7}$$

These kernel functions have properties of symmetric probability density functions:

- $K(t) \geq 0$ for all t
- $\int_{-\infty}^{\infty} K(t)dt = 1$
- $K(-t) = K(t)$ for all t

Common kernel functions are the Gaussian kernel function, the triangular kernel function and the uniform kernel function. [25]

In a locally weighted regression, data around the neighbourhood of the location x_0 is used, just as in kernel regressions. This neighbourhood is defined by the span, which is a fraction of the total points. This neighbourhood's points are used to generate a weighted least squares estimate of y at x_0. Software packages usually use a tri-cube weighting function to assign weights for each point in the neighbourhood of x_0:

$$w_j = W\left(\frac{|x_j - x_0|}{\Delta(x_0)}\right)\ with\ W(t) = \begin{cases} (1-t^3)^3, & 0 \leq t \leq 1 \\ 0, & t > 1 \end{cases} \tag{2-8}$$

The advantage of this method is its flexibility to different locations in a scatter plot; however, it is computationally intensive. [25]

With regressions being such a fundamental topic in statistical analysis and machine learning, the methods it encompasses have various benefits and drawbacks. A simple linear regression has the advantage that it is easily understandable, and it rarely overfits. It is, however, seldomly recommended because it oversimplifies real world problems. Multiple linear regression gives a deeper insight in that regard; however, its complexity and the required knowledge are higher. Additionally, the sample size needs to be higher to get a higher confidence level on the analysis outcome. [27] Polynomial functions are able to demonstrate more complex relations, but they also often tend to overfit the data. [28] Nonparametric algorithms minimise the risk of model misspecification, on the one hand. On the other hand, they are sensitive to atypical and outlying observations in the data. [29]

2.1.3 Decision Trees

Decision trees are based on the – in computer science known and fundamental – data structure of a tree. In its simplest form (figure 2-2), a decision tree has a root (R), where the evaluation starts. At the end, there are leaves (L), which are nodes where no decision is made, but a state is classified. A decision is made at every node (N), and the branches are followed appropriately until a leaf is hit. [21]

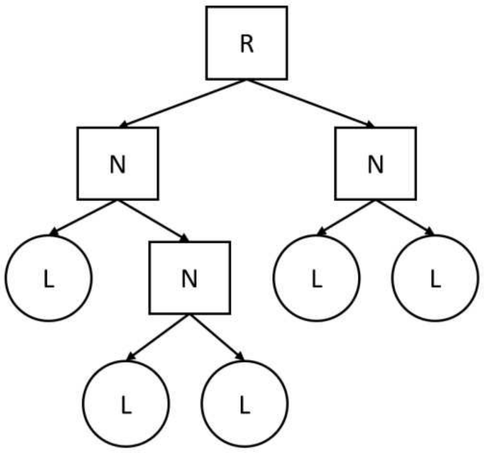

Figure 2-2: A simple decision tree (own illustration based on [21])

A priori, no parametric form is assumed, and the tree structure is not fixed. Instead, the tree grows during learning, in the sense that nodes and leaves are added. A decision tree breaks down a complex function into a series of simple decisions. The advantages of decision trees are their readability and interpretability. Due to the hierarchical placement of decisions, it is easy to localise an output and follow the path it took from the input. The tree’s nodes can be converted to "if-then" rules that make it easily understandable. Therefore, decision trees are prevalent and often used instead of more accurate methods that are harder to interpret. [18]

2.1.3.1 Univariate Trees

Each node uses only one of the input dimensions in univariate trees and implements an n-way split. That means a threshold w_{m0} discretises a numeric input (x_j):

$$f_m(x) = x_j \cdot w_{m0} \tag{2-9}$$

The input would then be split into two spaces, a so-called binary split. The construction of a tree with a given sample is called tree induction. There are many options a tree can be constructed with no errors. Usually (and due to simplicity) the smallest among them is of interest. Size is measured as the number of nodes and the complexity of the nodes. Finding the smallest decision tree is a computational problem for which no efficient solution algorithm has been found and where a heuristic approach is used. [18]

An impurity measure quantifies the goodness of a split in *classification trees*. Purity can best be described by imagining a node m that N_m training instances have reached. N_m^i belong to the class C_i (with $\sum_i N_m^i = N_m$). The estimated probability p_m^i of class C_i is:

$$p_m^i = \frac{N_m^i}{N_m} \tag{2-10}$$

Node m is called pure if p_m^i for all i is 0 or 1. In the case of purity, no further splits need to be conducted, and the leaf is labelled. Possible measures for impurity are entropy, the Gini index, or the misclassification error, but research has shown that there is not a significant difference between these measures. If node m is not pure, the instances should split to decrease impurity. The split that minimises impurity is important, as this returns the smallest tree. So, the closer to pure, the fewer splits will be needed, at least from a local perspective. For numeric attributes with N_m data points, $N_m - 1$ splits are possible; however, one does not need to check for all points, as it is enough to test at halfway points. The tree construction continues recursively and in parallel until all the branches are pure. [18]

As a tree grows until it is purest, it is possible to grow a huge tree. To alleviate this problem tree construction ends, when it is pure enough. This means that p_m^i is not precisely 0 or 1 but close enough (according to a threshold). [18]

A *regression tree* is constructed almost identically to a classification tree; however, it works with continuous values and the impurity measure used is a measure for regression, such as the mean squared error (MSE). This impurity measure can be calculated using an estimated value such as the mean or median of the required outputs of instances reaching a node. Data reaching the node is split further if the error is not acceptable. As in classification, at each node, the split threshold that minimises the error is computed, and the tree induction continues recursively. [18]

2.1.3.2 Multivariate Trees

Contrary to a univariate tree, all input dimensions can be used at a decision node in multivariate trees. The multivariate node takes a weighted sum, so discrete attributes should be represented by 0/1 dummy variables. A simple equation can define a hyperplane, as seen in figure 2-3. More nodes would define a polyhedral in the input space. The univariate tree does the same thing in essence; however, the linear discriminant is orthogonal to the axis. With this knowledge, it is also possible to understand that an exhaustive search for possible hyperplanes is no longer practical with multivariate nodes. However, multiple algorithms exist for learning multivariate decision trees. [18]

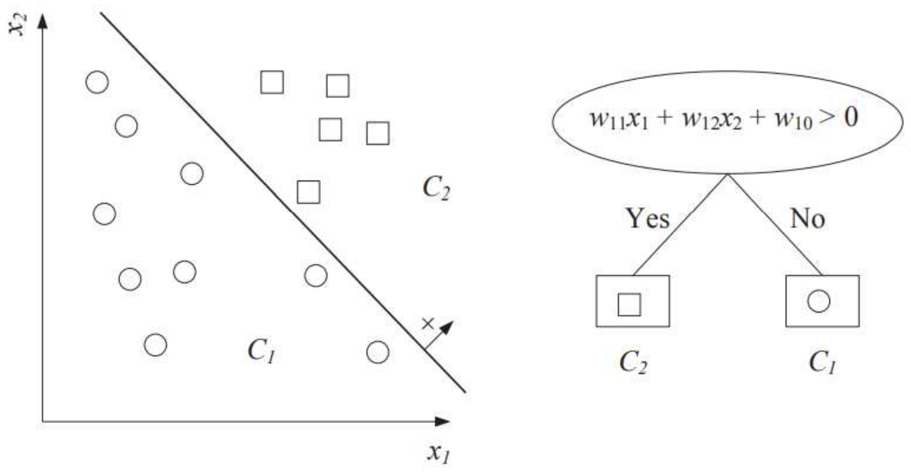

Figure 2-3: A linear multivariate decision tree [18]

Decision trees are commonly used to classify data. They have the advantage that they are suitable for regression as well as classification and they can be easy to interpret. Their disadvantage lies in them possibly being unstable, the problem of overfitting, and the tree size becoming too large. A special ensemble method called random forests can be used to prevent overfitting and increase accuracy. [27, 30]

2.1.4 Naïve Bayes

Naïve Bayes classifiers are robust, easy to train, and determine the probability of an outcome using Bayes' theorem. The approach is relatively simple, and the algorithms are still very efficient and not limited. While Naïve Bayes is used in many different situations, their performance is especially good when probabilities of some causal factors determine the probability of a class. The method is based on Bayes' theorem. To derive Bayes' theorem, one must consider two probabilistic events, A and B, with probabilities $P(A)$ and $P(B)$. When the product rule is applied:

$$\begin{cases} P(A \cap B) = P(A|B)P(B) \\ P(B \cap A) = P(B|A)P(A) \end{cases} \tag{2-11}$$

The two terms in (2-11) are equal, which leads to Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2-12}$$

This formula is fundamental to statistical learning because it combines a kind of a priori probability (meaning there are no other known values), often determined by mathematical considerations or a frequency count, with a posteriori probability (meaning it is estimated after knowing some additional elements). [14]

When there are multiple concurrent conditions, conditional independence is often assumed. This means that the effects of every cause are independent of one another and allows the expression (with a normalisation factor α) to be written as:

$$P(A|C_1 \cap C_2 \cap \dots \cap C_n) = \alpha P(C_1|A)P(C_2|A) \dots P(C_n|A)P(A) \tag{2-13}$$

The assumption of conditional independence is also the reason why the Naïve Bayes classifier is called naïve. However, research has shown that it is not rare that different dependencies cancel each other out. [14]

The merits of Naïve Bayes are its ease of implementation, its insensitivity to irrelevant features, and its scalability. Demerits are that it is often too simple and the fact that continuous variables have to be categorised into buckets. Examples of use cases are recommendation systems and forecasts of cancer relapse after radiotherapy. [27]

2.1.5 Support Vector Machines

During the 90s Vladimir Vapnik presented the basics of today's SVMs. A hyperplane is thereby used to separate and classify two datasets. This can be illustrated using a function $g(x)$, where x are the different parameters and w the weights of the parameters: [21]

$$g(x) = b + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n = b + w \cdot x \tag{2-14}$$

Creating machine learning models with SVMs is often about finding the optimal hyperplanes for separation. This is the case when the distance between the hyperplanes and the elements of the datasets is as big as possible. By rewriting and adapting equation (2-14) it is possible to show that:

$$\frac{y(b + w \cdot x)}{\|w\|} \geq \frac{1}{\|w\|} \tag{2-15}$$

where y is a code that allows for both positive and negative signs. The right side of the equation is defined as margin and signifies the distance to the hyperplane. Only those data points closest to the hyperplane define its position and are known as support vectors, which can be seen in figure 2-4. Instead of maximising $\frac{1}{\|w\|}$, which can be time-consuming, algorithms usually minimise $\|w\|^2$ to calculate the largest possible margin. [21]

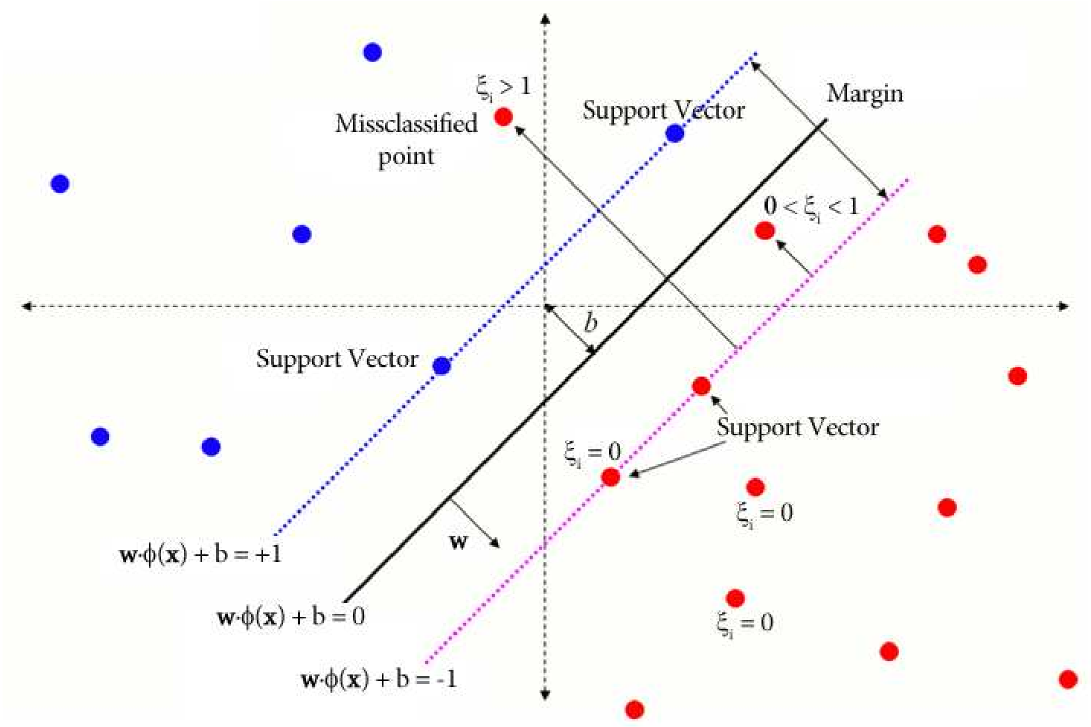

Figure 2-4: Construction of an SVM as a flat-affine 1-dimensional subspace [31]

Although this allows for hyperplanes to solve problems that can be linearly separated, problems and data sets cannot always be analysed and classified in this manner. There are two approaches to this, namely *soft margins* and *kernel tricks*. [21]

2.1.5.1 Soft Margin

In the case of the approach from above, there are two possible problems: firstly, data sets that lie on the wrong side even at the best level and are thus incorrectly classified, and secondly, data sets that lie on the right side but do not fulfil the condition (2-15) and are too close to the decision boundary. Both problems are demonstrated in figure 2-4. In order to solve the problem, slack variables ξ_i are defined. This leads to the following condition:

$$y_i(b + w \cdot x_i) \geq 1 - \xi_i \tag{2-16}$$

$0 < \xi_i < 1$ means that x_i was classified correctly, while $1 < \xi_i$ means that there was a misclassification. In the best case, ξ_i equals 0. The sum of ξ_i is called the *soft margin*. While this approach extends the application of SVMs, their true performance is generated by applying more dimensions to data sets (instead of simplifying them). [21]

2.1.5.2 Kernel Tricks

It is often helpful to transform the original vectors into a higher dimension to separate them linearly, even if that adds complexity. For this, a function Φ is defined that transforms the variables that need to be analysed. An example explains this concept: Three variables need to be transformed into a higher dimension, which can be done through the multiplication of the variables with each other so that:

$$\Phi(x) = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3) \tag{2-17}$$

However, the introduction of this transformation increases the computational effort, as the following term is used to further calculate in SVMs with multiple parameters:

$$\Phi(\bar{x}_i)^T\Phi(\bar{x}_j) \tag{2-18}$$

This term is unacceptable for larger problems, which is why there are particular functions K with specific properties. These are called *kernels* or *kernel functions*: [21]

$$K(\bar{x}_i, \bar{x}_j) = \Phi(\bar{x}_i)^T\Phi(\bar{x}_j) \tag{2-19}$$

The idea is to replace the functions' product with a kernel function. The kernel function is directly applied in the original space instead of mapping two instances to another dimension and calculating the dot product. [18] With this, the computational effort remains almost the same. It is important to note that there are multiple kernels, such as the Radial Basis Function kernel, Polynomial kernel, Sigmoid kernel, and more. [14] A graphical explanation of the procedure can be seen in figure 2-5, whereby the dataset seen in a is transformed by adding another dimension, as shown in b. A hyperplane then separates the data as demonstrated in c, and the resulting separation of the data is shown in d.

Following are the advantages of an SVM: It can handle complexity if the appropriate kernel function can be derived, and it can scale up with high dimensional data. Disadvantages are the decreasing performance with increasing data due to higher training time, finding the proper kernel function, and bad compatibility with noisy data sets. Though an SVM can handle both classification and regression (also called support vector regression) problems, its main applications include classification problems like cancer diagnosis, face detection, and credit card fraud detection among others. [27]

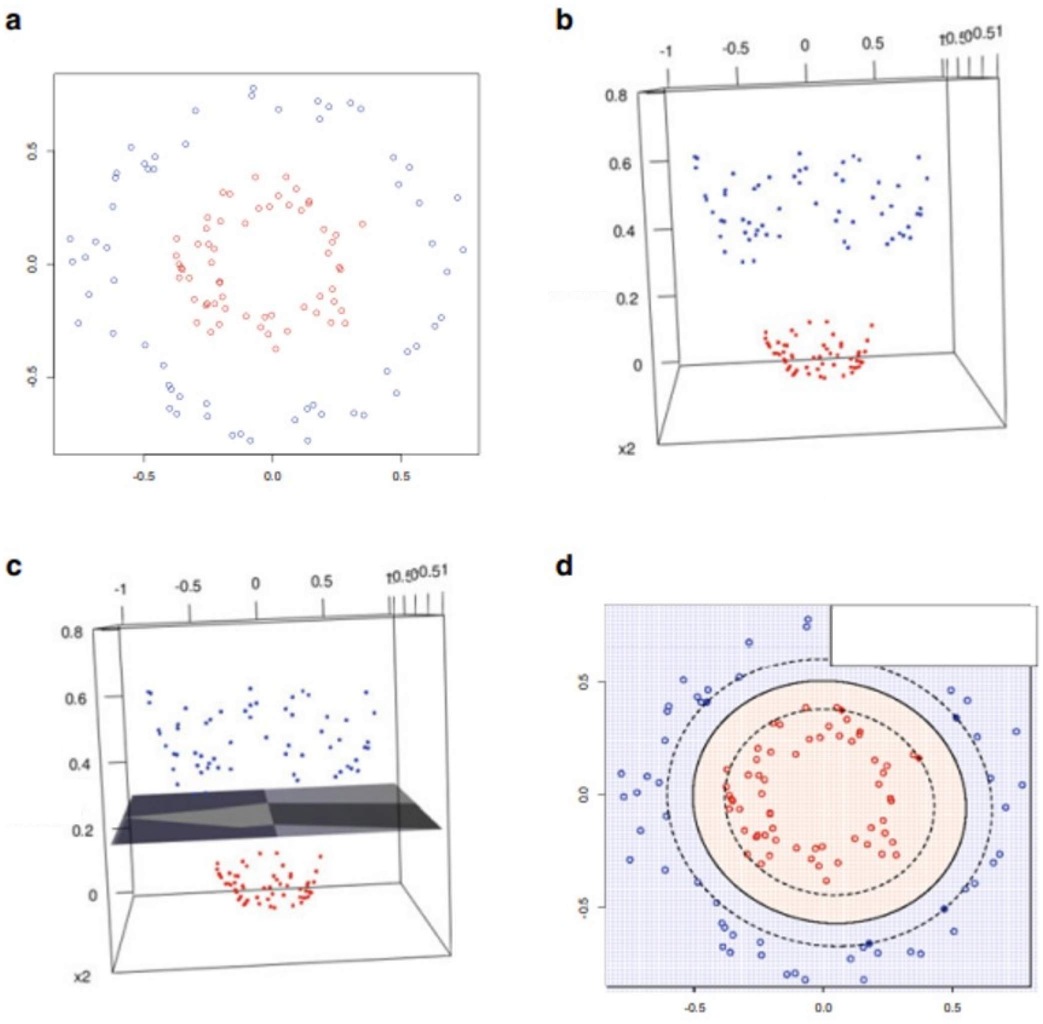

Figure 2-5: Application of a kernel function to a set of 2-dimensional data [12]

2.1.6 Neural Networks

Often expert knowledge is required when applying machine learning to real-life data to reduce the number of dimensions of to-be-analysed data. An example of this would be reducing raw data of a picture (1,000 × 1,000 pixels = 10^6 dimensions). This picture would be compromised, and for every pixel, features like colour, sharpness, and colour of neighbours are compiled to bring the dimensions down to 1,000. Using this procedure, SVMs and decision trees deliver good results, given that no information loss occurred. Neural networks, in contrast, directly work with high-dimensional training data and do not need this reduction through expert knowledge. Neural networks build flexible classes of functions to approximate continuous functions of which the structure and the evaluation resemble that of a nerve cell (this is where the name neural network comes from). While the goal is also to ultimately reduce the amount of information, the reduction happens through training data itself and not through expert knowledge. This reduced dataset then undergoes a regression. The reduction thereby happens through steps of the same nature: in every step, the data vectors are modified with a simple transformation, and the data dimensions are adjusted. [12]

The *perceptron* (also often referred to as a node) is the basic processing element of a neural network and has inputs coming from the environment or other perceptrons. Each input x_j has a weight w_j associated with it:

$$y = \sum_{j=1}^{d} w_j \cdot x_j + w_0 \tag{2-20}$$

where w_0 is a bias value. The perceptron defined in this equation (2-20) defines a hyperplane, and two classes can be separated by checking the output sign, for example. A simple single-layer neural network with multiple outputs can be seen in figure 2-6. [18]

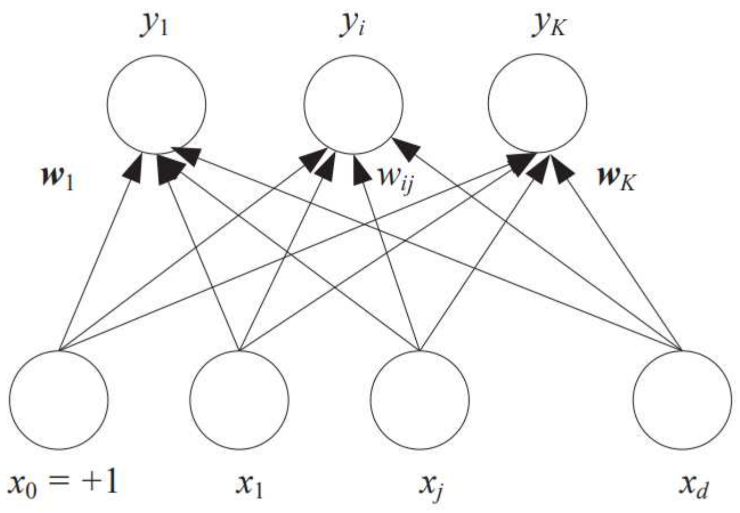

Figure 2-6: A single-layer neural network [18]

To implement a given task, it is necessary to compute the weights of the neural network parameters such that correct outputs are generated given the inputs. [18]

Usually, neural networks have hidden layers between their input and output layers. These so-called multilayer perceptrons (MLPs) are the most popular network architecture: the units each perform a biased weighted sum of inputs and pass this into a transfer function to produce output. [11] This function, also called the activation function filters the resulting sum from the calculation and produces an output based on its input. In practice, rectified linear unit (ReLU) functions are frequently used. When the number of hidden layers is not too large, the sigmoid function is often used. [12, 14] Figure 2-7 shows a simple multilayer neural network with two hidden layers. The hidden layers of the nodes ${x_i}^{(1)}$ and ${x_i}^{(2)}$ and the output layer $g(x)$ can have activation functions.

The units are thus arranged in a *feedforward* topology, meaning information only flows from the input to the output layer, and it therefore provides a simple interpretation with the weights and thresholds as free parameters of the model. This leads to neural networks being able to model functions of almost arbitrary complexity, with specifications being the number of hidden layers and the number of nodes in these layers. While the problem usually predefines the input and output units, the number of hidden layers is unclear. An appropriate starting point for the number of layers is one, and for the number of nodes per layer is half the input units. After selecting the number of layers and units, training algorithms minimise the prediction errors made by the network using historical data. This historical data can also be used to determine the error of a neural network by comparing the actual output to the desired output. [11]

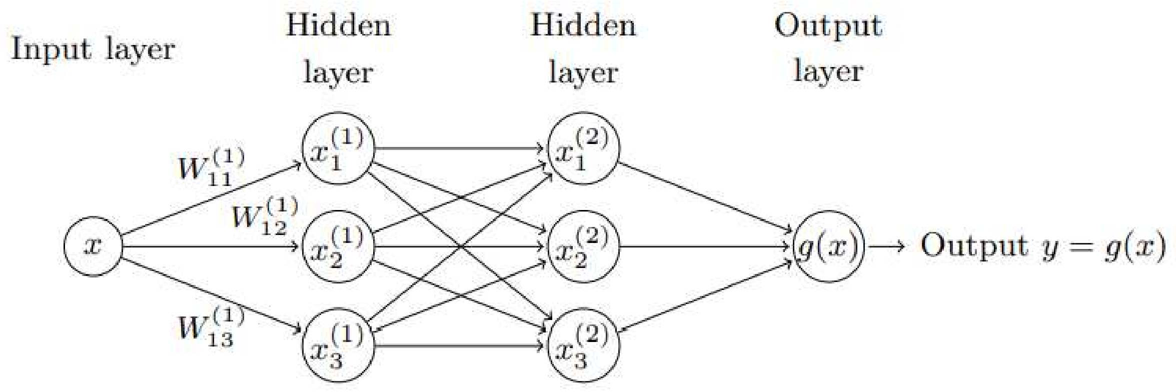

Figure 2-7: A feedforward neural network with two hidden, one input, and one output layer [12]

The best-known example of a neural network training algorithm is *back propagation*. In back propagation, the gradient vector of the error surface is calculated. The error surface is any possible configuration of weights and thresholds, with each of them being a dimension in space. This vector points towards the steepest descent from the current point, often initialised with 0 for thresholds and random decimal numbers for weights. Moving along this steep descent will decrease the error; however, it is difficult to decide how large the steps should be. In practice, the step size is proportional to the slope and learning rate (an input that determines the magnitude of change to the step size), application-dependent, and typically chosen by experiment. [11] The algorithm usually also includes a momentum term, a mathematical term that gives the step size momentum (in analogy to physics) and sometimes helps escape local minima and move over plateaus. [21] Figure 2-8 shows the difference between a global and a local minima. The blue line, in this case, indicates the step size of the algorithm. Since it bounces from one side of the valley to the other, it is set too large in this example.

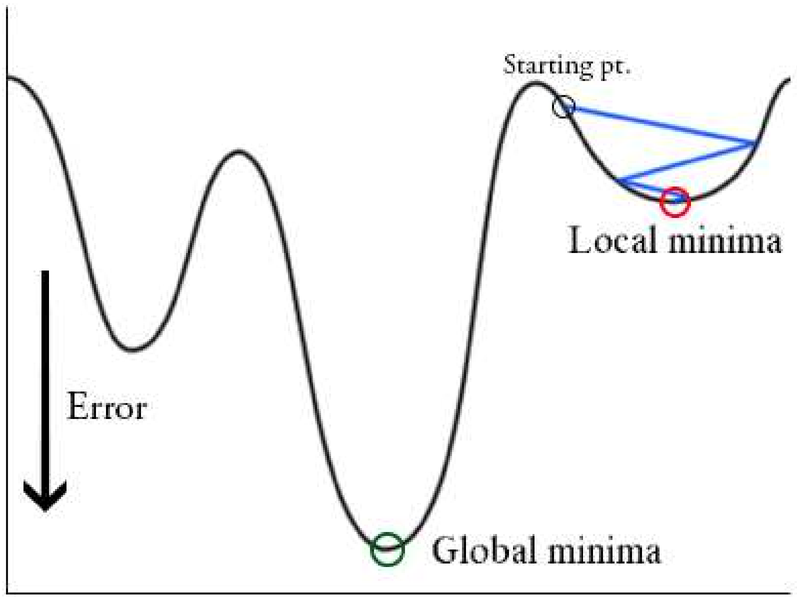

Figure 2-8: Demonstration of the local minima and the learning rate via a gradient descent algorithm applied to an error function [32]

Machine learning algorithms learn and adjust their internal parameters, called *model parameters*, depending on the input provided. However, there are also other types of parameters that have to be preconfigured, so-called *hyperparameters*. Hyperparameters cannot be estimated by the model from given data during training or validation. Some examples are the number of layers, the number of nodes in a layer, or the aforementioned learning rate. They play an important role as a change in hyperparameters leads to a change in the model's performance. Depending on which hyperparameter and by how much it was adjusted, the change in performance can be significant. [33]

Overfitting has been described in section 2.1.1 and is a fundamental concept to understand when modelling with neural networks. After all, an extensive network will almost always achieve a lower error, but this may be more due to overfitting than good modelling. To maintain the quality of the model, it can be checked against an independent data set that is not used for training but for checking the progress of the algorithm (validation set). Often even a third set - the test set – is used to ensure that the results are not artefacts of the validation and training sets. On the other hand, if under-learning occurs through the algorithm and it does not achieve the desired performance level, more nodes should be added to the hidden layers. If this does not help, extra hidden layers can be added. [11]

It has to be noted that not all neural networks use a feedforward topology and the backpropagation algorithm for learning; they are, however, the most prominent ones. In general, artificial neural networks (ANNs) can be classified into various sub-categories. Well-known ones are shown in figure 2-9, with three major ones being multilayer perceptrons

(MLPs), denoted as Deep Feed Forward, convolutional neural networks (CNNs), and recurrent neural networks (RNNs). [34]

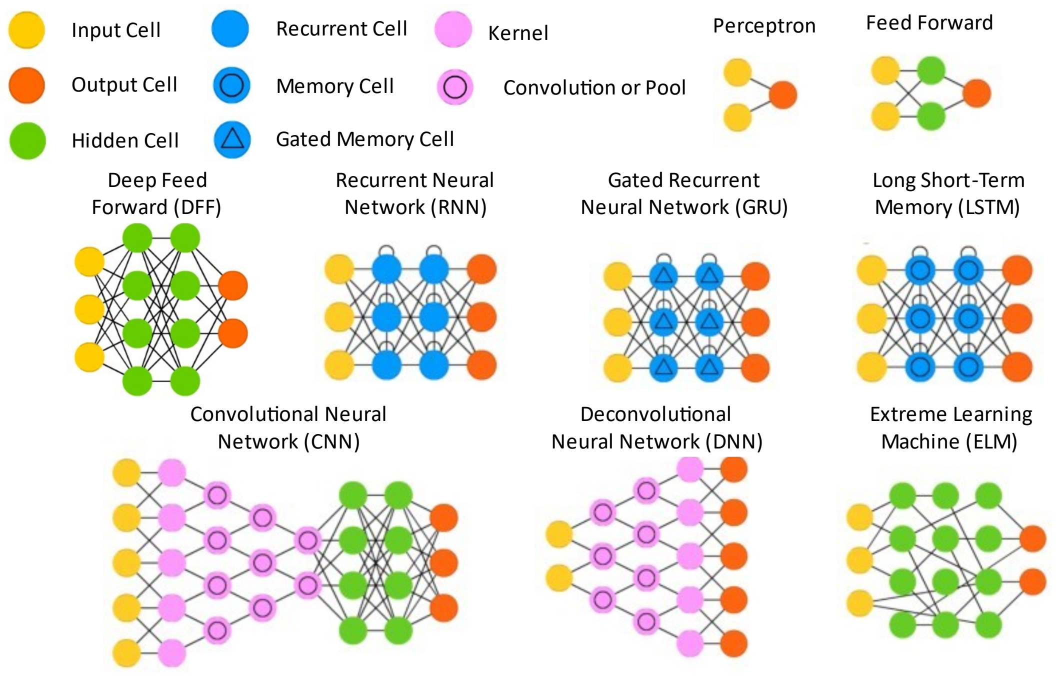
Input Cell
Output Cell
Hidden Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron
Feed Forward
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Gated Recurrent Neural Network (GRU)
Long Short-Term Memory (LSTM)
Convolutional Neural Network (CNN)
Deconvolutional Neural Network (DNN)
Extreme Learning Machine (ELM)

Figure 2-9: An overview of different neural network architectures (own illustration based on [35])

MLPs are one of the most popular of types neural networks, which Paul Werbos developed in 1974 and are also often called feedforward networks. CNNs consist of convolution layers, pooling layers, and fully connected layers. They are primarily used in pattern recognition within images and encode image-specific features into the architecture. The critical difference to MLPs is, therefore, that the layers in CNNs are comprised of neurons organised into three dimensions (height, width and depth). These neurons only connect to a small part of the preceding layer. [36] An RNN is best suited for recognising patterns in data sequences and is heavily used to classify, cluster and make predictions about data. Similar to MLPs, the architectures consist of input, hidden, and output layers; however, the nodes of hidden layers are interconnected and create so-called loops. This ensures that the output at a certain time step depends on previous inputs. Long short-term memory networks (LSTMs) deal with problems of RNN architectures, such as vanishing and exploding gradients causing the RNNs to train slowly, and are therefore capable of learning long-term dependencies. [37] They do so by adding gates to an RNN. These gates are composed of a sigmoid neural net layer and a pointwise multiplication operation. An LSTM cell (figure 2-10) takes the cell (upper horizontal line) and hidden (lower horizontal line) state of the previous time step as well as the input variables as input and returns the hidden state as output, while both the hidden state and the cell state are transferred to the LSTM cell of the next time step. [38]

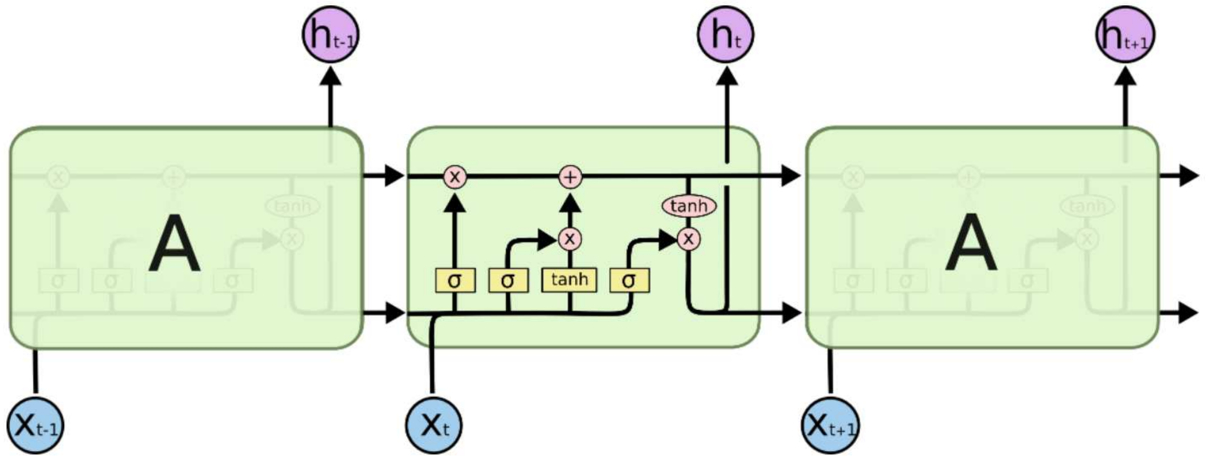

Figure 2-10: A LSTM cell containing three gates and using cell and hidden states [38]

The benefits of using neural networks are their easy adoption to new scenarios, fault tolerance, and ability to handle noisy data. Disadvantages are that training times are very long, they are non-transparent due to their black-box like behaviour, and the sample data sets need to be large. Neural networks are used in different industry segments, especially when there are no well-defined criteria or rules to find an answer to a problem. Applications are various kinds of classifications or forecasts of market dynamics. [27]

2.1.7 Clustering

In machine learning, clustering is a method of unsupervised learning, which means that, contrary to supervised methods, there is no validation or testing data and, therefore, no target data. Cluster analysis aims to divide a set of objects into groups while maximising the similarities inside the groups and maximising the differences between the groups. There are multiple clustering mechanisms, the most popular ones being k-Means and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). [21]

2.1.7.1 k-Means

The k-Means algorithm is based on assigning each data point in a data set to one of k clusters defined by centroids: After a first initialisation of the centroids, the distance between each data point and centroid is computed, and the data point is assigned to the closest cluster. This process is iterative, as once the samples have been processed, the centroids are computed again, and the distances are recomputed. [14] Mathematically this can be defined as minimising the so-called inertia J via the Euclidian distances of the data points:

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - \mu_i \right\|^2 \tag{2-21}$$

where k stands for the number of clusters, x_j for a data point, C for the cluster and μ_i for the centroid. [21] The clustering process is stopped once the desired tolerance is reached and the centroids become stable. This process is illustrated in figure 2-11. Methods have been researched to improve the convergence speed; one of them is k-Means++ which selects the initial centroids so they are statistically close to the final ones. [14]

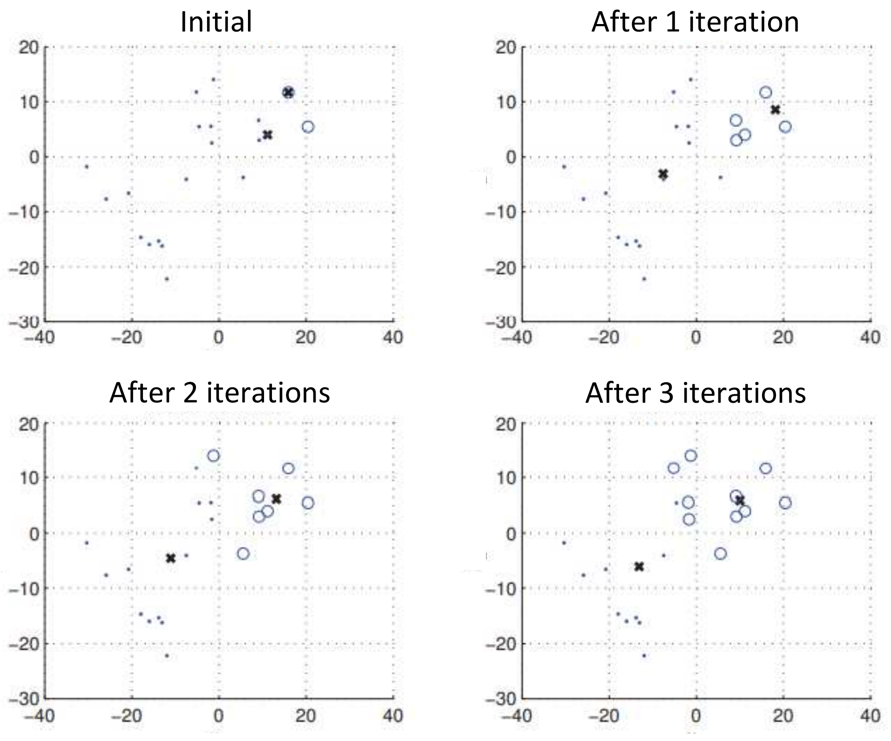

Figure 2-11: Evolution of k-Means over iterations: the black crosses indicate centre positions (centroids); the data points are marked depending on the closest centre as a circle or dot [18]

One of the difficulties of working with k-Means clustering is finding the correct number of clusters. An assumption may be that the number of clusters must produce the smallest inertia; however, the absolute minimum of inertia is reached when the number of clusters equals the number of data points. Therefore, one can look at a trade-off between the number of clusters and the inertia. Figure 2-12 shows such a diagram for an exemplary dataset. There is a considerable reduction in inertia from two to three clusters. After the fifth one, any cluster only leads to marginal inertia decrease and is likely to produce intracluster splits. A recommendation is thus to work with four or five clusters in this case. This is one of the more straightforward methods of defining the number of clusters. More complex ones are the Silhouette score or the Calinski-Harabasz index. [14]

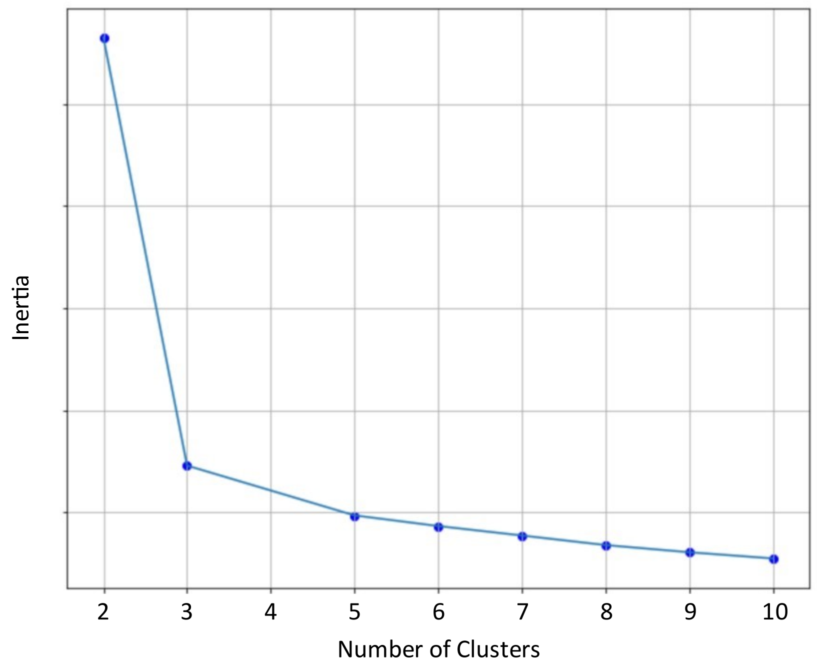

Figure 2-12: Plotting the inertia over the number of clusters in an exemplary dataset [14]

2.1.7.2 DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise and is an algorithm that, as the name suggests, works based on the density of data points. DBSCAN differentiates between three different types of points: core points, border points, and noise points. A data point can be classified into one of these categories depending on two parameters, ε and minPts. [21] ε is responsible for defining the maximum distance between two neighbours, and minPts determines the number of surrounding points necessary to define a core-point. [14]

The algorithm starts with an arbitrary point p and looks for all reachable points with respect to ε. If p is a core point with respect to the starting conditions (ε and minPts), a cluster is defined; if the conditions do not apply, the next point in the dataset is visited. The algorithm may also merge clusters into one if they are close to each other. This can happen when the distance between two sets of points S_1 and S_2 with core points p and q is smaller than or equal to ε:

$$\min\{dist(p,q) | p \in S_1, q \in S_2\} \leq \varepsilon \tag{2-22}$$

Consequently, a recursive call of DBSCAN is necessary, meaning the function must run multiple times to find the correct number of clusters. [39] A schematic overview of finding one cluster with DBSCAN can be seen in figure 2-13.

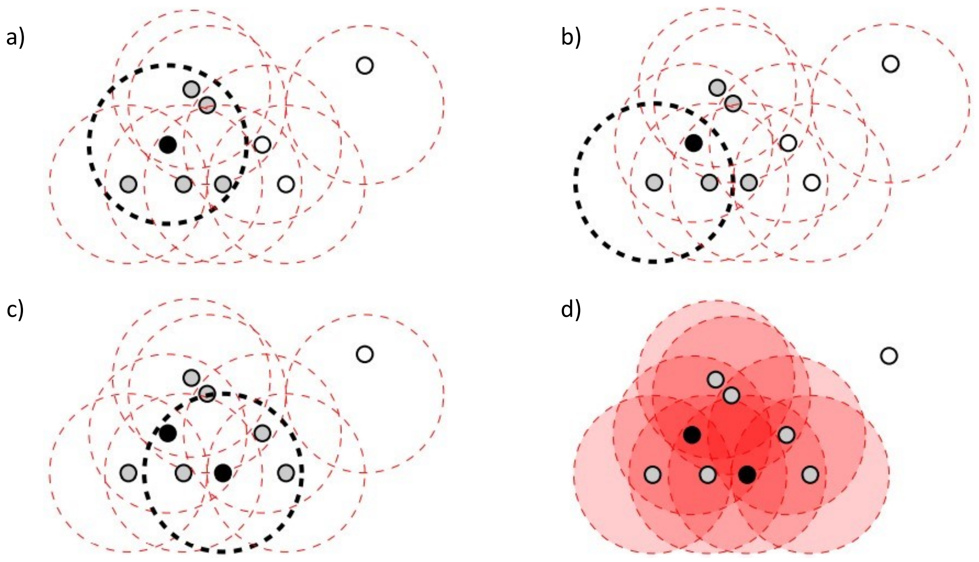

Figure 2-13: A graphical representation of finding one cluster via DBSCAN from a) to d): black dots are identified core points, grey dots are border points and white dots are noise points with the red circle demonstrating the ε of the points [21]

k-Means has the advantage of ease of implementation and interpretation. Furthermore, it is computationally efficient. A disadvantage may be that the prediction of the number of clusters k is difficult, and different initial starting positions impact the algorithm's performance. Use cases are found where data needs to be segmented or classified, such as document classification or customer segmentation and call record details analysis. [27] In contrast to k-Means, with DBSCAN it is unnecessary to define the number of clusters in the beginning, as the algorithm automatically finds the appropriate amount. Additionally, noise/outliers in the data set are detected, which is also its primary use. A disadvantage is the fact that DBSCAN is not able to scale very well. [21]

2.1.8 Reinforcement Learning

Reinforcement learning is a machine learning technique based on feedback from the environment. Thereby the learner is a decision-making agent that takes actions in the environment and receives feedback. This feedback is usually called reward, which is critical to understand whether an action performed in a state is positive or negative. A policy is the sequence of actions to always make the best decision in terms of reward. This policy should be learnt after a set of trial-and-error runs. The agent is thus modelled using a Markov decision process, with the difference being that in Markov models, there is an external process that generates a sequence of signals. In reinforcement learning, the agent creates this sequence of signals. Figure 2-14 shows the interaction the learning agent has with the environment. This technique is very efficient when the environment is dynamic, not wholly deterministic, and when there is no precise error measure. [14, 18]

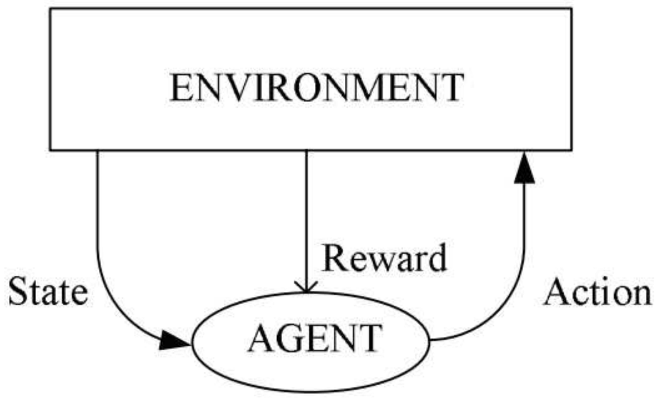

Figure 2-14: Interactions between the learning agent and the environment [18]

In contrast to other supervised learning methods, reinforcement learning is about learning with a critic that does not tell one what to do but how one has been doing in the past. This information comes late and is scarce (after solving or failing a problem, for example), leading to the so-called credit assignment problem. To overcome this problem and to assess individual past actions and find the moves that led to the reward, an internal value for intermediate states or actions is computed. This internal value can then be maximised to take local actions. [18]

In reinforcement learning, there is a difference between model-based and model-free methods. Model-based means that the agent tries to understand the concept and create a model to represent it, and during learning, it uses predictions of the environment's response. In a model-free method, on the other hand, the agent cannot make predictions about the next state and reward before taking action. [21]

Supervised and unsupervised learning may appear to exhaustively classify machine learning paradigms; however, this is not true. Compared to unsupervised algorithms, reinforcement learning does not try to find a hidden structure. Compared to supervised algorithms, it has the advantage of not needing training data of all the correct and representative states in which the agent has to act, but the agent can learn by itself. On the other hand, problems are the trade-off between exploration of the agent to find the best continuation in a certain situation and exploitation of the rewards in a certain situation. [40] Applications of reinforcement learning include finding winning tactics or strategies in games or letting an autonomous robot learn independently. [12] However, reinforcement learning has also been utilised to build bots that trade in and predict the stock market [41, 42] and has been used for predictions in the energy industry. [43]

2.2 Energy System Modelling

This section serves as a review of papers that further explore the application of machine learning for forecasting in industrial settings. The purpose is to combine the theoretical knowledge about machine learning described in section 2.1 with the real-life implementation of these methods in the iron and steel and energy industry. Best practices in energy demand forecasting, which can later be used to create the models for the steel mill Marienhütte, are discovered. The researched papers and the methods discussed are listed in table 2-1 in order of their appearance in this thesis.

Table 2-1: Notable papers about forecasting methods in the iron and steel and energy industry

Category	**Reference**	**Discussed Methods**						
		Regression	Decision Trees	SVMs	Neural Networks	Ensembles & Hybrids	Genetic Programs	Markov Chains
Overview of Forecasts in the Energy Industry	[44]		x	x	x	x		
	[45]	x		x	x	x		
	[46]	x	x	x	x			
Electrical Energy Forecasts in Steel Mills	[47]	x					x	
	[48]	x				x		
	[49]							x
Electrical Energy Forecasts with a Focus on Neural Networks	[50]				x			
	[51]	x	x	x	x			
	[52]				x			
	[53]				x			
	[8]				x			
	[54]				x			

Section 2.2 lists two forecasting methods that have not been described in the previous chapter: genetic programming and Markov chains. Genetic programming belongs to the

subclass of evolutionary algorithms. Genetic programs frequently have a tree-like structure with the nodes being function genes such as additions, subtractions, multiplications, divisions, or terminal genes (the parameters). Initially, random computer programs of various forms and lengths are generated. In a next step, they are varied through genetic operations like crossover and mutation over several iterations. After comparing the program with the experimental data, this process is repeated until a termination criterion is reached. Based on the literature, it is unclear, whether models created through genetic programming can be classified into machine learning, as they fail to learn from experience. Nonetheless, they are often applied to machine learning problems. [55] Markov chains, on the other hand, are types of stochastic processes, as they sample from probability distributions and are not part of machine learning subsets. [56] Despite these facts, the papers discussing these methods are presented, as they provide valuable information about possible alternatives to a forecast using machine learning and list relevant input parameters for EAFs.

2.2.1 Overview of Forecasts in the Energy Industry

Mosavi et al. [44] presented the state of the art of machine learning models used in energy systems. Different machine learning models were classified according to the modelling technique, energy type, and application area. The paper features a literature review (70 original papers) about these models and further assesses their performance. The authors identified ten major machine learning models frequently used in energy systems: ANNs, MLPs, extreme learning machines, SVMs, wavelet neural networks, adaptive neuro-fuzzy inference systems, decision trees, deep learning ensembles, and advanced hybrid machine learning models. [44]

By looking at the available literature with comparative performance parameters like the root mean squared error (RMSE) and the correlation coefficient r, the authors concluded that while there are some differences in the forecasting performance between certain models and none between others, hybridising methodologies and algorithm ensembles are the most effective ways to improve machine learning methods. The literary research suggested that hybridising existing methodologies improves the accuracy of energy demand forecasting significantly. Additionally, a hybrid model has a higher generalisation ability and lower forecasting uncertainty. Hybrid machine learning models are effective ways to improve machine learning models and will continue to deliver sophisticated energy models. [44]

In this context, it is vital to note that the applications of machine learning models reviewed in this paper range from energy consumption prediction, solar radiation prediction, power generation forecasting, storage planning, dynamic energy pricing, and more. While the applications reviewed are diverse, there was no connection to the metallurgy industry. [44]

Hahn et al. [45] gave an overview of different models and methods used to predict future load demands. The authors defined time periods under one week as short-term load forecasting (STLF), time periods ranging from one week to one year as medium-term load forecasting (MTLF) and from one year to up to 30 years as long-term load forecasting (LTLF). The paper differentiated between regression-based models, time-series approaches, neural networks, SVMs, and hybrid and other approaches. The methods were described, and studies on them were discussed; however, none of the studies had a connection to the metallurgy sector. [45]

The authors determined that regression models are relatively easy to implement, and the relationship between input and output is easy to understand. According to research, large ANNs can be seen as competitive with other models, and SVMs won a competition organised by the EUNITE (European Network on Intelligent Technologies for Smart Adaptive Systems) in 1999 to predict the load demand. Therefore, the authors concluded that support vector regressions have emerged as a, back then (2007), relatively new and competitive method for load forecasting and found that hybrid models were becoming increasingly popular. [45]

Statistical models for predicting the electricity consumption of EAFs were also reviewed and discussed by Carlsson et al. [46]. They aimed to explain the challenges and considerations that statistical models impose and they found that while nonlinear models outperform linear ones, they lack transparency with regards to which input variables have an influence on prediction of the electricity consumption. [46]

Multiple linear regression and partial least squares (PLS) as linear methods were reviewed. PLS has the goal of predicting the dependent variable from a matrix of independent variables and describing their common structure. It combines features from principle component analysis and multiple regression. [57] The following nonlinear methods were examined: ANNs, deep neural networks (DNNs), SVMs, and decision trees (including random forests). The total amount of papers reviewed for these methods was 21. In total, 22 parameters were used to create these models. They can be categorised into the following divisions: time, chemical, temperature, ladle furnace, material, and other variables. [46]

A general rule of thumb when creating a model is that a model with few coefficients requires fewer data points to converge than a model with more coefficients. Due to the limitation of available data, it is thus advisable to use a shallow ANN over a DNN when committing to a nonlinear model. Furthermore, the input variables should be selected in accordance to experience and science and their number should be relatively low to avoid collinearity. [46]

Transparent models reveal to what extent each input variable affects the prediction, and while multiple linear regressions are transparent (due to the coefficients and input variables in the mathematical equation), neural networks are black-box models and not transparent. Carlsson et al. argue that interpretable machine learning algorithms like feature importance (FI) and Shapley Additive Explanations (SHAP) are helpful to better explain predictions. [46]

The authors summarised their research, where 14 out of 15 studies used linear models and eight of 15 used nonlinear models. The most popular models in each category were multiple linear regressions and ANNs, respectively. Regarding the input variables, they concluded that additives such as lime, dolomite, and carbon and the scrap composition are rarely used in the studies reviewed, even though these variables account for 20 – 50% of the total energy requirement. Only one of the studies completely specified what their data treatment looks like. Based on the literary research, it was recommended that the model performance should be based on predictions on future data relative to training data and the models should be in line with principles of physics and chemistry. [46]

2.2.2 Electrical Energy Forecasts in Steel Mills

Kovačič et al. [47] analysed the per batch electric energy consumption of an EAF during its melting and refining stage. The two approaches they took were creating a linear regression and a genetic programming model. [47]

The model was created using 3,248 consecutive batches and 278 batches were used to validate the model. The furnace was typically charged with three baskets with the capacity of 22 – 30 t, 15 – 20 t, and 6 – 15 t, respectively. Each charging lasted about three minutes, and each melting lasted about 20 min, 15 min, and 10 min for the three baskets. The dataset used included 26 process parameters, including the electric energy consumption. [47]

The linear regression model showed that $R^2 = 0{,}63$ and the model significantly predicted the electric energy consumption. 8 out of the 25 independent variables did not pass the null hypothesis significance test ($p > 0.05$). The most influential factors were E-type scrap, low-alloyed steel (moderate chromium content), scrap packets, oxygen consumption during melting, natural gas consumption, limestone, other technological delays, and coke injection during refining. [47]

Genetic programming is a general evolutionary optimisation method. During the random generation of the programs the previously generated linear regression was used. According to this model, dolomite, E-Type scrap, low-alloyed steel (moderate chromium content), other technological delays, and coke injection were the most influential factors. Coke, lime, limestone, scrap-charging, reparation of the linings with dolomite or magnesite, electrode settings, chemical analysis delay, oxygen and temperature analysis delay, and the delay during tapping were not considered in the model. [47]

The results show that the average relative deviation between the experimental data and the linear regression model was 3.65%, while that between the experimental data and the genetic programming model was slightly lower at 3.49%. The authors consequently concluded that it is possible to use the approaches for precise EAF energy consumption forecasting on a per batch basis. [47]

A linear regression model, with the nonnegative least squares method applied, was used by Zhou et al. [48] to forecast the daily electricity consumption of a large steel corporation. Additionally, the relevant features were selected using the random approximated greedy search (RAGS) and an ensemble model was built by bagging approach. [48]

The daily electricity consumption data with corresponding maintenance scheduling and daily production planning was collected in a 9-month time period, totalling 258 samples. Maintenance scheduling contained 87 features; daily production planning contained 88, which leads to a total of 175 features. 80% of the samples were used as training set and 20% as validation set. The electricity consumption was inversely proportional to maintenance duration and directly proportional to production quantity; this information was included in the linear regression model. [48]

Firstly, the linear regression models without nonnegative coefficient constraints were trained; however, the regression coefficients obtained were neither stable nor reasonable. Nonnegative coefficient constraints led to stable regression coefficients within a reasonable rangel. With a mean absolute percentage error (MAPE) of 2.37%, the forecast results proved effective. [48]

By adding 20 independent random features to the model as probe indicators, it was analysed whether the linear regression selects the correct features to build the model. This was not the case, so to avoid unrelated features in the model and choose appropriate ones, a feature selection, in this case the RAGS algorithm, which belongs to the mathematical field of ordinal optimisation, was implemented. [48]

In a final step, the ensemble model is built by bagging approach, which is used to improve the stability of forecast models. It uses the mean (or median) of the output of a group of forecast models for the final forecast value. This usually leads to a decrease in variance while keeping the bias of the original models. The ensemble model was composed of 20 linear regression models using the 20 chosen feature subsets. The final result showed that the MAPE has improved to 1.75%. While there was no discussion about what features were used for the model, the paper contained a clear procedure on how to develop a regression model to forecast the daily electrical energy consumption of a steel corporation. [48]

Dock et al. [49] presented an energy system model of an EAF steel mill using a modular design. This allows the model to be adopted for other electric steel mills to a certain extent. The model was developed holistically, focusing on being time and technologically resolved. [49]

During research for their paper, the authors noted, that while several energy consumption models have been used to predict the future energy demand of individual steel mill components, the models do not allow time-resolved calculation, which is why they were not used in this study. Their work applied two approaches: stochastic modelling of process parameters and load profile generation with Markov chains. The sub-models of the considered system components were built using Python with its libraries Pandas, NumPy and Matplotlib. [49]

The Pearson correlation coefficient between the energy consumption and the parameter defined determines whether this parameter should be considered when modelling or not. For the EAF, the scrap mass, and the tap-to-tap time were identified as vital input parameters. Therefore, a linear regression was calculated to evaluate the amount of energy needed based on a stochastic model of the scrap mass distribution. The energy input is then allocated to a Markov chain, of which the length corresponds to the tap-to-time (also determined stochastically). Markov chains predict future states of a system based on historical data with the help of a so-called transition probability matrix. The simulation time steps of the model have a length of 60 s and the data for them were obtained through measurements of 130 batches in the steel mill Breitenfeld Edelstahl AG in Austria. [49]

Models were created for the EAF, the LF, induced draft fans, LHs, vacuum treatments, annealing furnaces and other consumers, composed of several smaller consumers and considered baseload. The model results were compared to actual values of the year 2018 over four production weeks. The MAPE for the overall energy system was 1.6%, with the highest and second-highest MAPE being the other components (10.9%) and the LFs (8.0%), respectively. [49]

2.2.3 Electrical Energy Forecasts with a Focus on Neural Networks

Gajic et al. [50] applied a neural network model to predict the batch-wise electrical energy consumption based on the chemical composition of the charge material mix of an EAF. For the experimental part, 46 melts were run with different chemical compositions, of which 32 were assigned to the training set, seven to the testing set and seven to the validation set. Carbon, chromium, nickel, silicon and iron were measured with other properties such as scrap type, density, molten steel's final temperature and injection of chemical energy being kept at almost constant levels to minimise their effects. [50]

The neural network in the paper was trained by Statistica 8.0 (Stat Soft, Inc.), with the Broyden-Fletcher-Goldfarb-Shanno algorithm being used to compute the weights most efficiently. The activation functions were hyperbolic tangent and linear for the hidden, and output layers. The optimal architecture for the neural network (5-5-1) was obtained after 89 cycles. [50]

The proposed model explained 92% of the variation in the specific electrical energy consumption for the training, test and validation data (overall $R^2 = 0.92$). The model confirmed that the charge's chemical composition plays a significant role in to electricity consumption in the EAF. It was found that carbon content has a greater impact on the consumption than iron content. The decreasing order of the impact on the electrical consumption is: carbon content > chromium content > iron content > nickel content > silicon content. [50]

A neural network approach was also used by Chen et al. [51] to establish a prediction model for an EAF. As the dataset was large and multidimensional with 40 attributes and 10,990 instances, a deep learning method was used for its increasing performance due to its capability to learn hidden patterns. Additionally, statistical models, like multiple regression, often have problems with impurities in real-life data sets and become inefficient as the training data grows. However, the deep learning model was compared to other methods regarding their technical performance. [51]

As the model's input parameters, scraps and additives were added at once since their mass directly affects the energy consumption. An attribute selection software called Weka selected the rest of the features, which were PON time (min) and TTT level 2. However, no further explanation of their meaning was given. Altogether, 16 attributes were thus selected. As the data was recorded continuously, it was necessary to randomise the dataset to prevent the model from learning hidden patterns in the time series. In a last data preparation step, the dataset was normalised. [51]

The deep learning model was built using Keras, an advanced deep learning package of Python. As an optimiser for the gradient descent problem, "Adam" was selected, the number of hidden layers was set as four, and the number of nodes was set to 500. Linear regression, an SVM and a decision tree were used to compare the accuracy of the neural network. These used the same in- and output features as the deep learning model and were implemented with Python's Scikit-learn package. K-fold validation was used with K being set to 5. [51]

The deep learning model presented excellent performance. The correlation coefficient of the linear regression (0.785), the SVM (0.762) and the decision tree (0.775) was lower than that of the neural network (0.854). Additionally, only 10.6% of the instances had an absolute error over 3 MWh (linear regression: 22.3%; SVM: 14.9%; decision tree: 19.9%). The paper, therefore, concludes that the model is an effective tool for predicting the daily energy consumption of an EAF. [51]

The process of developing an ANN to predict the electrical energy consumption of an EAF and using statistical tools to investigate the most influencing input variables were documented by Carlsson et al. [52]. A nonlinear model was used as the nonlinear elements of the EAF process make linear statistical models a suboptimal prediction tool. An example of a nonlinear parameter would be the tap-to-tap time because it can be broken down into smaller components, charging, melting, refining, extended refining, tapping, and preparation, making it nonlinear. The optimal combination of model parameters was found using a grid search, also known as parameter search, where multiple models are trained for each combination of parameters. Python was used to create the model, and the paper specifies what package was used for each step in the modelling process. [52]

The grid search was performed using multiple input variable batches. The variables were divided into specific variable batches, partly because the number of model types to create using each of the 35 variables would have been very large ($2^{35} \approx 3.44 \times 10^9$) compared to six bundled together variants ($2^6 = 64$). When creating the model, the test data was selected in chronological order with respect to the training data. This is realistic because a statistical model will predict data that is produced after training the model. To compare models with a different number of inputs the adjusted R^2 was used:

$$\overline{R^2} = 1 - (1 - R^2)\frac{n-1}{n-p-1} \tag{2-23}$$

where n stands for the number of data points and p for the number of input variables. [52]

The authors differentiate between two categories for data treatment: domain-specific and statistical outlier detection. In domain-specific outlier detection, data is assessed to its physical possibilities, which in their model creation was to remove trial heats, remove heats with electrical energy above 60 MWh, to define a range for a tap-to-tap time and to remove heats with delays over three hours. Statistical data cleaning was not applied to the data set for three reasons:

- The data set became too small after applying a robust outlier detection.
- Most conventional methods assume that the data is normally distributed.
- It makes sense to not apply outlier detection methods on some variables (e.g. weight percent, as it can vary tremendously).

Only 10% of the original data was removed after all cleaning steps. A total of 36,864 parameter combinations were available. Model-specific parameters were the activation function, learning rate and the hidden node topology, and domain-specific parameters were the input variables and whether the validation set was ordered chronologically or randomised. The validation fraction, the gradient-descent algorithm and the setting “early stopping = True” were kept constant for all models. Each combination of parameters equals one model type, and in addition, each model type was instanced 10 times to assure stability of the parameter selection. [52]

The study applied both an algorithmic and a domain-specific approach. While the domain-specific approach used the different variable batches as described before, the algorithmic approach calculated the pair-wise correlation between the input variables and the output variable using dCor (Distance Correlation), a mathematical expression that can detect both linear and nonlinear relationships. For each subsequent model, the input variable with the lowest correlation value is removed, which leads to a total of 20,160 model types. Both models were analysed using the methods: Kolmogorov-Smirnov (KS) test, dCor and FI. [52]

The grid search showed that validating on chronologically ordered data during training is often not better than validating on randomised data, and 87% of the best models per batch are built of one hidden layer. The authors discussed that the results show that an algorithmic approach for selecting variables can be used; however, it is crucial that the initial selection of the input variables are not random but domain-specific. In addition to that, the model should strive for parsimony, as the different models showed that using all the input variables available resulted in an almost equal performance to the model that used only one-sixth of the input variables. To create state-of-the-art neural networks, it is thus not necessary to increase model complexity through more layers and input nodes. [52]

In another paper, Carlsson et al. [53] created an ANN to predict the electrical energy demand of future heats in an EAF. They also explored two machine learning algorithms to predict the black-box behaviour of the neural network and reveal the influence of input variables. [53]

The neural network was computed using Python. The test set represented 3.24% of the data points, and the training set was split 80/20 into training and validation data. A grid search over one and two hidden layers determined the topology of the neural network, with the number of nodes in each hidden layer being 1 to 24 (48 topologies). The input variables were the total power on time for the heat, the total weight of ingoing scrap, the total volume of propane, the total volume of oxygen through lance, and the total volume of oxygen through burners. Three models were created, one using all variables, the second using all variables but the power on time, and the third using only the power on time. [53]

To analyse the effect of the input variables, two interpretability algorithms were used: Permutation Importance (PI) and Shapley Additive Explanations (SHAP). By comparing the models and looking at the PI values, it was evident that the power on time significantly affects the predictions. The authors noted that this was not surprising since the total power on time is linearly related to the electrical energy consumption. As the power output is similar for most heats, the authors argue that the power on time should not be part of the input variables.

SHAP was in agreement with the PI values with regards to power on time, and five selected test heats were further analysed for the influence of the input variables on the energy consumption prediction. [53]

When interpreted and compared from a metallurgical perspective, three of the five variables were accounted for correctly by the model: power on time, the total weight of the scrap, and the total volume of propane. A higher power on time leads to higher predicted electricity; a higher weight of ingoing scrap leads to higher predicted electricity, and more propane leads to increased energy added by exothermic reactions, which leads to less predicted electricity being used. A model error likely caused the volume of the oxygen through the lance to be wrong, and the volume of oxygen through the burners was superfluous in the model due to high correlation with the volume of propane and should be removed. [53]

Butt et al. [8] performed a study on the usage of different forecasting techniques for STLF, MTLF, and LTLF of electricity grids on an hourly basis. The forecasting methods used were MLPs, LSTMs, and CNNs. The exact configurations for setting up the MLP, CNN and LSTM can be found in Butt et al.'s paper. [8]

The quality of the models was examined using the RMSE, R^2, the MSE and the mean absolute error (MAE) for a forecast of 24 hours, 72 hours, one week and one month. Although the error measures that were calculated showed that, in general, the MLP was the most accurate model, the authors concluded that for STLF (< 1 week), the LSTM and MLP give a good forecast, while for MTLF (< 1 month) the CNN and LSTM have slightly better error measures. They further stated that "it indicates that as data demands are getting higher, the deep learning models with a high number of neurons and optimised activation functions provide better predictions." [8]

To forecast the hourly electricity load of a large Polish steel factory, Klempka and Swiatek [54] presented a neural network. The electrical energy consumption at a specific hour of the day depends on the status of operation, the hour of the beginning and end shift, weekday/weekend/holidays, the hour, the set identification number, and the indicator of exterior illumination activation. [54]

The goal of the neural network was, therefore, to find a relation between the operation status and the electricity consumption from a principal transformer station. The neural networks (one for each of the seven transformer stations) were built the same way: they contained one hidden layer with the number of neurons being the number of inputs, the sigmoidal function was used for activation on all neurons, and the network is trained through the back propagation algorithm. [54]

The results for a forecast on a month’s scale show that in 90% of the cases (provided the plant’s operation was stable), the forecast errors were less than 10%. These errors had two basic reasons: the actual statuses of operation are entered manually and not automatically, and planning errors for a month in advance are almost inevitable due to disturbances in production. [54]

3 EMPIRICAL PART

With the aim of the thesis being to research ways to forecast the power demand of different aggregates of a steel mill, this chapter serves as the description of the approach developed to fulfil this task. During the literature research, it has become apparent that neural networks often found use for prediction scenarios. Due to the internal structure of their nodes, LSTMs have proven to be effective when dealing with time-series forecasting. Therefore, it was decided to put an emphasis on these network types when creating different methods of forecasting.

The primary aggregates of the steel mill Marienhütte besides the EAF are the LF, CC, LHs, and D. Steel scrap is inserted into the EAF to produce crude iron, which is then filled into ladles and put into the LF to perform secondary metallurgical treatments whilst holding a specific temperature. The iron is then transported to the CC where the steel cools down, solidifies and gets cut into billets. The LHs, of which there are five (two vertical, two horizontal, and one so-called "booster fire"), are responsible for transportation and keeping the ladles at a predefined temperature. The emissions of the EAF, LF and LHs flow into the D before exiting the steel mill. While the LF, CC and D use electricity as energy input, LHs only use natural gas, and the EAF uses both. A focus has been laid on the forecast of the EAF's power demand in this thesis due its high share of power demand in the steel plant. Of the steel mill's power demand, the EAF uses 82.8%, which is a much bigger share compared to the LF (2.6%), CC (1.0%), LH (4.0%), and D (9.5%). Due to its small and stable power demand, the CC was not modelled.

In total, the following methods were developed to forecast the power demand of the EAF of the steel mill Marienhütte:

- Perfect Forecast using an LSTM
- Perfect Forecast using an MLP
- Recursive Forecast using an LSTM
- Forecast with Delayed Input using an LSTM
- Multistep Forecast using an LSTM
- Forecast with Phased Input using an LSTM

The Perfect Forecast using an LSTM was also employed on the LF, LH, and D. A description of these methods, the hard- and software used to create them, and the grid search performed for each of them can be found in the following sections. The data used as input for the models were provided by Marienhütte and were cleaned for outliers before being used in the empirical part of this work. Due to data privacy, all data in this thesis will be presented in its scaled version between 0 and 1.

3.1 Hardware and Software

The code for preparing the data, carrying out the grid searches, and creating the final models were primarily run in Amazon Web Services’ Sagemaker Studio Lab and on a computer of the Chair of Energy Network Technology at the University of Leoben. Sagemaker Studio Lab is a cloud computing service with 15 GB of persistent storage and 16 GB of RAM with a runtime of 12 hours for T3.xlarge CPU instances or 4 hours for G4dn.xlarge GPU instances. The in-house computer can make use of an AMD Ryzen 2400G CPU with 47 GB of RAM and an NVIDIA GeForce RTX 3070 Ti graphics card. It should be noted that all the models trained faster when the code was run on a GPU instead of a CPU.

The code was solely written in Python 3.9 using Jupyter notebooks and Spyder as an integrated development environment. Libraries used were Pandas and NumPy for handling data and MatPlotLib and Plotly for displaying results. Scikit-learn was used for the preparation of data and calculating error measures. To further prepare the data and create the neural networks TensorFlow, an end-to-end open-source platform for machine learning, together with Keras, an open-source software library that acts as an interface for TensorFlow, was used. For efficient hyperparameter tuning, applied in the form of a grid search described in section 3.2, the Ray Tune library was imported. Modules from the standard Python library used are the math module, the DateTime module, the time module, the os module and the sys module. An example of the code for the grid search and the final model of the EAF’s Perfect Forecast LSTM can be found in the appendix (chapter 8).

3.2 Grid Search

Hyperparameter tuning is a non-essential part of constructing machine learning algorithms; however, it has the potential to improve the accuracy of the model significantly. It is the process of determining the right combination of hyperparameters to optimise the model performance. [58]

Multiple techniques can be applied to find these performance-maximising hyperparameters, including a grid search, manual search, random search, or a Bayesian search among others. Grid searches are a traditional way to find the best hyperparameters, by going through a manually given set of hyperparameters. [33, 58]

In this thesis, a grid search was performed for every developed method to find the best model architecture. The search space was defined as the number of hidden layers and the number of nodes per hidden layer, meaning the number of nodes in each hidden layer would stay the same. The number of hidden layers was confined to a maximum of three. While using no hidden layer would only make the model capable of representing linear separable functions, Jeff Heaton [59] explains that “two or fewer layers will often suffice with simple data sets. However, with complex datasets involving time series or computer vision, additional layers can be helpful”. Experimental analysis showed that given the data, the models’ accuracy did not increase when using more than three hidden layers. The number of nodes per hidden layer that were searched were 3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100. These numbers of nodes were chosen to be able to cover the number of nodes both between 0 and the number of input parameters (in most cases: ten), as well as the number of nodes far exceeding the number of input parameters. If the maximum number of nodes (100) was reached as the result of a grid search, it was restarted with the search space reduced to the numbers of hidden layers found in the first grid search, but the number of nodes increased (100, 150, 200, …). The nodes in different hidden layers were not varied. The reasons for not optimising all the different hyperparameters that are available are the limitations in computational power when training numerous models and only increasingly small improvements in the final model.

As described above, the Ray Tune package was used for performing the grid searches. Ray Tune allows for integration with many machine learning frameworks, including Keras. It supports distributed training of models, meaning it is possible to train multiple models simultaneously and thus can make training more efficient. Additionally, the user can choose among state-of-the-art algorithms for hyperparameter tuning. [60]

One of these algorithms and the one used for the grid searches in this thesis is the Asynchronous HyperBand scheduler, a trial scheduler. Trial schedulers can terminate bad trials early, pause and clone trials, and alter hyperparameters of a running trial. The Asynchronous HyperBand scheduler, also called ASHA scheduler since it implements Asynchronous Successive Halving, was used as it aggressively terminates low-performing trials and can therefore lead to reasonably good efficiency when performing hyperparameter tuning. [61]

All grid searches were performed with a maximum of 30 but a grace period of one epoch. This means the worst-performing trials were stopped after using the training data set once to adjust the model’s weights, while the best performing trials could calculate through the training set up to 30 times. 30 epochs as the maximum were chosen as experimental approaches showed that improvements in the loss function of the models became increasingly small with more epochs.

3.3 Perfect Forecast using an LSTM

To differentiate between the various models created to forecast the power demand of the EAF, a comprehensive nomenclature is introduced. When talking about a Perfect Forecast LSTM, it is assumed that one has holistic knowledge over the input parameters at the present point of time (or the point of time that one wants to predict), as well as knowledge over a certain number of points of time before that. This is called perfect because it does not reflect reality, especially in the case of an electric steel mill. In reality, it is possible that the input parameters are not measured and transferred into a data processing system in real-time, and even if that were the case, this technique could only make a forecast for the current time step. For this to be a useful forecasting technique, it is necessary to know the exact future course of the input parameters, which is nearly impossible due to the stochastic operational behaviour of the EAF.

The function of such a Perfect Forecasting LSTM model is shown in figure 3-1. Here the window length is 11, meaning the output at t=10 will be based on the input parameters of t=10 and the outputs of the previous ten points in time. As discussed in section 2.1.6, the LSTM is able to communicate information between each LSTM cell, thereby can learn structural dependencies in the time data and, in the end, predict a value for t=10. When training or using test sets, this prediction can then be compared to the label, the "true" value at t=10.

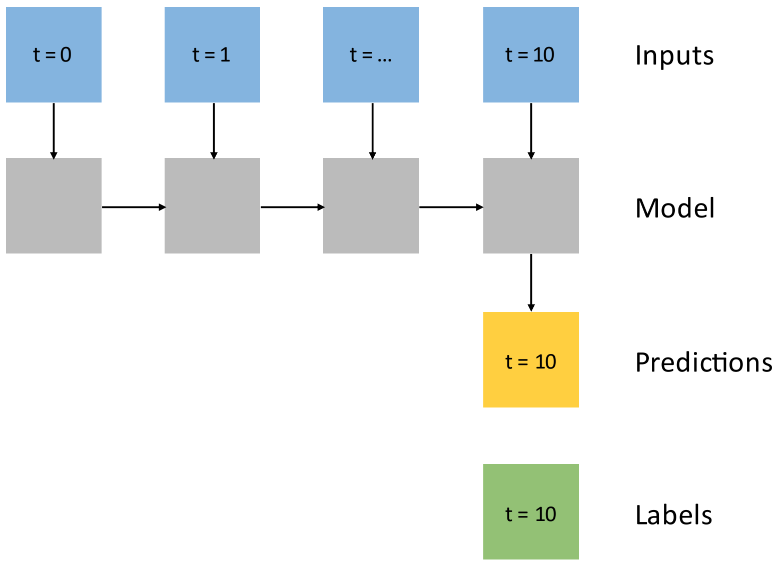

Figure 3-1: Schematic representation of how a Perfect Forecast LSTM operates (own illustration based on [62])

This model was created to predict the power demand, to test the suitability of LSTMs for predicting data of the steel mill, and to serve as a benchmark against other methods used for the power demand forecast of the EAF; however, a Perfect Forecast LSTM was also used to predict the power demand of the LF, LHs and D. The procedure of data preparation is very similar to that of the EAF and will not be discussed in detail here. The parameters used when modelling are presented and the results of the forecasts of these aggregates will be shown in section 4.1.

3.3.1 Data Preparation and Model Construction

During a period between the 30th of April 2021 and the 30th of June 2021, the power demand, scrap amount, melting temperature, coal mass flow, and the oxygen volume flow of the EAF at Marienhütte were measured. In addition, specific states of the EAF were logged (further called phases): preparation, basket 1, basket 2, basket 3, fining, and tapping.

While the scrap amount, melting temperature, coal mass flow, and the oxygen volume flow were provided as continuous floating-point values, the different phases of the EAF were category variables. As the neural network only takes input in the form of integers or floating-point numbers, the different phases were one-hot encoded. Thereby categorical data are processed into numerical values, suitable for machine learning algorithms. This means that for each phase, a new column was generated, that only contained the value “1” at the specific minutes where these phases happened, with the rest of the time being set to “0”. This is the preferred way over simply encoding the phases using specific numbers for each phase, e.g. “1” for preparation, “2” for basket 1, …, “6” for tapping. The model could have trouble learning with data like this, because the absolute difference between 6 and 1 is larger than between 2 and 1, but both of these cases define phases that follow each other.

The output of the model and, therefore, the parameter to be predicted was the power demand, combining both the natural gas and the electrical power demand of the furnace. These two forms of power demand were added because in a later stage of the project DSM_OPT, the share of natural gas and electricity should be optimised, depending on the different commodity prices. In figure 3-2 the used (and scaled) parameters are shown. These parameters remained the same for all the different EAF models created in this thesis (with an exception being the Recursive Forecast, covered in section 3.5) as it was found that using all of the ten input parameters produced the best performing models.

After screening the data, points of time with missing values were deleted. The data was split into a training (80%), validation (10%), and test set (10%) without being randomly shuffled before. This was done to ensure that chopping the data into windows of consecutive samples is still possible and that the validation and test results are more realistic, as they would be evaluated on data collected after the model was trained. The data was then scaled to a value range between 0 and 1 using Scikit-learn’s MinMaxScaler. In this step, it was essential to first fit the scaler only to the training set and then transform the training, validation, and test set data, as this ensures that no information of the validation or the test set is “leaked” to the model during training.

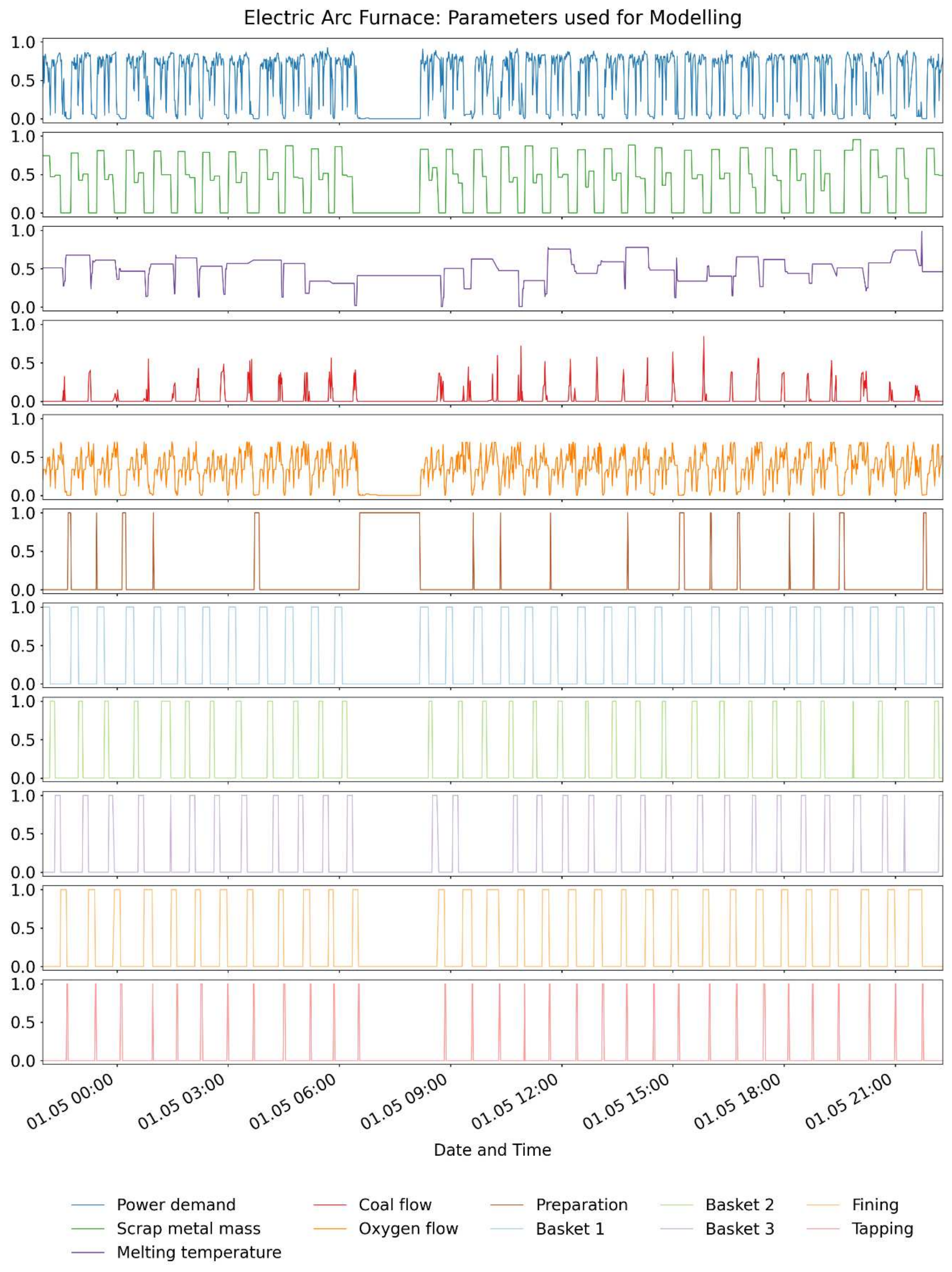

Figure 3-2: Scaled parameters used for modelling the EAF, time period: 1 day (own illustration)

Next, the data was brought into a special format via Keras' class of TimeSeriesGenerator to be used as input for the model. The TimeSeriesGenerator class prepares data in a way that more than one time step can be used as input. This is possibly through the use of the hyperparameters "length" and "batch_size". The length, also often referred to as the window length, defines how many time steps are used as input for the neural network, while the batch size, which should not be confused with the physical batches of the EAF, defines after how many calculations the model weights are updated during training. For the Perfect Forecast LSTM, the window length was set as 1,440 time steps (equals one day), and the batch size was set to 64. This batch size was found to work well experimentally, with the additional characteristic of being a power of 2, which provides advantages in terms of memory allocations in CPUs and GPUs [63].

The final model was trained with an architecture found through the grid search and consists of one hidden layer with 70 nodes. LSTM nodes already contain multiple activation functions, as discussed in section 2.1.6 and as can be seen in figure 2-10, so no further activation function was introduced in the hidden layers. The output layer contained a linear activation function. The model included three callback functions, which are objects that perform actions during training. The first callback function was "ModelCheckpoint", which ensures that the best model according to the loss metric of the validation data is saved after training each epoch. The second callback was the "EarlyStopping" function with a patience set to 10. This means that if the model's accuracy does not improve for ten epochs, the training will be stopped under the assumption that the model cannot improve anymore. The third and last callback function reduces the learning rate from 10^{-3} to 10^{-4} if, for two epochs, the validation loss does not decrease, to further improve the model's performance. Via the given loss function, the model will minimise the MSE in regards to its training (calculated throughout training) and validation data (recalculated after every epoch). The MSE assigns high penalties to large errors by squaring the prediction errors and is therefore helpful in preventing significant forecasting errors [6]. An optimiser that implements the Adam algorithm was used for compiling and training the model. It is a stochastic descent method, and according to Kingma & Ba [64] it is "computationally efficient, has little memory requirement, is invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters". The maximum number of epochs for model training was set to 100.

By plotting the loss of the validation data at the end of each epoch over the number of epochs, one can see that the loss function converges to a minimum pretty quickly after about 20 epochs and only small gains in terms of accuracy are made in the following 60 epochs (figure 3-3). Finally, it is observable that the model does not even train to 100 epochs but stops training after about 80 epochs due to the EarlyStopping callback.

A Perfect Forecast LSTM model was also trained for the LF. The parameters used for the model were measured in the same time frame as those of the EAF and are shown in figure 3-4.

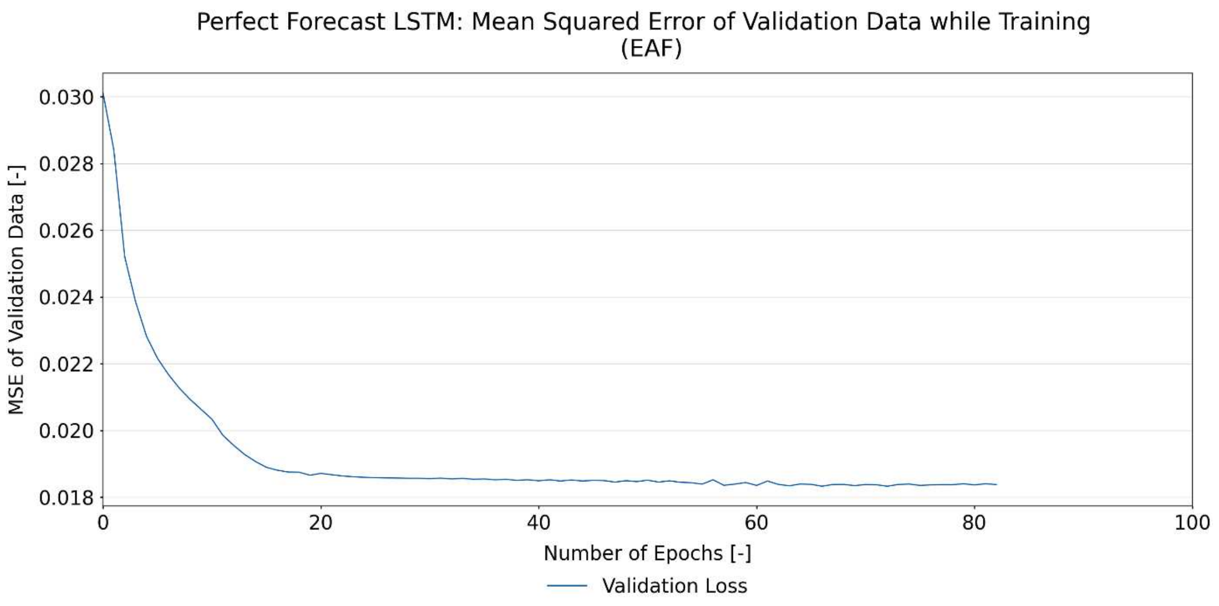

Figure 3-3: MSE over training epochs for the Perfect Forecast LSTM of the EAF (own illustration)

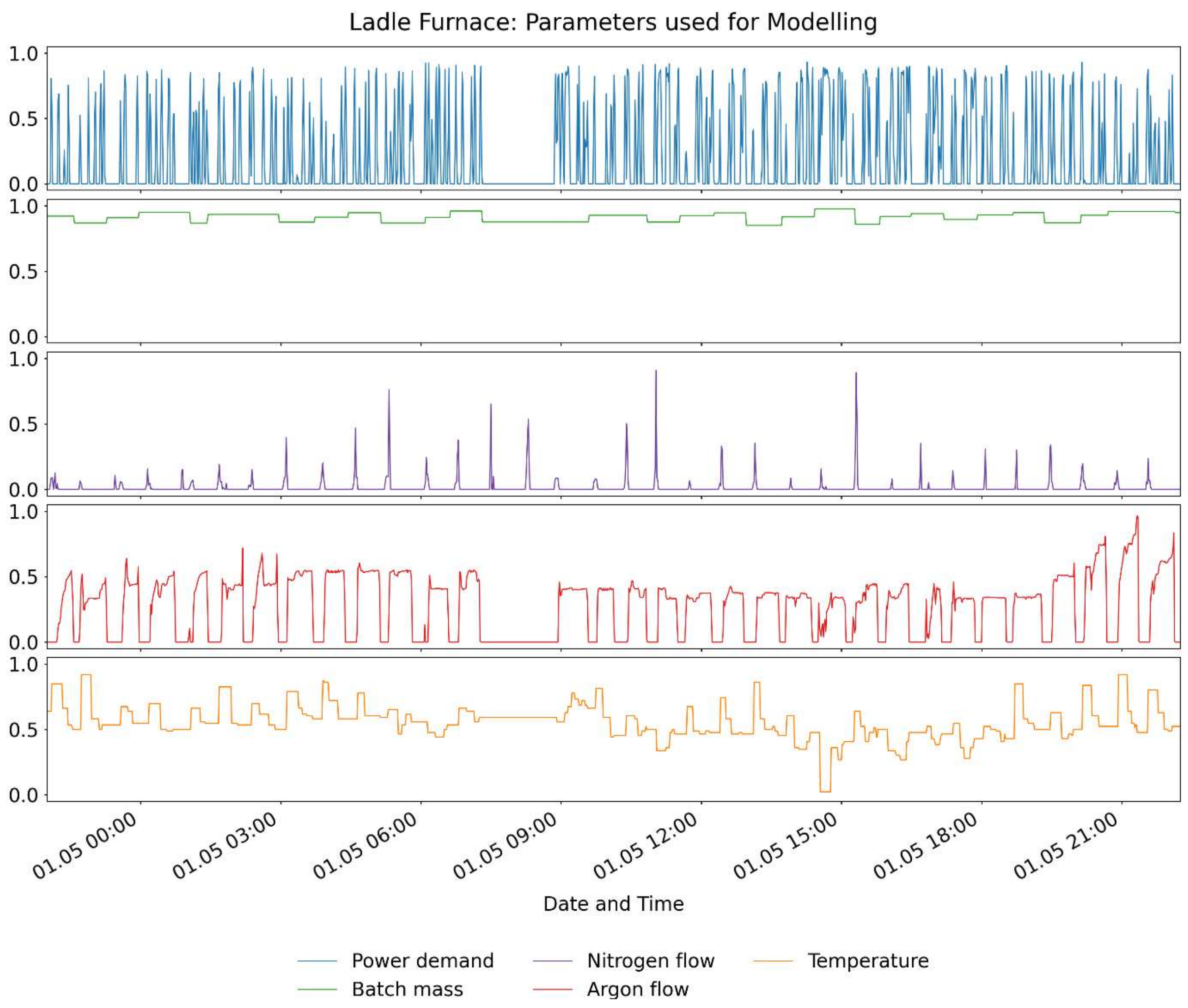

Figure 3-4: Scaled parameters used for modelling the LF (own illustration)

The training of the model took 32 epochs and the course of the MSE over those epochs can be seen in figure 3-5.

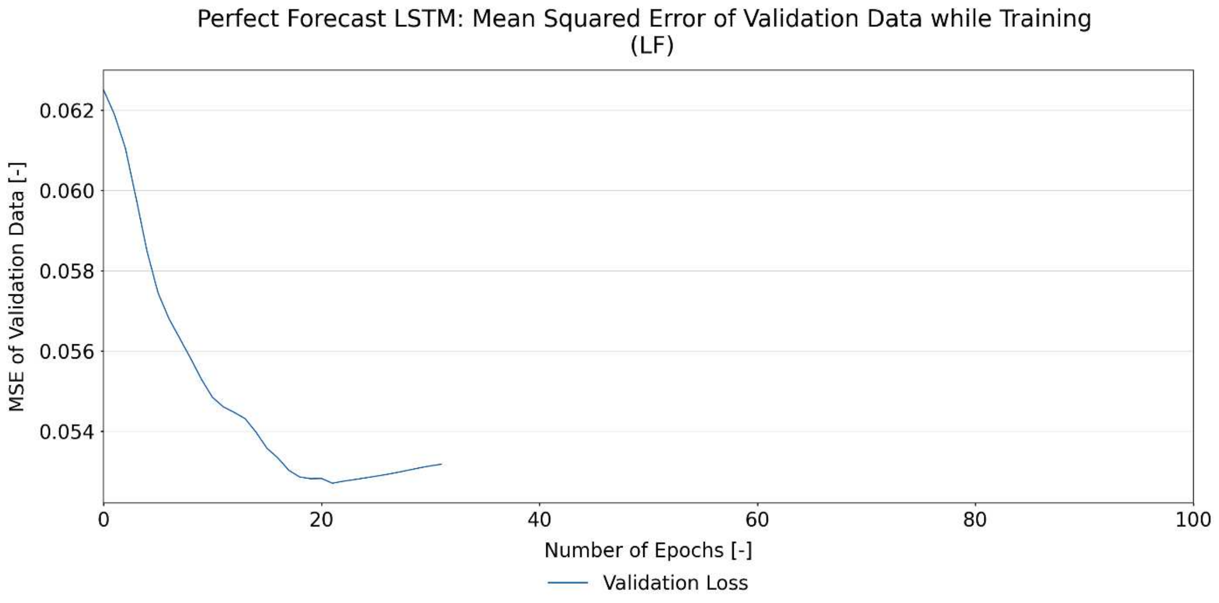

Figure 3-5: MSE over training epochs for the Perfect Forecast LSTM of the LF (own illustration)

The LH system of the Marienhütte consists of five ladle heaters altogether (two vertical, two horizontal and one booster fire). Within the scope of this thesis, one vertical and the booster fire heater were modelled to see if LSTMs can be used to forecast the power demand of these units. The input parameter used for the vertical LH was the current temperature of the LF and is displayed in figure 3-6. Note that for the vertical LH, the time frame has been lengthened from one day to one week to display the course of the parameters better.

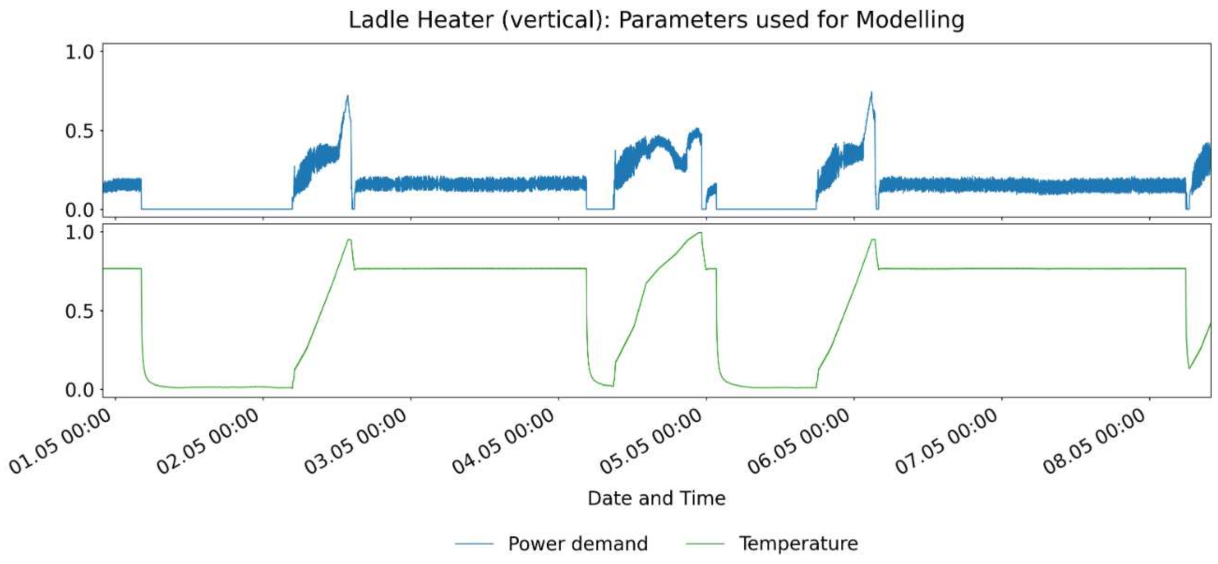

Figure 3-6: Scaled parameters used for modelling the vertical LH (own illustration)

The training over 60 epochs is displayed in figure 3-7.

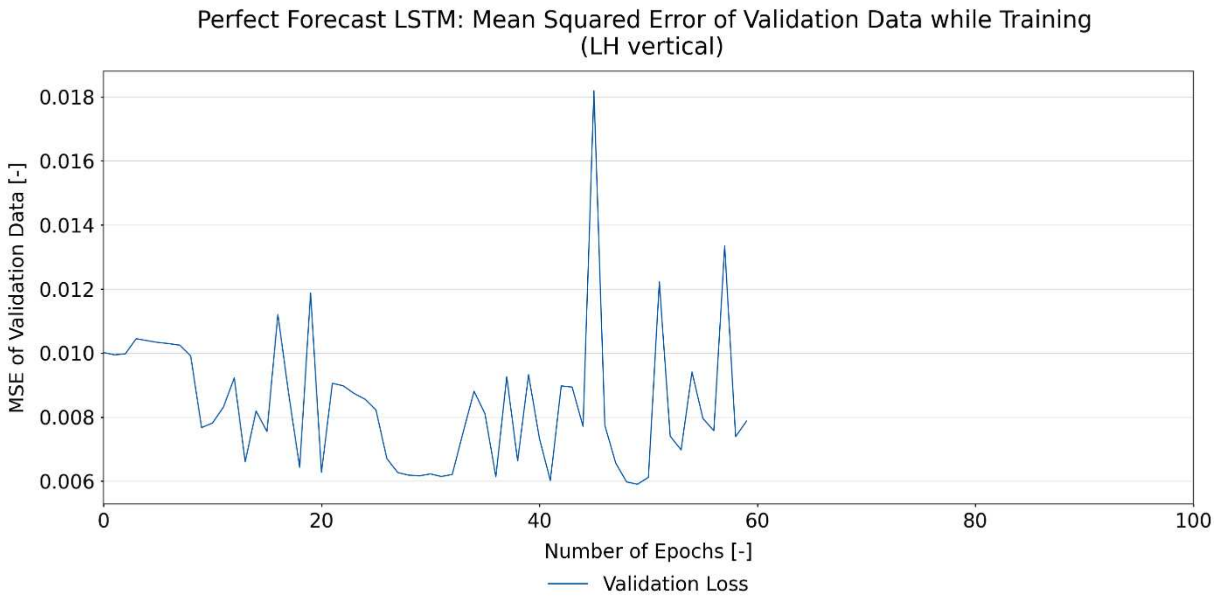

Figure 3-7: MSE over training epochs for the Perfect Forecast LSTM of the vertical LH (own illustration)

The LH (booster fire) took both the oxygen flow and the temperature as inputs, as can be seen in figure 3-8.

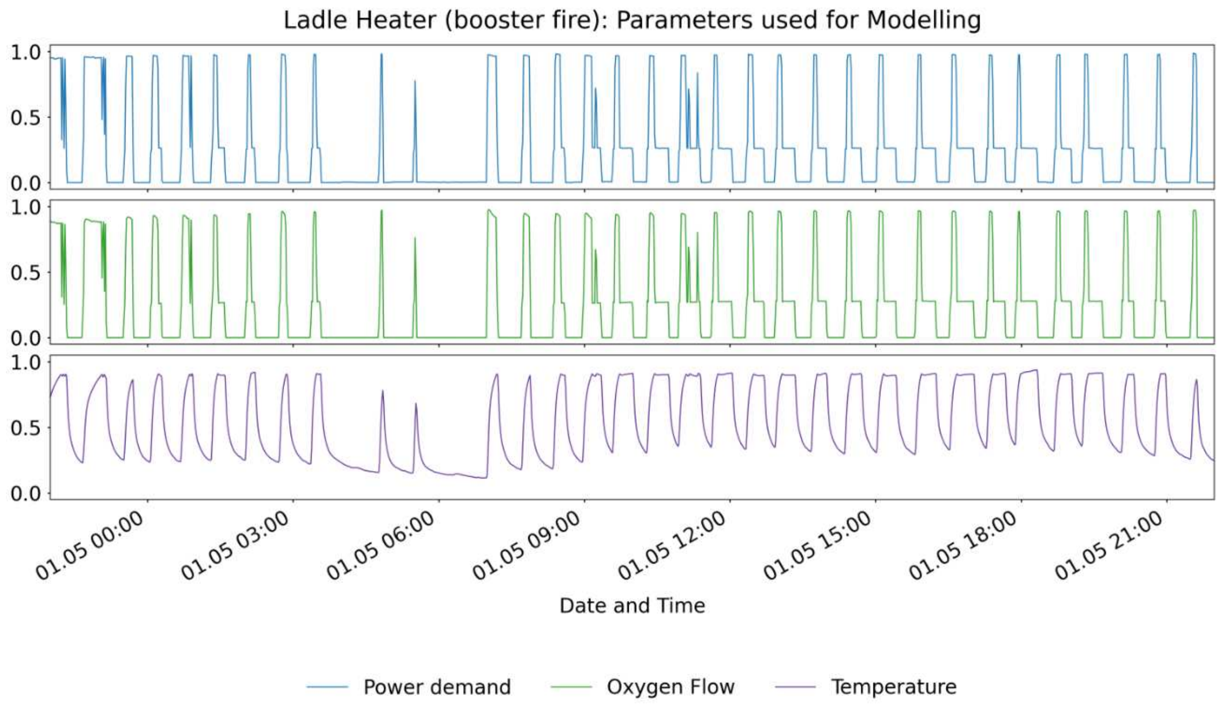

Figure 3-8: Scaled parameters used for modelling the booster fire (own illustration)

The model was trained for 100 epochs, the training progress can be seen in figure 3-9.

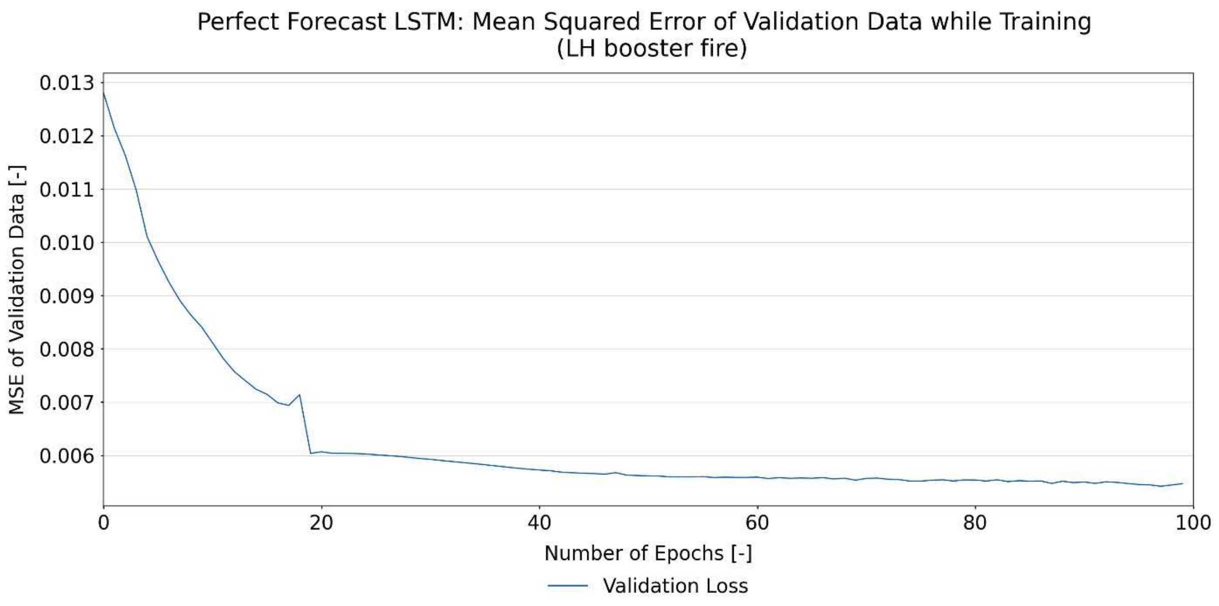

Figure 3-9: MSE over training epochs for the Perfect Forecast LSTM of the booster fire (own illustration)

For creating the model of D, nine different parameters, which are shown in figure 3-11, were used: the power demand, actual pressure, furnace pressure of the EAF, exhaust gas temperature, position of the flap, volume of the clean gas, temperature of the clean gas, dust content of the clean gas, and the external (ambient) temperature. The course of the MSE over the training epochs is shown in figure 3-10.

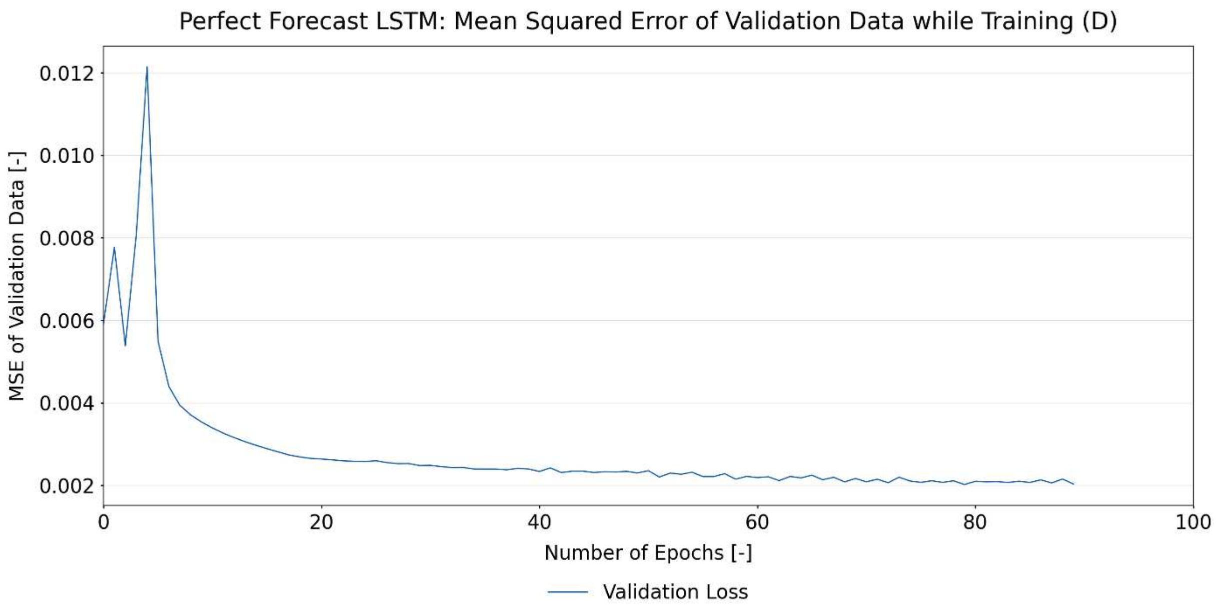

Figure 3-10: MSE over training epochs for the Perfect Forecast LSTM of D (own illustration)

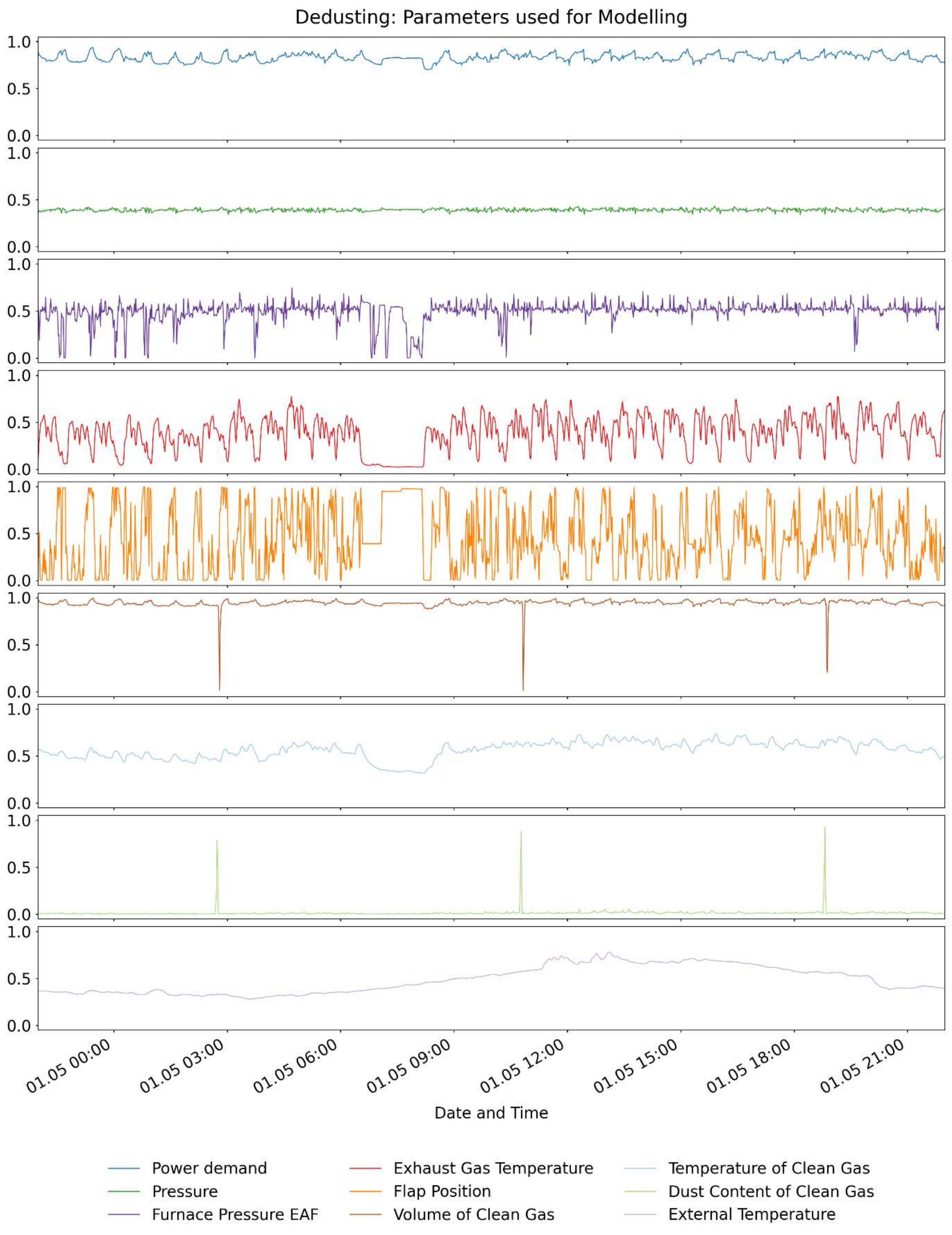

Figure 3-11: Scaled parameters used for modelling D (own illustration)

3.4 Perfect Forecast using an MLP

A Perfect Forecast was also tried using a standard feedforward neural network, subsequently referred to as Perfect Forecast MLP. This was to see if using an LSTM would indeed bring better performance than an MLP when using the presented time-series data from an electric steel mill.

Unlike the Perfect Forecast LSTM, the MLP does not take multiple time steps as input, but only one. The basic function of a Perfect Forecast MLP can be seen in figure 3-12. For each output step, only one point of time of the input parameters is taken as input.

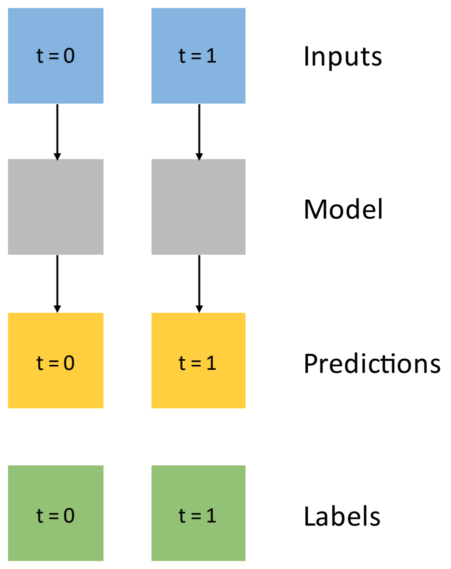

Figure 3-12: Schematic representation of how a Perfect Forecast MLP operates (own illustration based on [62])

3.4.1 Data Preparation and Model Construction

The data was prepared in the same way as for the Perfect Forecast LSTM; however, no TimeSeriesGenerator needed to be applied. The data consisting only of pairs of x- and y-values (input parameters and power demand) for each time step was then fed into the model with the architecture that was found to be best by the grid search. This architecture consisted of three hidden layers with 70 nodes in each hidden layer. The activation function chosen for the hidden layers was the ReLU activation function. A linear activation function was chosen for the output node.

The training progress in figure 3-13 shows that the MSE decreased consistently during a large period of training, especially after about 5 epochs when the learning rate was reduced from 10^{-3} to 10^{-4}. The model was trained for all 100 epochs.

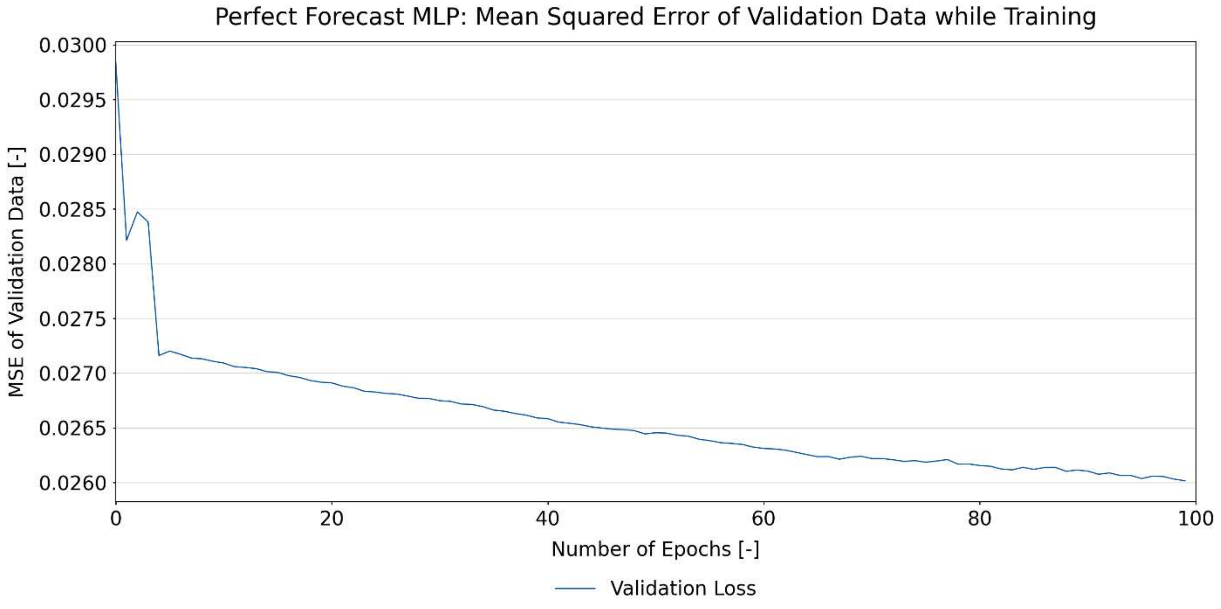

Figure 3-13: MSE over training epochs for the Perfect Forecast MLP (own illustration)

3.5 Recursive Forecast using an LSTM

While these so-called Perfect Forecasts are acceptable for same-minute predictions and explain the difference of basic functions between LSTMs and MLPs, they cannot predict the future power demand in realistic scenarios for the project DSM_OPT, when the values for the input parameters are not known because of process-inherent behaviour.

To solve this problem, multiple forecasting methods were developed; the first one is named Recursive Forecast, which has its name due to the way it forecasts: after the model is trained, it will predict output values and then take these outputs as input values for the prediction of the next time step. A prerequisite for this is that the model can only take its outputs as inputs, which is why the only parameter used in this model for both input and output is the power demand. The principle of the recursive function can be seen in figure 3-14: the outputs of the previous time steps are used as inputs for the time steps t=11 and t=12, while t=0 and t=1 would get dropped, respectively, as the window length stays the same.

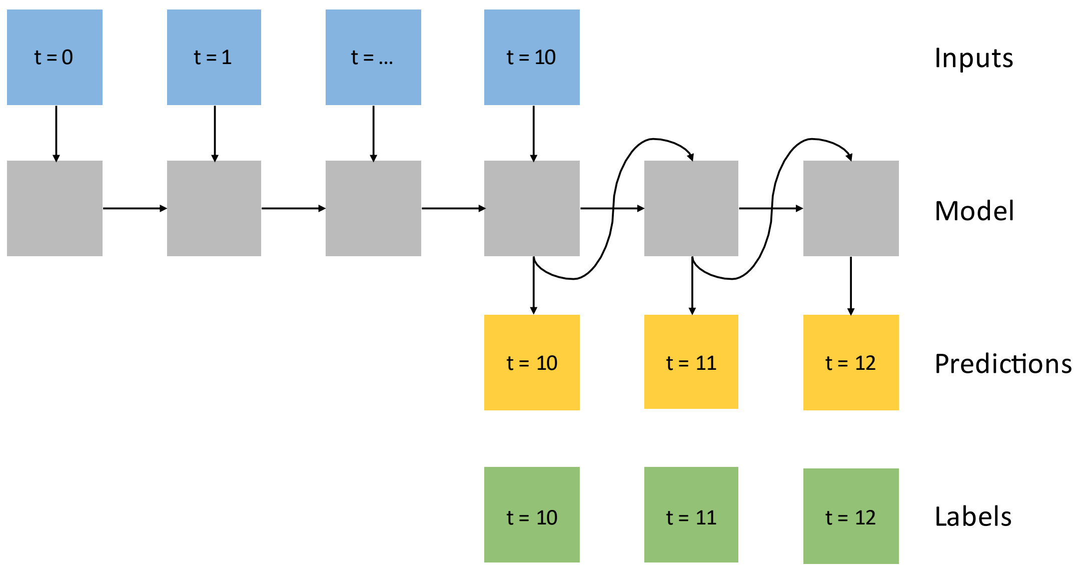

Figure 3-14: Schematic representation of how a Recursive Forecast operates (own illustration based on [62])

3.5.1 Data Preparation and Model Construction

The only parameter used for this type of forecast was the power demand, both as input and as output. Additionally, the data was cleaned regarding the idle states of production. Longer and sometimes unusual periods of time where the production was down (idle states and Sundays) were removed to get continuous batches from which the model could learn. This is more important for this type of neural network than for others, as more emphasis is placed on the time series and less on individual input parameters.

The window length of the model was set to 90 and the batch size set to four. During training, the model would take the previous 90 values and predict the following value for each data point. It therefore learned to predict data points one at a time and not recursively. Based on this model, the final forecast, of which the result is documented in section 4.3, was developed through recursive application of the trained model. The final model had three layers with 90 nodes per layer.

Other settings not described in this section remained the same as for the Perfect Forecast LSTM. The model was able to learn quickly, and it took less than 40 epochs for the training to be stopped due to the loss measure not improving for more than ten epochs (figure 3-15).

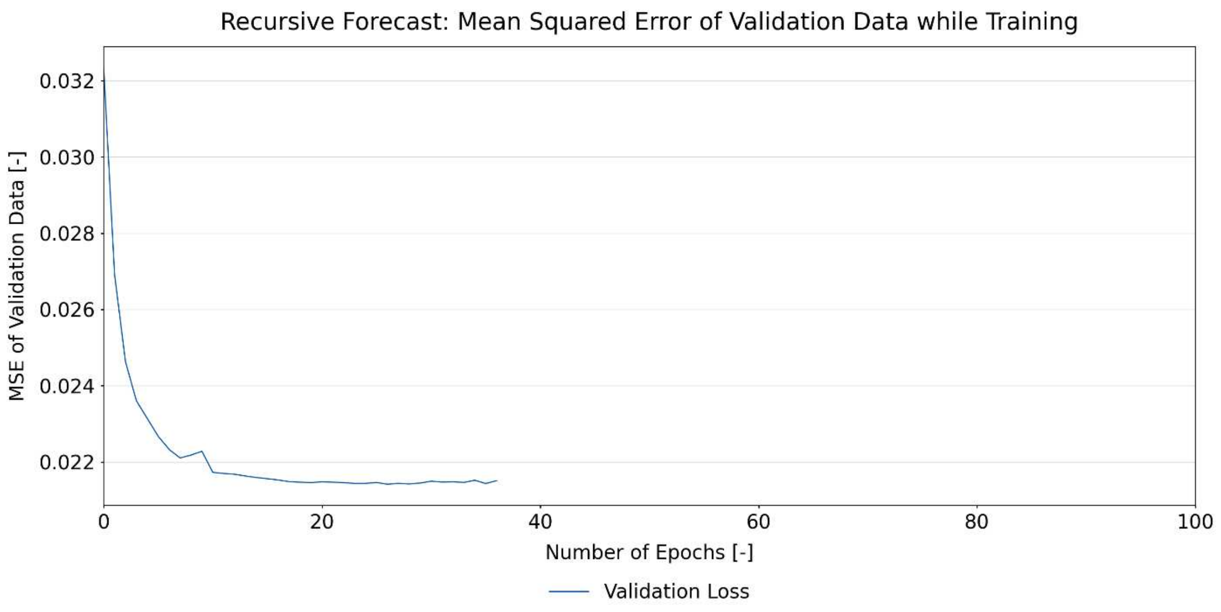

Figure 3-15: MSE over training epochs for the Recursive Forecast LSTM (own illustration)

3.6 Forecast with Delayed Input using an LSTM

Another idea was to predict the power demand based on input parameters further in the past. This means that because the batches in the steel mill follow a similar characteristic, the input parameters of past batches can be used for predicting future batches. The basic operating principle of this type of forecast can be seen in figure 3-16, with a delay of ten time steps. This type of forecast has the advantage that the data measured, e.g. during the previous week (t=0 to t=10) can be used as the model's input at the beginning of the actual week that should be forecasted (starting at t=20).

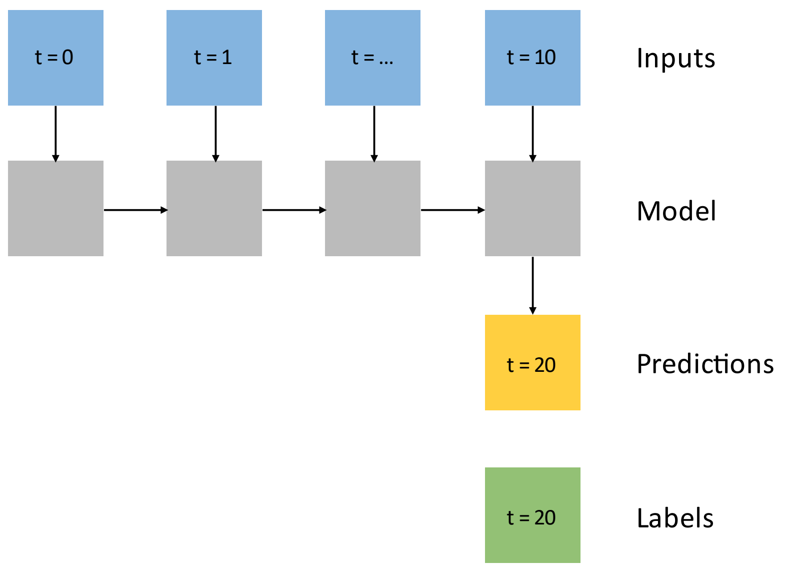

Figure 3-16: Schematic representation of how a forecast with delayed input operates (own illustration based on [62])

3.6.1 Data Preparation Model Construction

For this forecasting technique, two individual models have been created through separate grid searches, one for a time delay of one week and the other for a time delay of the median time of one batch: 41 min. The time delay of one week makes sense to be able to forecast one week into the future; a more minor time shift (roughly one batch) was tried to see if a model would better handle these smaller shifts.

The data used in the section for the Recursive Forecast, which had been cleaned of idle states and Sundays, was further prepared by letting the first value be the start of the production week on the night of the first Sunday to Monday. This ensured the start of a new batch and a new week for the model. The ten input parameters were shifted either 8,971 min of one week or 41 min for the delay of one batch. It has to be noted that the time shift of one week was not 10,080 min but less because the idle states and most of Sunday were deleted. The 8,791 min corresponds to the minutes from the start of the first whole production week to the second production week in the measurement data. The rest of the hyperparameters were set the same way as in the Perfect Forecast LSTM (section 3.3).

The final model of the 1 Week Delayed Input Forecast had hidden layers with three nodes each, while the final model of the 1 Batch Delayed Input Forecast had one hidden layer with 25 nodes. The history of their training is displayed in figure 3-17 and figure 3-18.

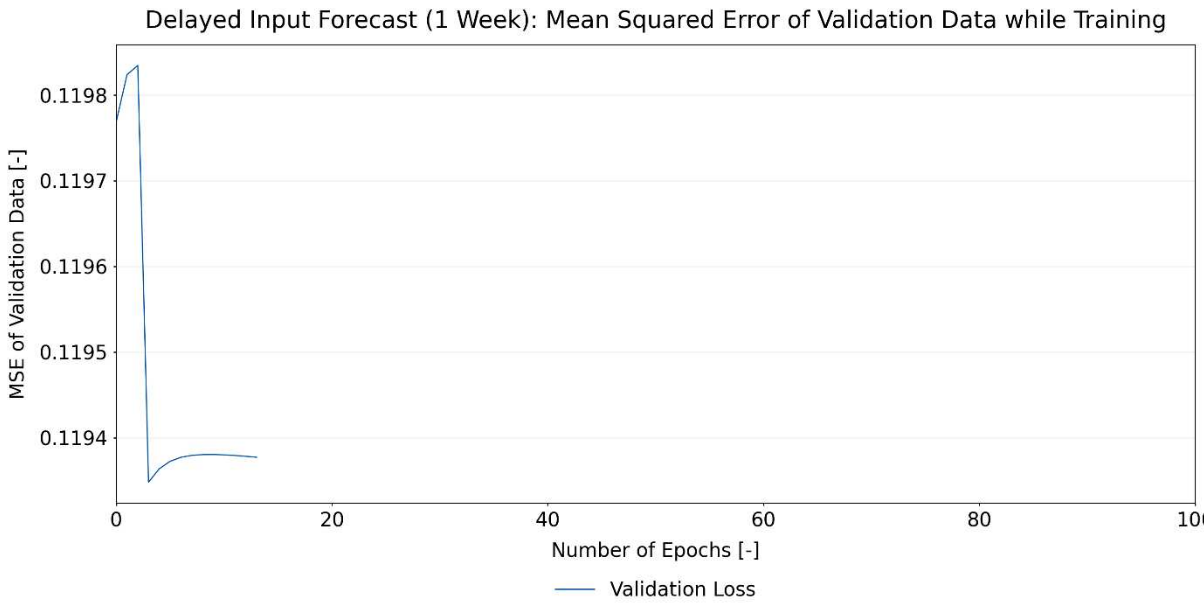

Figure 3-17: MSE over training epochs for the Delayed Input Forecast (1 week) (own illustration)

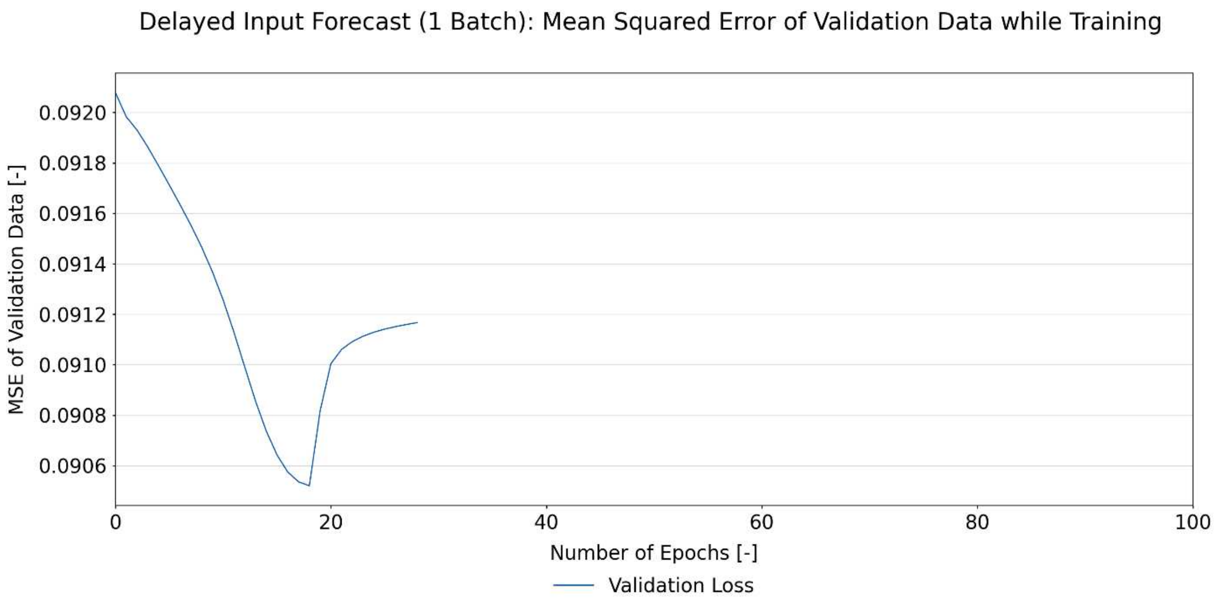

Figure 3-18: MSE over training epochs for the Delayed Input Forecast (1 batch) (own illustration)

As seen in these two similarly looking figures (in both, figure 3-17 and figure 3-18, the MSE drops sharply and rises again afterwards), the models are not trained for more than 20 epochs, as they cannot improve their performance anymore.

3.7 Multistep Forecast using an LSTM

The forecasting models created thus far share one similarity in that they only forecast one time step at a time, as even the Recursive Model is trained to only predict one time step after the other. The aim of this Multistep Forecast is to predict multiple time steps from a given set of inputs. This is also shown in figure 3-19, where the number of outputs is set to three.

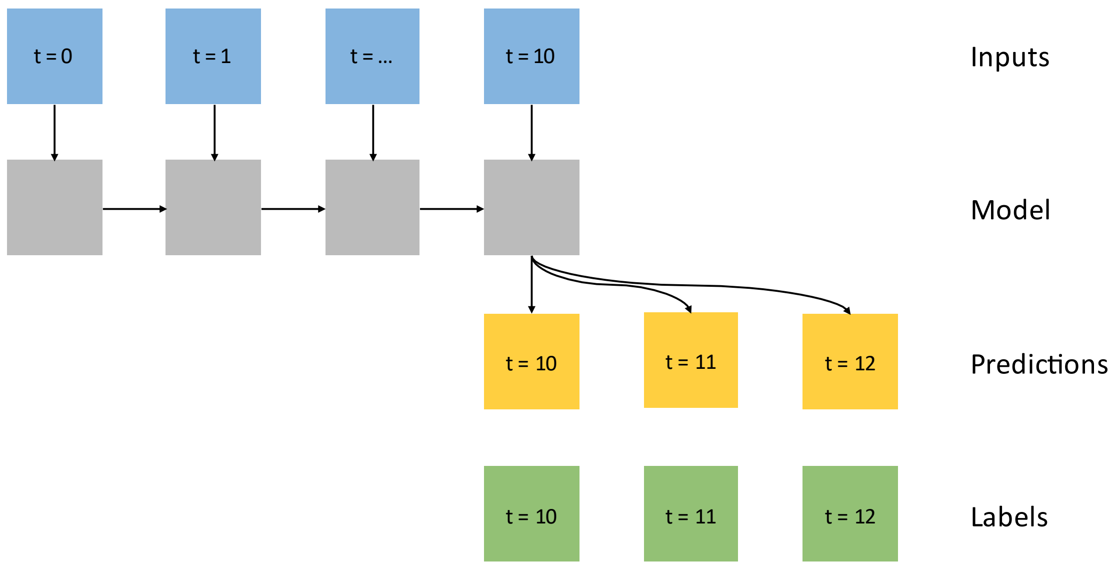

Figure 3-19: Schematic representation of how a Multistep Forecast operates (own illustration based on [62])

The thought behind this type of forecast was similar to that of the Recursive Forecast presented in section 3.5 and the Delayed Input Forecast presented in section 3.6: To predict future values without knowing the input parameters at the point of time that should be predicted. In this case, the problem with predicting one week of output values is that a forecast of 10,080 output values would be extensive given the training data of 60,496 min. Additionally, it can be assumed that a model would not be able to learn this number of outputs, even if the training data was larger.

3.7.1 Data Preparation and Model Construction

The dataset used as input for the model was again the set cleaned from idle states and Sundays. The data was then structured in a way to have 720 input values (window length) paired with the following 41 output values (output length, which corresponds to the median of the batch lengths). The batch size of the model was chosen as two. The rest of the settings for data preparation and setting up the model remained the same as in the Perfect Forecast LSTM. The final model, found to perform best by the grid search, had two hidden layers and 150 nodes per layer. The model was trained in a short amount of time (under 40 out of 100 epochs), as shown in figure 3-20.

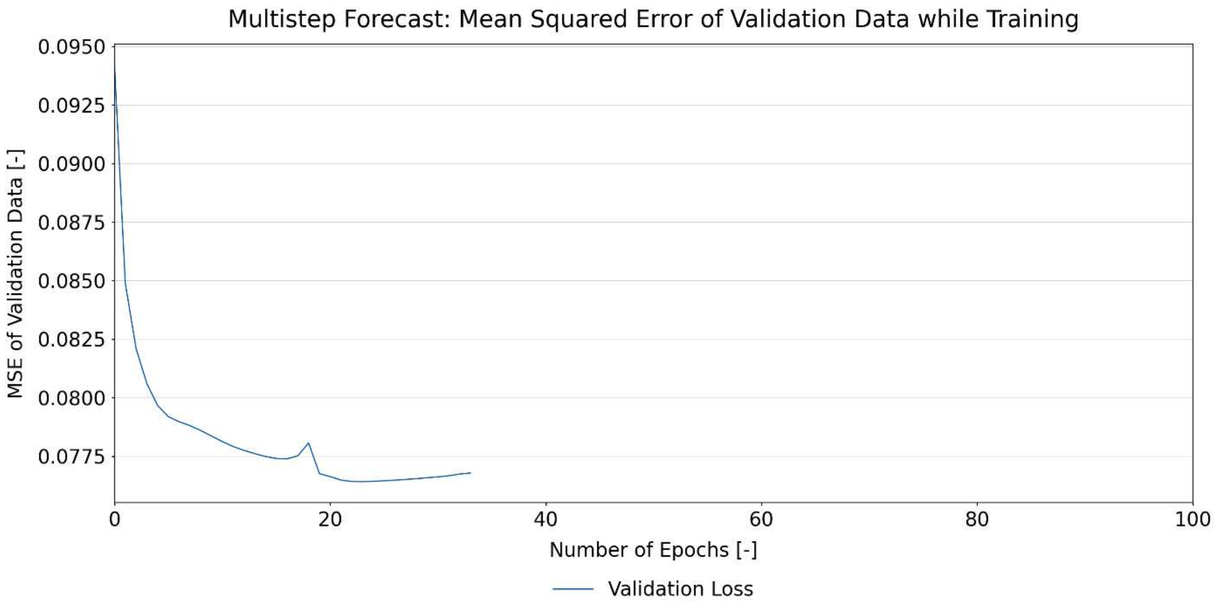

Figure 3-20: MSE over training epochs for the Multistep Forecast (own illustration)

3.8 Forecast with Phased Input using an LSTM

The question arised if it is possible to sample values for the input parameters out of past measurements and create a time series of input parameters that can then predict the continuous and time-resolved power demand. The problem with going about this technique is however that the differences between the values of input parameters are significant between different phases (e.g. there is no coal flow in the preparation phase, but there is a big coal flow in the fining phase) and even in a single phase, the parameters do not remain constant.

However, it was tried to create a model with constant input parameters throughout one phase of a batch. So, while each minute of a single batch's phase used the same input parameters, the LSTM could potentially still learn the time series of the power demand and return the power demand as a continuous time-resolved variable. The main way how this forecasting technique works is shown in figure 3-21. Here t=10 is predicted through the input parameters of t=10 and the ten previous time steps. In contrast to the Perfect Forecast LSTM, presented in section 3.3, the input parameters of the previous time steps are not unique. Rather, they are grouped according to their phases, where the input parameters remain the same for each phase (indicated by the same shade of blue for time steps of the same phase).

A standard feedforward network would not be able to return the power demand in a time-resolved manner; instead as the input parameters only change with each phase, the power demand would also remain the same in-between phase changes.

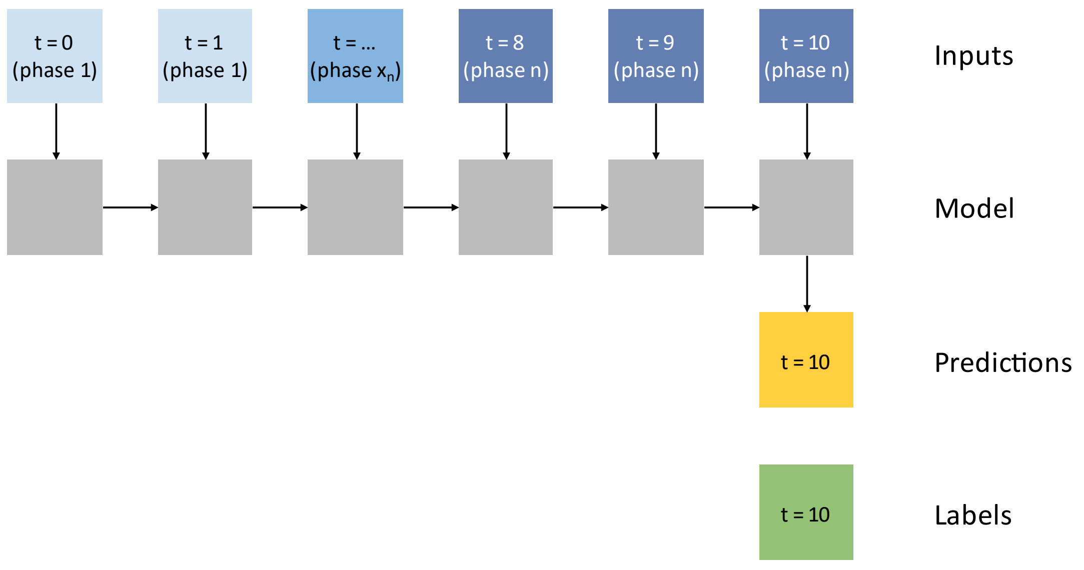

Figure 3-21: Schematic representation of how a Phased Input Forecast operates (own illustration based on [62])

3.8.1 Data Preparation and Model Construction

In order to train this model, the recorded data had to be prepared in a special way: For each phase in every batch, the mean of each input parameter was calculated so that in the end, all ten input parameters stayed the same throughout a phase in one batch. The power demand used to train the model stayed in a time resolution of minutes. This data was then used to train the model.

The creation of a TimeSeriesGenerator class for the data and the model settings stayed the same as section 3.3 except for the number of nodes in the hidden layer, which was set to 90. The course of model training can be seen in figure 3-22.

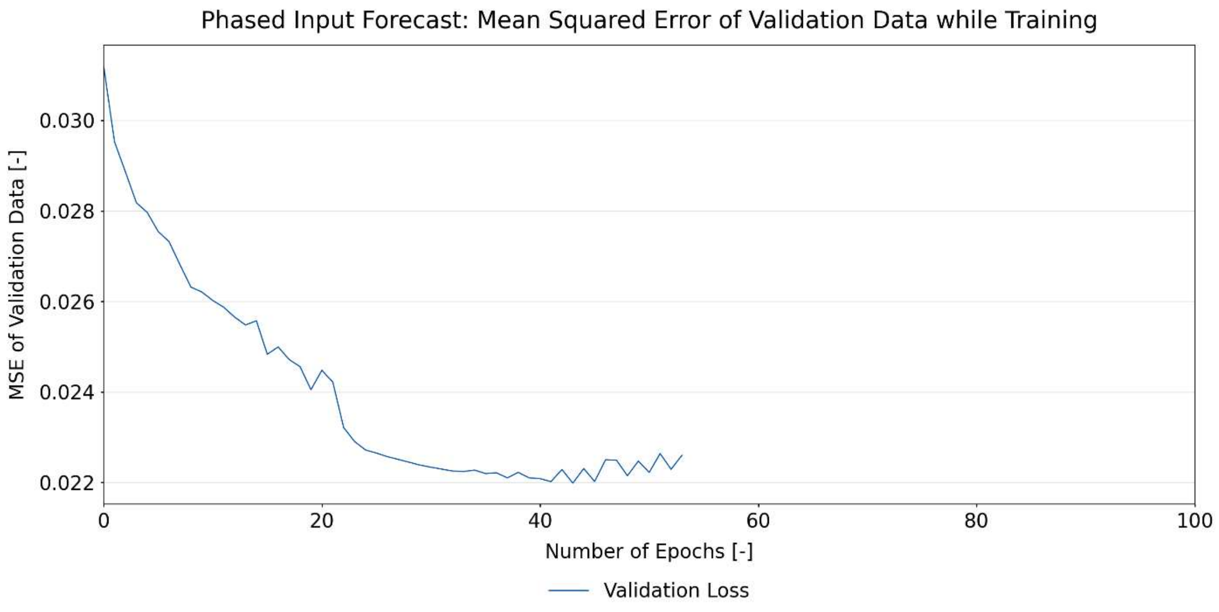

Figure 3-22: MSE over training epochs for the Phased Input Forecast (own illustration)

While this course of action was essential to train the model, it does not help produce forecasts without knowing the actual input parameters in advance. Instead, what is possible with this method is to predict the future power demand via generated input values for each phase. The inputs are generated for each phase and not each minute because the input values are very dependent on the phase of operation, but in a minute resolution they are mostly random.

In a first step, generating the input parameter time series was done by creating a histogram of each input parameter for each phase. These histograms were cleaned of outliers, and due to the underlying physical process, some inputs were set to specified values; for example, the coal flow in the preparation phase was set to 0 as any other value would not make sense because of the process characteristics. In the next step, the duration of the different phases for all the measured batches were calculated, and again histograms with these values were created. Using the created histograms, values were sampled stochastically for the duration of each phase. Each time step and phase was then filled with the other input parameters, also stochastically sampled from the created histograms. This way, a whole time series was generated from the distributions of past input parameters and the duration of phases. To compare the results of the model using this stochastically sampled, generated timeline, another time series was created using only the mean of the input parameters and the mean of the duration of the phases.

4 RESULTS

This chapter covers the results of the neural networks used to forecast the power demand of the EAF. Additionally, the Perfect Forecast LSTM was trained and applied to data of the LF, LHs, and D to predict their power demand. The chapter also discusses the specific models covered, including their training and their performance.

The error measures chosen to compare the different forecasting methods were the RMSE, the RMSE of resampled 15 min values (“15min-RMSE”), and the mean difference. The RMSE is one of the standard ways to measure the error of a model in predicting quantitative data [65] and is often used in time series forecasts in the energy industry [6, 66]. Additionally, it is an error measurement of interest for the models created in this thesis, as the objective was to minimise the MSE during training. The RMSE is defined as the square root of the MSE:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \cdot \sum_{i=0}^{n} (y_i - x_i)^2} \tag{4-1}$$

with n being the number of features in a subset and $(y_i - x_i)$ being the difference between the experimental result and the actual value. This returns the RMSE of the minute-resolved outputs that the model produces. However, it is sometimes the case that the model predicts a batch or a phase one minute earlier or later than in reality. Additionally, for the implementation of forecasting techniques in the DSM_OPT project, it is crucial to be as accurate as possible over a time window of 15 min (electricity is purchased in 15-minute-blocks and the electricity price depends on the maximum power demand of 15 minutes throughout the year); however, not necessarily on a minute basis. This is why for the 15min-RMSE, both the test data and the predicted values were resampled to their mean of 15 min, and the RMSE of these two data series was calculated. Furthermore, to compare the correctness of the forecasts, the mean of both the test data and the predicted values were determined and their difference calculated. This leads to the mean (signed) difference, a measure of central tendency. An overview of these error measures for the forecasting techniques is displayed in table 4-1.

Table 4-1: Overview of different error measures for the forecasting types

Section	Type of Forecast	RMSE	15min-RMSE	Mean Difference
4.1	Perfect Forecast LSTM EAF	0.14	0.07	-1×10^{-3}
4.1	Perfect Forecast LSTM LF	0.25	0.08	-5×10^{-3}
4.1	Perfect Forecast LSTM LH (vertical)	0.05	0.04	0.01
4.1	Perfect Forecast LSTM LH (booster fire)	0.07	0.02	-4×10^{-3}
4.1	Perfect Forecast LSTM D	0.05	0.03	-9×10^{-3}
4.2	Perfect Forecast MLP	0.17	0.07	-0.01
4.3	No Input Forecast	0.15	0.05	-3×10^{-3}
4.3	Recursive Forecast	0.49	0.32	-0.08
4.4	Delayed Input Forecast (1 week)	0.34	0.20	0.01
4.4	Delayed Input Forecast (1 batch)	0.31	0.17	-0.01
4.5	Multistep Forecast	0.28	0.14	-5×10^{-3}
4.6	Phased Input Forecast	0.15	0.07	-0.01
4.6	Phased Input Forecast with Mean Values as Input	0.52	0.36	0.11
4.6	Phased Input Forecast with Stochastically Generated Input	0.45	0.29	-0.01
4.6	Phased Input Forecast with Stochastically Generated Input and aligned batches (5 batches)	0.42	0.24	-0.01
4.6	Phased Input Forecast with Stochastically Generated Input and aligned batches (1 batch)	0.24	0.06	-0.01

For forecasting and comparison purposes, the test data was used. This is best practice, as the test data has not influenced the model in the choice of hyperparameters or during training and is, therefore, “unseen” data by the model. This resembles a realistic scenario for the implementation of these forecasting models.

4.1 Perfect Forecast using an LSTM

4.1.1 Electric Arc Furnace

Predicting the power demand of the EAF using the Perfect Forecast LSTM led to an RMSE of 0.14, with the RMSE of the 15 min resampled values being 0.07. The mean difference of the predictions to the actual values was -1×10^{-3}. The predicted power demand of five batches is compared to the true data in figure 4-1. Five batches were chosen for visualisation purposes and make the diagrams more comprehensible than a plot of all the test data. The error measures were applied to all of the test data nonetheless. This applies to all of the following diagrams.

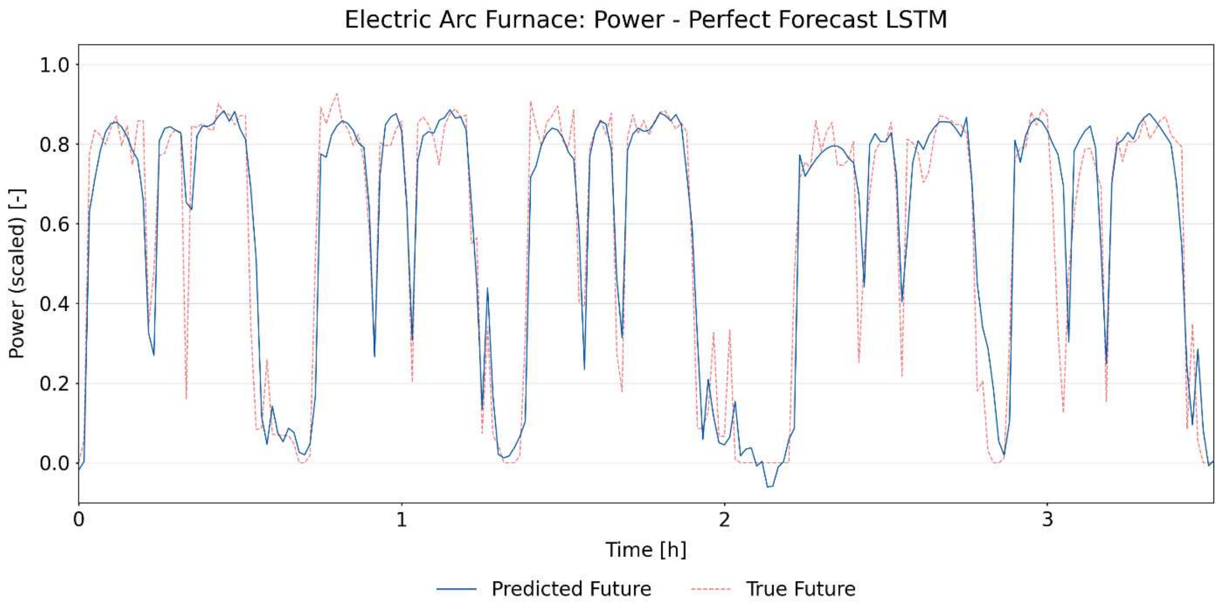

Figure 4-1: Predicted and true power demand of the EAF using a Perfect Forecast LSTM (own illustration)

Comparing the forecast and the test data (figure 4-1), it is observable that the predicted time series is a lot smoother than the true future in that there are no oscillations in the range of the maximum power demand (0.75-1.00). This phenomenon will also be observable when looking at the results of the other forecasting techniques. It could mean that these oscillations are mostly random, the model was not able to learn them and could further lead to the conclusion that by trying not to overfit, the model is still able to generalise well, which results in an RMSE of 0.14. Another option could be that the data that explains these oscillations is not available in any of the measured parameters. The RMSE is higher than the 15min-RMSE because individual time steps are compared: sometimes the “valleys” and “peaks” are not predicted as precisely as they are in reality, and sometimes these valleys and peaks are also time-delayed by a few minutes between the predictions and the true values. This can be seen in the last batch of figure 4-1, for example. These effects can lead to a large RMSE. The 15min-RMSE, on the other hand, is lower, which is easily explained due to the nature of the mean being calculated over the span of 15 min and therefore not having to deal with problems of slightly wrong forecasted “peaks” and “valleys” in the time sequence.

4.1.2 Ladle Furnace

When applying the Perfect Forecast LSTM to the LF, the RMSE is 0.25, the 15min-RMSE is 0.08, and the mean difference is -5×10^{-3}. The time series of the predictions and true values are shown in figure 4-2.

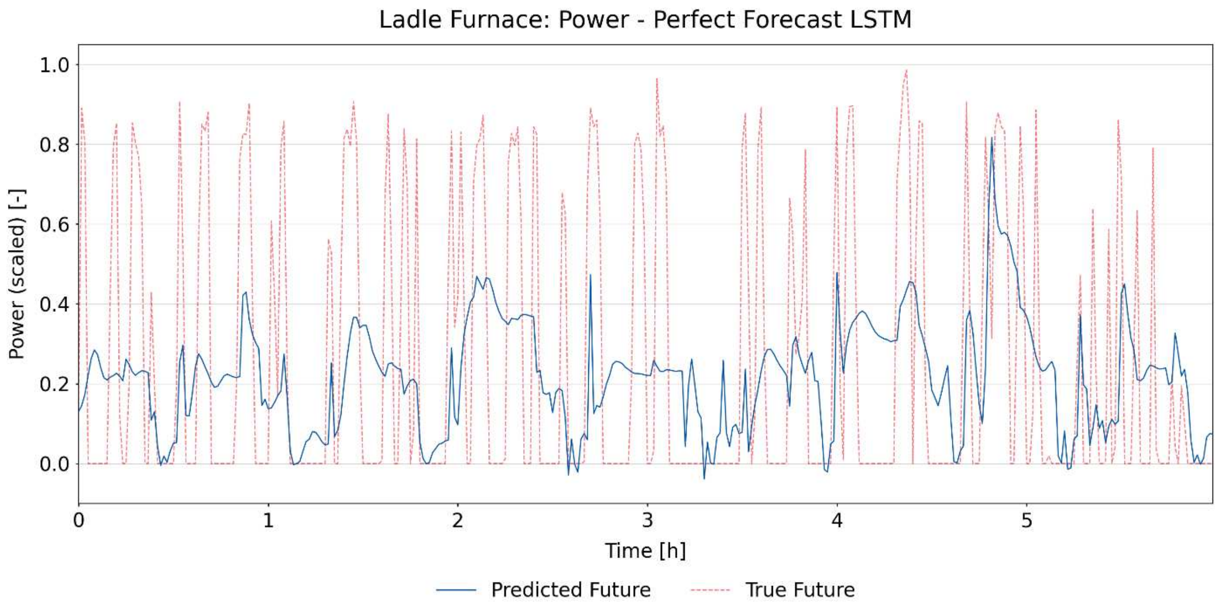

Figure 4-2: Predicted and true power demand of the LF using a Perfect Forecast LSTM (own illustration)

When looking at the different parameters of the LF, there is no apparent correlation between any of the input parameters and the power demand (figure 3-4), instead the power demand seems wholly independent and random from other measurements. This randomness is also visible when looking at the results illustrated in figure 4-2, as the model could not learn any correlations between the inputs and the output or find regularities in the time series. The inability to learn any information about the output from the inputs is also reinforced by the short training time seen in figure 3-5.

4.1.3 Ladle Heaters

The prediction of the power demand of the vertical LH using an LSTM leads to an RMSE of 0.05, a 15min-RMSE of 0.04 and a mean difference of 0.01. The course of the power demands is displayed in figure 4-3.

The model of the vertical LH was only given one input parameter, the temperature (figure 3-6), and one can already see that there is a correlation between the temperature of the heater and its power demand to some extent. The model learned these dependencies successfully, as shown in figure 4-3 and by looking at the error measures. However, the true data shows considerable oscillations, which the model could not reproduce; instead, the model generalised the outputs. This inability to learn the oscillations could also explain the validation loss over the training periods, which did not decrease consistently (figure 3-7). The model may have found the generalised outputs within a few epochs of training but may have had trouble dealing with the oscillations.

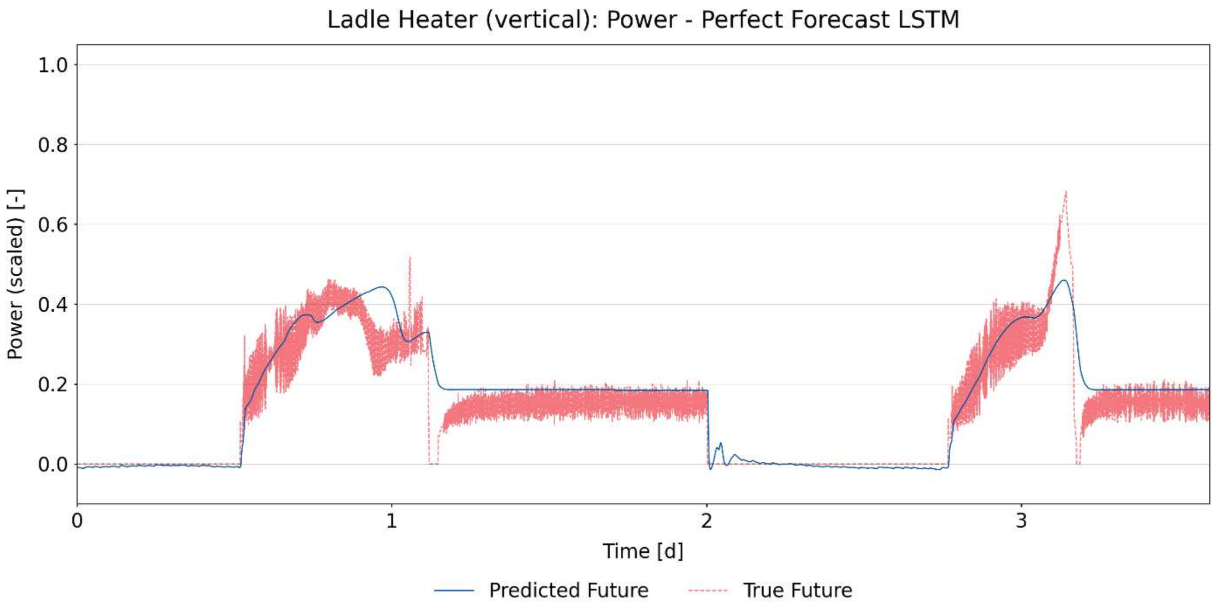

Figure 4-3: Predicted and true power demand of the vertical LH using a Perfect Forecast LSTM (own illustration)

The power demand of the booster fire LH was also predicted using a Perfect Forecast LSTM and resulted in an RMSE of 0.07, a 15min-RMSE of 0.02, and a difference of means of -3×10^{-3}. The prediction, as well as the actual values, are shown in figure 4-4.

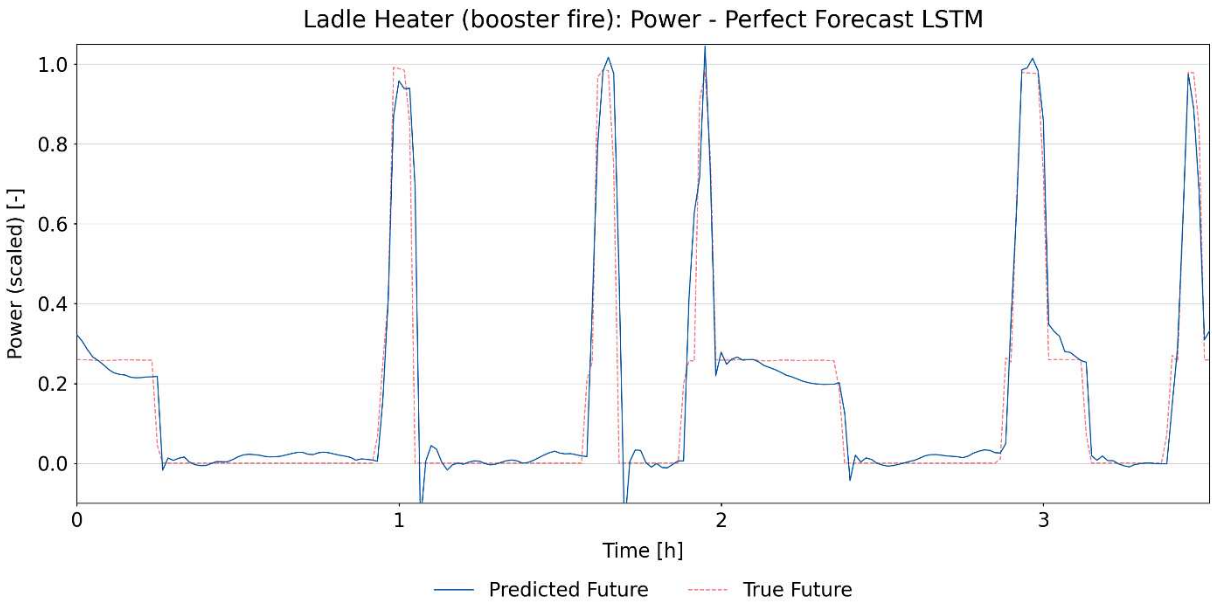

Figure 4-4: Predicted and true power demand of the booster fire LH using a Perfect Forecast LSTM (own illustration)

The booster fire could also be modelled through an LSTM, as can be seen by the results produced in figure 4-4. Like the vertical LH, the temperature input of the booster fire shows a good correlation with the power demand (figure 3-8) but without any oscillations. Therefore, the model was able to consistently learn over 100 epochs and decrease its MSE (figure 3-9).

4.1.4 Dedusting

The course of the predicted and true future of the D are shown in figure 4-5. Quantitatively, the model produced a forecast with an RMSE of 0.05, a 15min-RMSE of 0.03, and a mean difference of -9×10^{-3}.

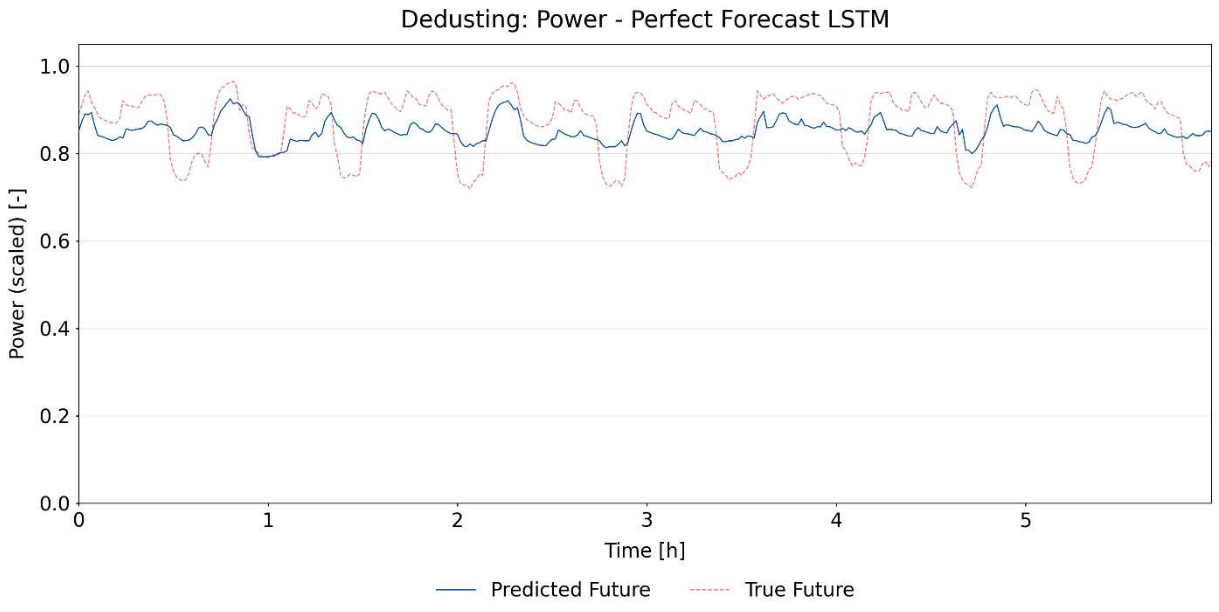

Figure 4-5: Predicted and true power demand of the D using a Perfect Forecast LSTM (own illustration)

When modelling D, multiple input parameters were used (figure 3-11). While the error measures showed promising results, one can see that the time series of the predictions do not accurately align with the actual values (figure 4-5). Instead, both, predictions and actual values, oscillate around almost the same mean with a relatively small amplitude, which leads to the low error measures.

4.2 Perfect Forecast using an MLP

The Perfect Forecast MLP that used a feedforward neural network to predict the output parameter resulted in an RMSE of 0.17, a 15min-RMSE of 0.07 and a mean difference of -0.01. Figure 4-6 shows the time series of the true values and the predictions for the first five batches of the test data.

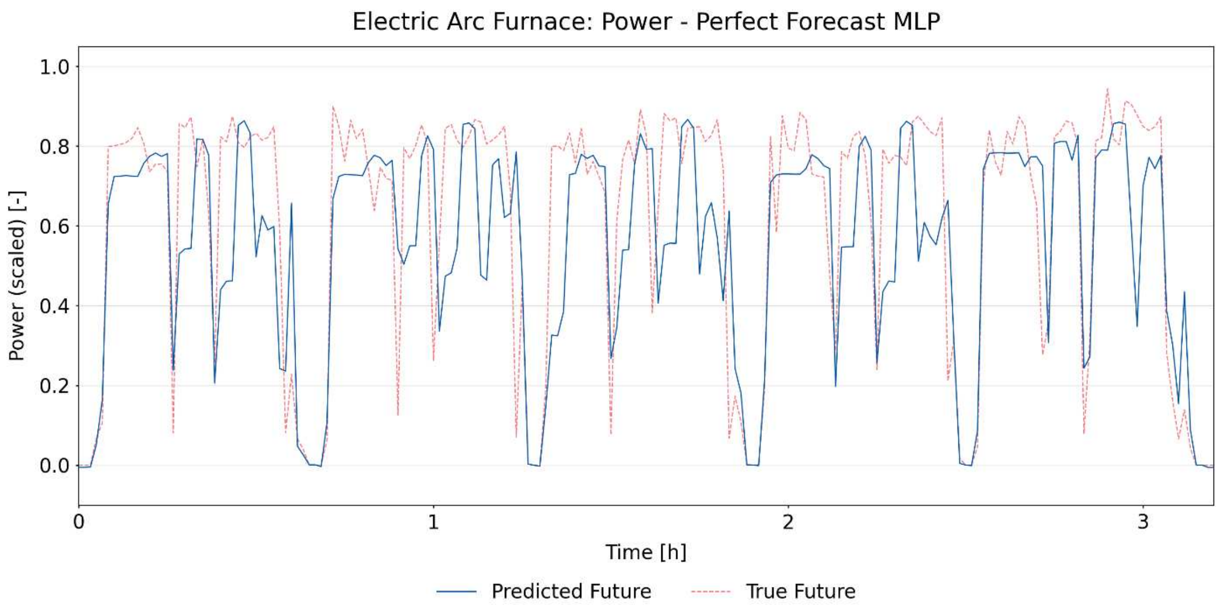

Figure 4-6: Predicted and true power demand of the EAF using a Perfect Forecast MLP (own illustration)

In contrast to the Perfect Forecast LSTM and other LSTM models, the MLPs trained in the grid search and the final Perfect Forecast MLP took less time per epoch to train. However, they need more epochs to finish training, as can be seen in figure 3-13, where the MLP trained for all 100 epochs with the MSE continuously decreasing. Another interesting feature of the final Perfect Forecast MLP was that it had three layers, despite literature stating that two hidden layers are usually enough and the input of the other LSTM models being more complex, but still having less hidden layers. The MLP was created to contrast LSTMs and to ensure that the right subtype of neural network is chosen for predicting the power demand. Comparing figure 4-1 and figure 4-6, it is observable that the MLP often has problems predicting baskets 2 and 3 as there are sometimes “steps” to the maximum range, and the course of the predicted power demand is not as “undisrupted” as in the test data or when using the LSTMs. The quantitative data also shows that the MLP underperforms in contrast to the LSTM: the 15min-RMSE is the same (0.07); however, both the RMSE and the difference of means are higher for the MLP.

4.3 Recursive Forecast using an LSTM

The model that was created during training for the Recursive Forecast LSTM took no input other than the past 90 energy values and predicted the following value. Due to this, the training model created was named No Input Forecast. Its RMSE is 0.15, the 15min-RMSE 0.05, and the mean difference is -3×10^{-3}. Figure 4-7 shows the true and predicted values for the test set.

To talk about the Recursive Forecast, which could theoretically be implemented to predict multiple time steps in advance, it is essential to first look at the trained model, the No Input

Forecast. Despite this model only taking one input parameter, the model architecture was of considerable size with three hidden layers and 90 nodes per hidden layer. The model can predict the power demand of one time step ahead given the past 90 time steps, as can be seen in figure 4-7 and through the quantitative data, even outperforming the Perfect Forecast LSTM in two of the three error measures.

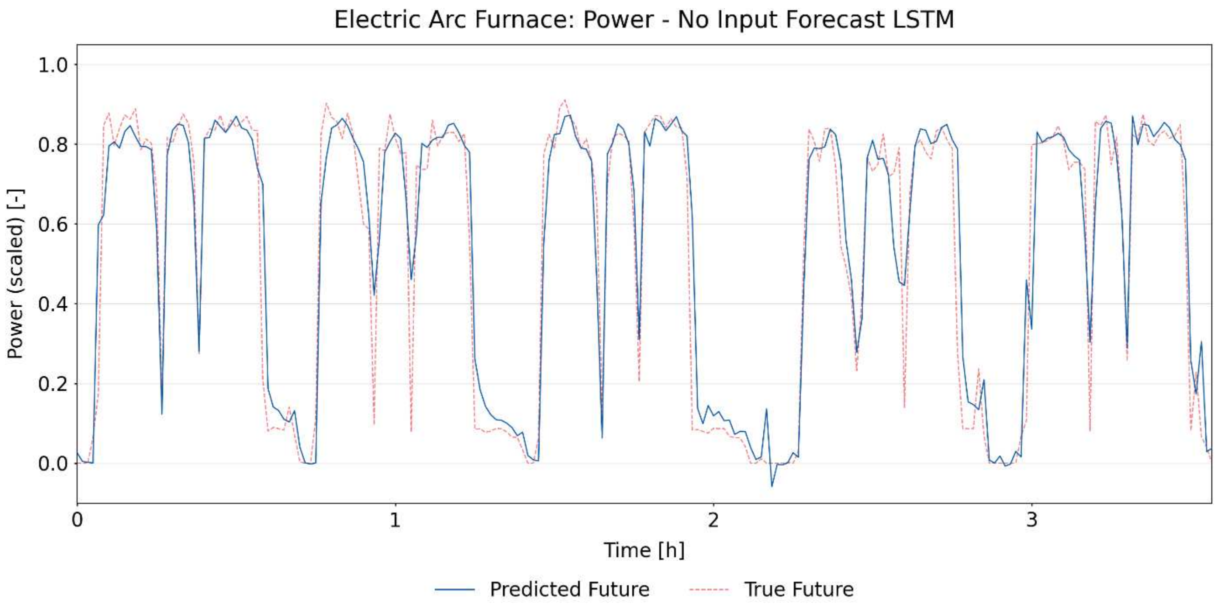

Figure 4-7: Predicted and true power demand of the EAF using an LSTM without input parameters (own illustration)

The Recursive Forecast was produced using the same model. However, using the No Input Forecast model to create a Recursive Forecast, where the model's outputs are used as inputs for the following time steps, does not yield as good results. Its RMSE was 0.49, the 15-min RMSE 0.32 and the mean difference -0.08. The time series of the forecast and the test set are displayed in figure 4-8.

After a short period at the beginning where the predictions are still in sync with the true values, the forecast becomes asynchronous to the actual batches, and the predicted batches start to look similar to each other. Most likely, the predicted values are quite general to avoid overfitting; however, that means after only a few batches an equilibrium is reached, where due to the past always staying the same, the future is always predicted in the same way. This also results in a higher RMSE, 15min-RMSE, and mean difference for the Recursive Forecast. As the machine learning model works on a statistical basis and does not introduce any random elements, it will not change the output, given a fixed set of inputs. Furthermore, what is assumed in this forecasting technique is that the output value at a certain point in time depends on the last 90 time steps (about two batches). In reality, the batches are independent of one another and depend on the process control in part of an automated computer system and to another part of the human operating this computer system. While the underlying assumptions are thus incorrect, the model could learn hidden patterns in the data. However, it was not able to reproduce them recursively.

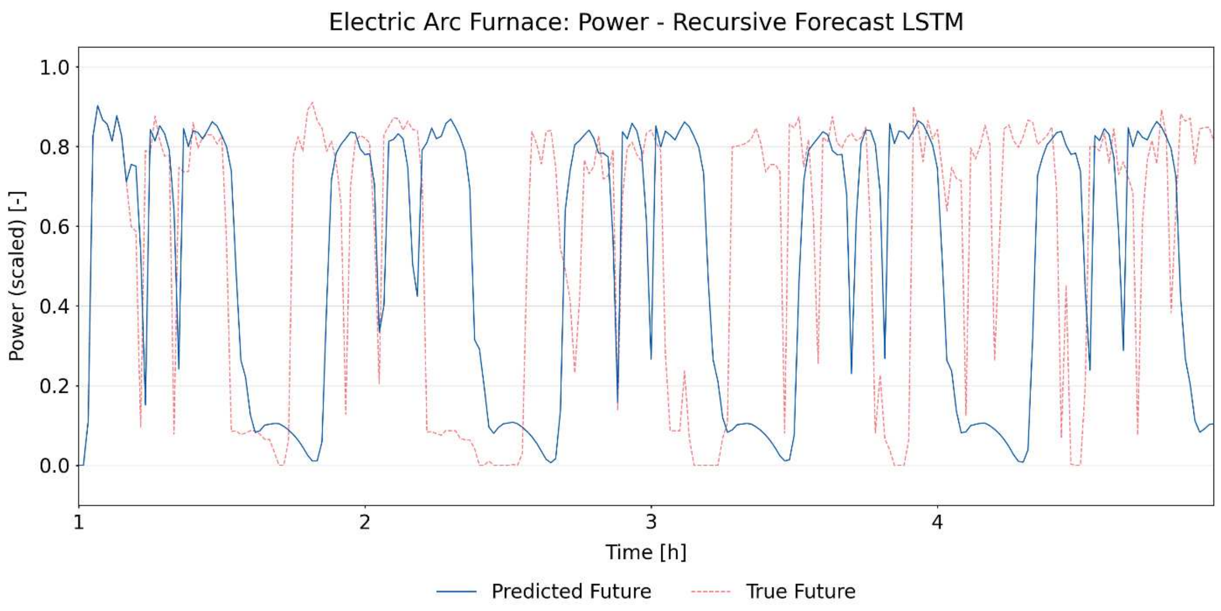

Figure 4-8: Predicted and true power demand of the EAF using a Recursive Forecast (own illustration)

4.4 Forecast with Delayed Input using an LSTM

The Delayed Input Forecast with a 1-week difference between inputs and outputs features an RMSE of 0.34, a 15min-RMSE of 0.20 and a difference of means of 0.01. Figure 4-9 shows the course of its prediction against the values of the test set.

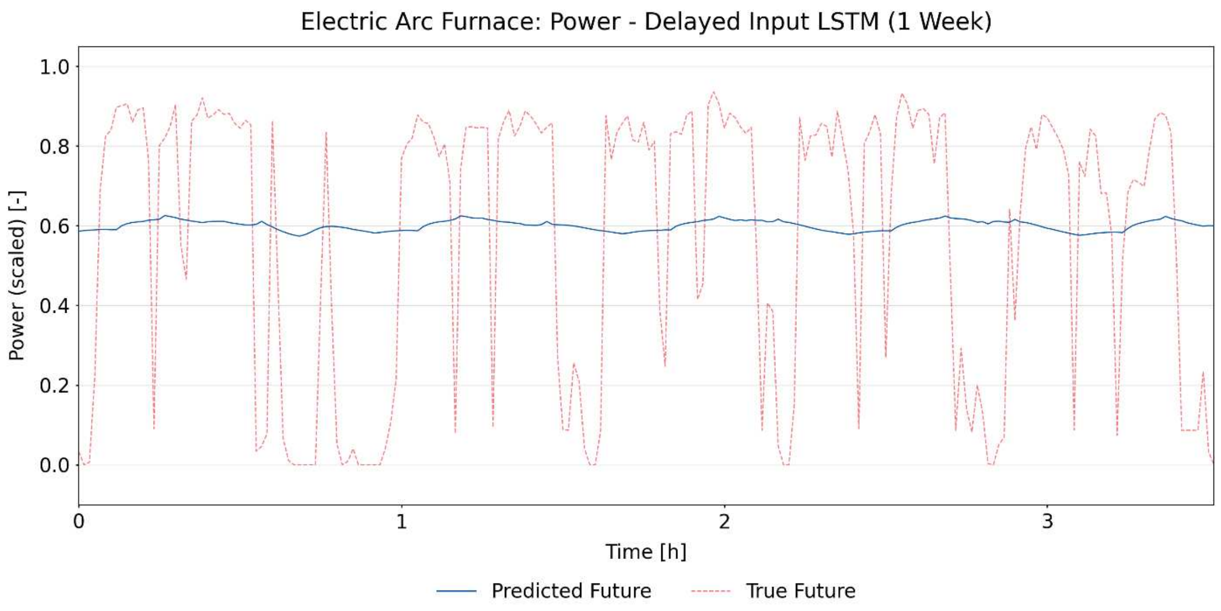

Figure 4-9: Predicted and true power demand of the EAF using a forecast with delayed input (1 week) (own illustration)

The same is shown in figure 4-10 for the delayed input of 41 min or 1 batch. Its RMSE is 0.31, its 15min-RMSE is 0.17, and the difference of means is -0.01.

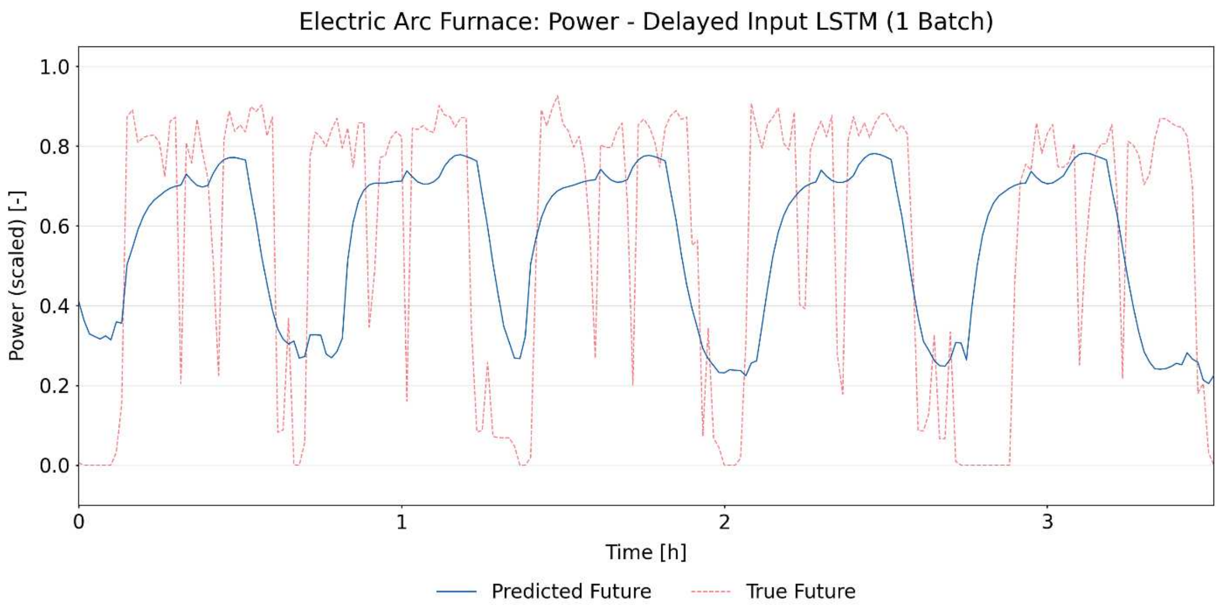

Figure 4-10: Predicted and true power demand of the EAF using a forecast with delayed input (1 batch) (own illustration)

The Delayed Input Forecast aimed to use past inputs to predict future power outputs. When looking at both figure 4-9 and figure 4-10, one cannot recognise a resemblance between the predicted values and the true values; instead, the predicted values seem to be completely independent of the real values. When looking at the training loss over the epochs for both forecasting models (figure 3-17 and figure 3-18), one can see that the training does not take long (less than 20 epochs) and that the MSE at the end remains high in comparison to that of other models. This concludes that the model cannot learn the correlations between the inputs and the outputs. Given that the inputs have been time-shifted 1 week or 41 min in relation to the outputs, the input parameters do not align anymore with the outputs in a timely manner, and therefore, the model cannot learn. Even when only shifting the parameters by 1 batch (41 min), they are technically only shifted by the median duration of the batches, which means that the phases of the inputs and the outputs are never precisely aligned. For example, the preparation phase of the input parameters might be shifted and aligned to an output of the first basket, which is very different in terms of power demand. The inability of the models to learn is also reflected in their error measures, which are significantly worse than those of the Perfect Forecast LSTM.

4.5 Multistep Forecast using an LSTM

The Multistep Forecast produced predictions with the following error results: an RMSE of 0.28, a 15min-RMSE of 0.14 and a mean difference of -5×10^{-3}. The time series of the predictions, as well as the test data, is presented in figure 4-11.

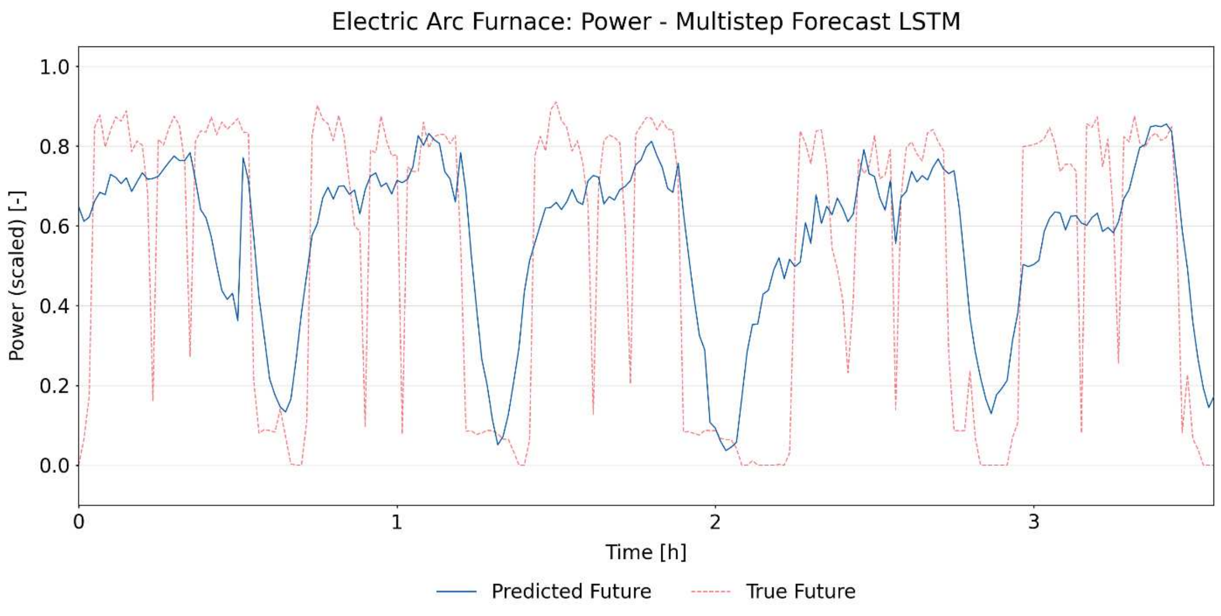

Figure 4-11: Predicted and true power demand of the EAF using a Multistep Forecast (own illustration)

The sequence of the predicted input of the Multistep Forecast (figure 4-11) shows similarities to that of the Delayed Input Forecast in that the model apparently cannot predict the individual phases of the batch. Despite this, the length of the batch is often predicted correctly, as can be seen in the sharp declines in power demand. The course of training, depicted in figure 3-20, shows that the model learns for less than 40 epochs in an MSE range significantly higher than that of other models. Thus, the model could not learn the correlation between the 41 output values and the inputs. A reason for this could be that the 41 min are very irregular (always a different combination of phases) and do not provide a solid basis of data for the model to learn dependencies to the previous time steps of the input. While these error measures are not as high as those of the other models that did not learn, like the Delayed Input Forecast or the Recursive Forecast, they are still significantly higher than those of the Perfect Forecast LSTM.

4.6 Forecast with Phased Input using an LSTM

Using phased input instead of time-resolved input returns predictions with an RMSE of 0.15, a 15min-RMSE of 0.07, and a mean difference of -0.01. Figure 4-12 shows the forecast.

Looking at the model that was trained using the phased input, one can see (figure 4-12) that the predictions match the actual values, meaning that even though the model takes the same inputs for each time step in a phase, the LSTM nodes can learn the sequences in the data and vary the outputs according to the course of the batches. Quantitatively, the results of the Phased Input Forecast are in the same range as those of the Perfect Forecast LSTM. They show that only varying the inputs for each phase, and not minute-wise, is also a viable option for creating minute-resolved power demand forecasts. The only problem with the implementation of this method is, again, that the input parameters are not known in advance for the following week or the future in general, even if it is the nearest future.

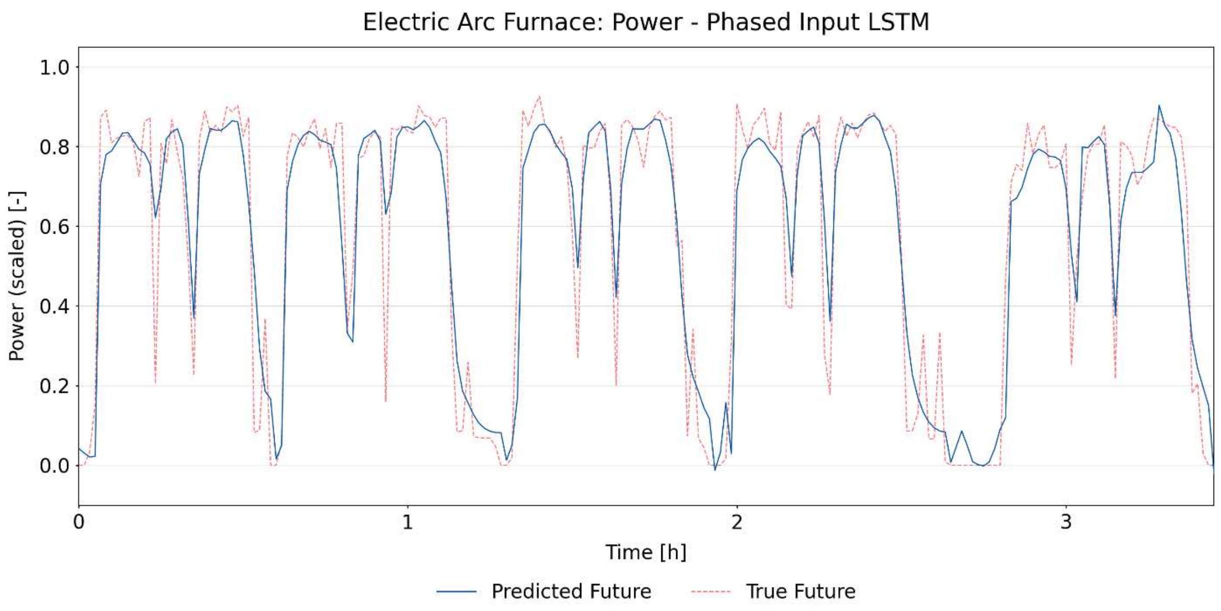

Figure 4-12: Predicted and true power demand of the EAF using a Phased Input Forecast (own illustration)

When using the mean values of the input parameters of each batch as input for the model the RMSE increases to 0.52, the 15min-RMSE to 0.36 and the mean difference to 0.11. Figure 4-13 shows the predicted and actual time series.

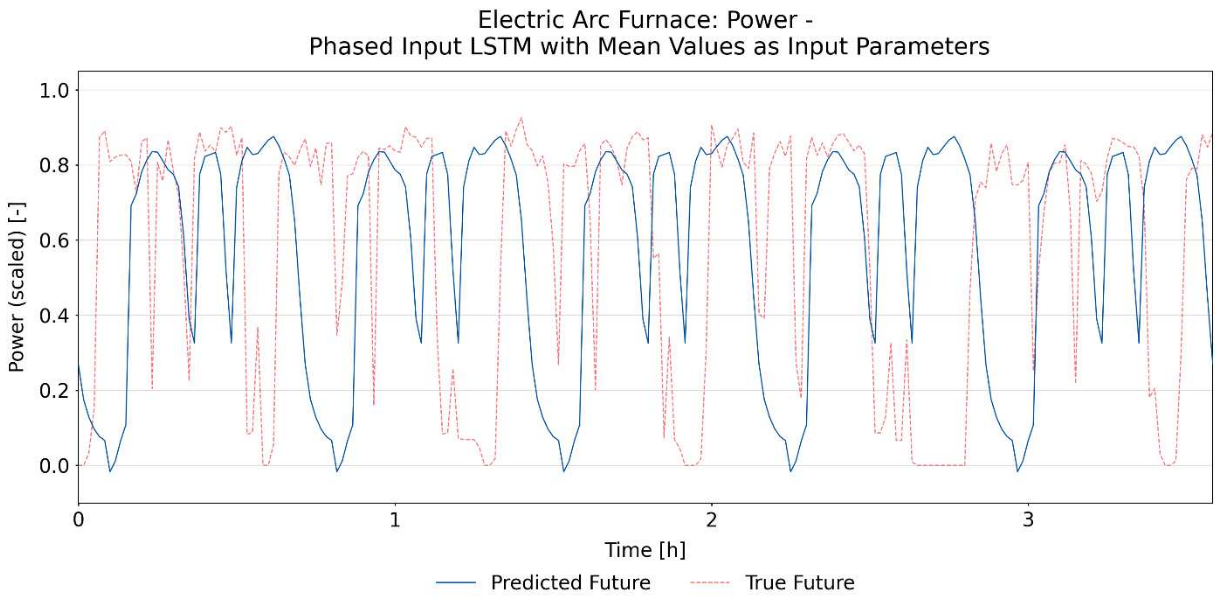

Figure 4-13: Predicted and true power demand of the EAF using a Phased Input Forecast with mean values as input parameters (own illustration)

Since the power demand can be predicted using phased input, the model used the inputs of the mean of all past phases, as well as their mean durations, for generating a forecast. The time series of the forecasts using this generated input shown in figure 4-13 has two apparent aberrations: the predictions are repetitive for each forecasted batch, and they are out of sync with the true values. The first phenomenon can easily be explained by the fact that the model inputs for a batch always remain the same for the whole time series. The second phenomenon, the predicted and true batches being out of sync, happens because the durations of the batches in real time do not always stay the same as they do for the input parameters. The durations of the forecasted and true phases are thus misaligned. The reason for the higher mean difference can only be explained by the fact that while the means of the phases were taken as input for the model, they do not generate the mean of outputs but are skewed towards producing output values that are in general higher than the true values.

In a next step, probability distributions were created for each input parameter of each phase. The parameters for each phase were then sampled from these distributions, creating an artificial time series of input parameters. The forecast created using these input parameters can be seen in figure 4-14, with a similar characteristic as the previously described forecast using mean values: the batches of the forecasts and the test data asynchronous. While the stochastically generated input forecast resulted in better error measures (RMSE: 0.45, 15min-RMSE: 0.29, mean difference: -0.01), it was still far off the benchmark model (Perfect Forecast LSTM).

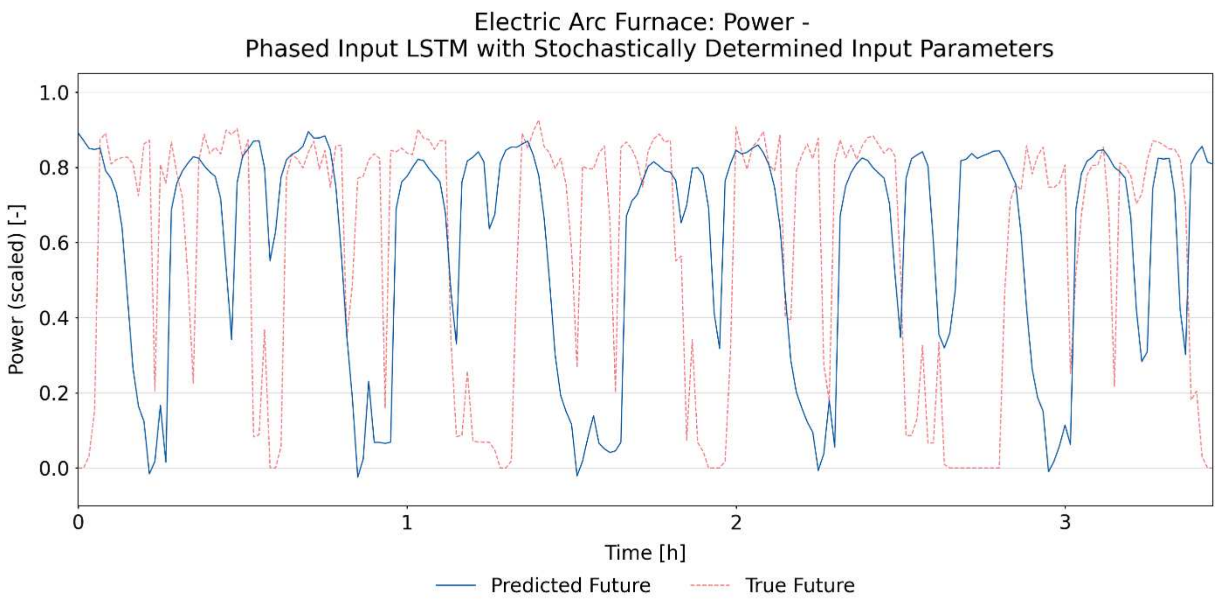

Figure 4-14: Predicted and true power demand of the EAF using a Phased Input Forecast with stochastically generated input (own illustration)

The problem was not the quality of the individual input parameters, as given a set of input parameters, the model could calculate a realistic prediction. However, the difference in duration of the batches led to the forecasts and test data being out of sync. To counter this problem, an algorithm was developed that realigns the batches at the start of the first basket and cuts them when the tapping phase of the last batch starts. Two lengths of batch sets were chosen: 5 and 1.

Aligning the starts of the batches every fifth batch, for example, leads to an RMSE of 0.42, a 15min-RMSE of 0.24 and a mean difference of -0.01. The course of five batches that were aligned at the start of the first basket of the first of five batches is shown in figure 4-15.

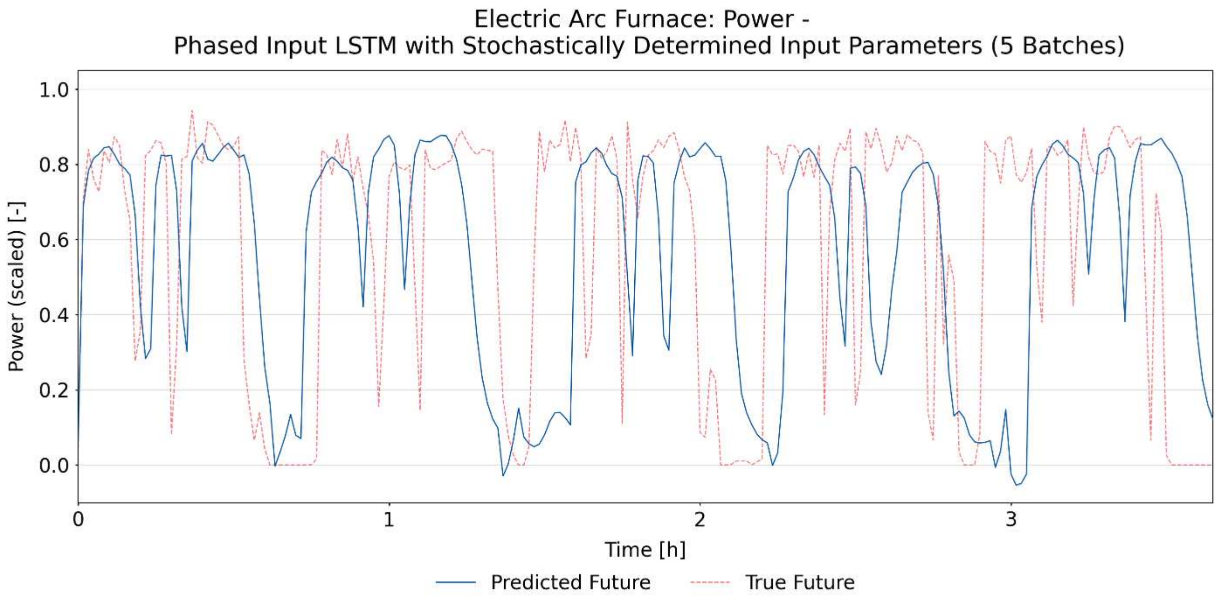

Figure 4-15: Predicted and true power demand of the EAF using a Phased Input Forecast with stochastically generated input, batches are aligned every 5 batches (own illustration)

The set of 5 was chosen to compare qualitatively and quantitatively to the other Phased Input Forecasts, where five batches are displayed (figure 4-12, figure 4-13 and figure 4-14). As shown in figure 4-15 the first batches of the predicted and true values are almost entirely in sync; however, the higher the number of the batch, the less aligned the values are. Nonetheless, the realignment leads to a lower RMSE and a lower 15min-RMSE compared to the non-aligned method. The mean difference remained the same, as the forecast of the stochastically generated input parameters did not change but was realigned.

When aligning the start of every batch, the RMSE is 0.24, the 15min-RMSE is 0.06, and the mean difference is -0.01. The time series of the predicted and true values of one aligned batch is displayed in figure 4-16.

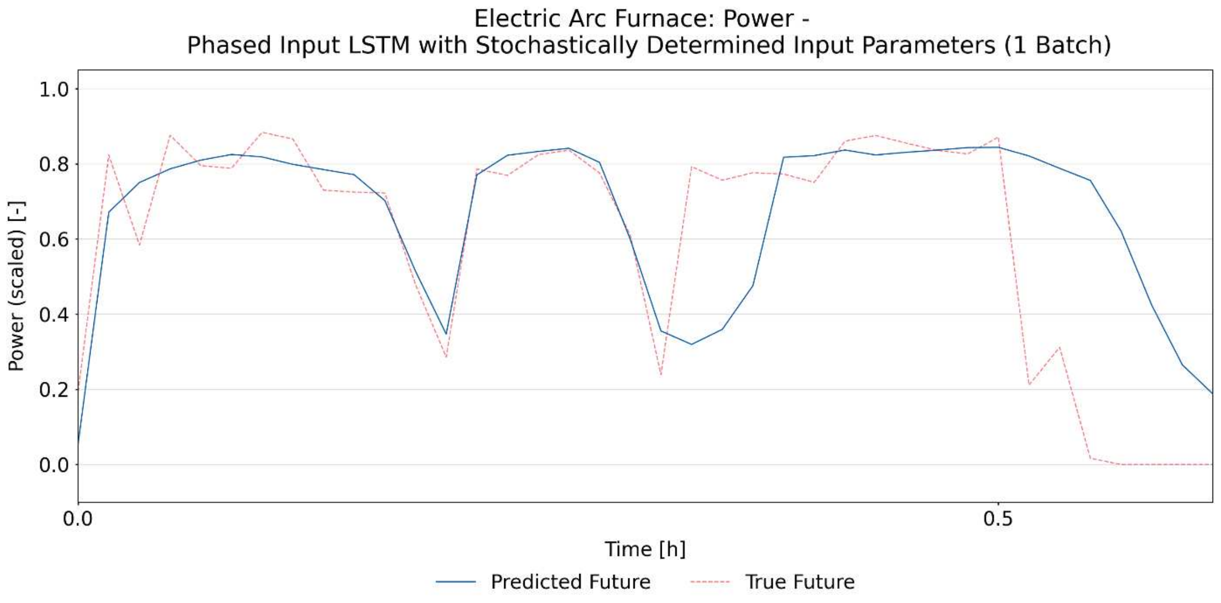

Figure 4-16: Predicted and true power demand of the EAF using a Phased Input Forecast with stochastically generated input, where the batches are aligned after every batch (own illustration)

The set of 1 was chosen to see how good a forecast can be achieved by realigning every batch. In figure 4-16 the same phenomenon as in the set of 5 (figure 4-15) can be seen, although on a smaller scale: While the phases of the first and second basket are well aligned, there is a discrepancy in basket 3 and the fining phase due to the difference in their expected (or stochastically determined) versus their actual duratio. The mean difference stayed the same again with -0.01, as the forecasts did not change but were realigned. While the RMSE is still higher than that of the Perfect Forecast LSTM, it is significantly better than the other forecasting techniques that could realistically be implemented. The 15min-RMSE is even better than that of the Perfect Forecast LSTM; however, it has to be noted that the calculation of this error measure is skewed: to calculate the 15min-RMSE a mean over 15 min is created and their values are compared using the formula in (4-1). If the longer batch is 47 min long, only three means are calculated: from minute 0 to minute 15, 15 to 30 and 30 to 45. The last 2 min values are dropped. Similarly, if the longer batch is 42 min long, only the means from minute 0 to 15 and 15 to 30 are calculated. In this case, the last 12 min would be dropped. This calculation method is not a problem for the other forecasting techniques as the time windows are much larger (five batches or the length of all the test set data). Despite this, the calculated error measure gives an indication over the performance of the forecast.

5 DISCUSSION

Due to the nature of LSTMs, the Perfect Forecast LSTM takes input variables of the same point in time as the output it will predict for. While this way of forecasting is not suitable for a realistic implementation in the scope of the DSM_OPT project, it can serve as a discussion point on whether LSTMs are the right tool for time-series prediction of the available data and as a benchmark for other forecasting techniques.

Using the Perfect Forecast LSTM to predict the future power demand of the aggregates (EAF, LF, two LHs, and D), it is observable that the technique is a viable option for some but not for all aggregates. LSTMs were able to model the EAF, the vertical LH, and the booster fire LH. However, they could not model the LF and the D. This may be because relevant parameters that provide information about the power demand were not in the input parameters or that the process of these aggregates is random and cannot be learned through statistical methods.

Regarding the topic of a realistic forecasting scenario, one of the difficulties that arise when forecasting the power demand of the EAF is that not only the future power demand is unknown but also the values of the input parameters that are used to forecast the power demand. Technically, one would first need to predict the input parameters and use those to forecast the future power demand. However, there is no way to confidently predict the inputs thousands of time steps into the future. This is the reason why other forecasting techniques have been developed, including the Recursive Forecast, Delayed Input Forecast, Multistep Forecast, and Phased Input Forecast.

By looking at the various forecasting techniques, it can be said that another difficulty for forecasting models, that can also be implemented later on in the DSM_OPT project, arises mainly because of varying durations of the phases and batches. This makes it impossible for the Delayed Input Forecast or the Multistep Forecast to learn from the training data and produce realistic forecasts. In addition to that, the duration of future batches has to be sampled from a distribution of past values for making forecasts with phased inputs. Thus, while the course of the batches and input parameters may seem quite regular and predictable (figure 3-2), they are not; instead, the individual phases and batches are always slightly different in length by a few minutes.

The independencies between every batch are another problem for forecasting. Due to each batch being unique, no dependencies can be derived from previous batches, and the time series of the parameters of each batch is to a certain extent random. The physical process of the EAF is controlled by an internal program of Marienhütte of which humans can only change some settings during each batch. Therefore, machine learning can have difficulty finding patterns over multiple batches.

A general point that applies to all forecasting techniques is that the model sometimes predicts slightly negative values instead of 0 for the power demand. This can best be seen in the Perfect Forecast LSTM (figure 4-1), where after the third displayed batch, the power demand is slightly lower than 0 instead of 0. While the results of the forecasts were not enhanced by editing them, this can later in the implementation phase of the forecasts be done by setting all the negative values of the output array to 0 (by using NumPy’s .clip() function, for example).

The grid search, conducted for each forecasting technique, had the aim of finding the best neural network architecture. While it is not possible to exactly quantify the performance improvement after a grid search, in general the error measures were able to improve by about 15%. The model can further be perfected by performing a grid search over more hyperparameters. Due to the time requirement of such a wide-ranging grid search for multiple different models, it was not performed as part of this thesis.

Another area of improvement could be the measurement of the data used for training the model. During the measurement of the parameters, the way of operating the EAF changed for testing purposes, and more natural gas was used as energy input in the process. This led to a slight increase in the power demand over the weekend between the 5th and 6th of June 2021, as shown in figure 5-1. This increase lasted until the end of June.

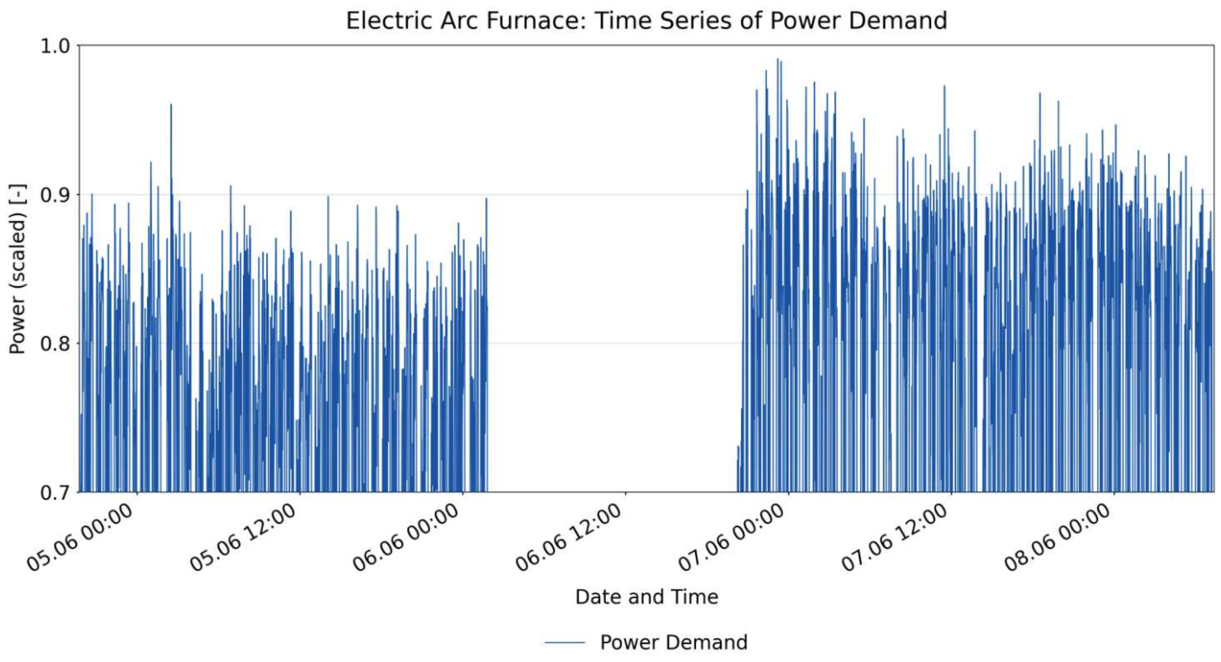

Figure 5-1: Time series of EAF’s power demand between the 5th and 8th of June 2021, with a regular production downtime on Sunday the 6th of June (own illustration)

For a perfect model, the training data should always be realistic based on an operation mode used in the future. Therefore, a recommendation is to only train the final model (that is going to be implemented) on the first month of training data or remeasure the parameters of the EAF in its operating condition. In this thesis, all of the data available was used to have more data for training, validating, and testing and because the increase in total power demand was minor (about 5%). Additionally, the training partition of the data was quite large (80%) so that the models trained got parameters of both operating modes as inputs.

Based on this thesis, it is advised to use a Phased Input Forecast when implementing a forecasting technique for the power demand of the EAF in the project DSM_OPT. The conclusion is that the Recursive Forecast, the Delayed Input Forecast, and the Multistep Forecast were not able to learn the complex time dependencies in batches and could not handle the random relations between batches. Therefore, it is sensible to sample phased inputs from a distribution based on recorded data and produce a weekly forecast. While this data will not accurately predict on minute-resolved power demand for all of the next week, there are two advantages: the first is that it will create a general load profile of what the power demand could look like throughout the following week. The second advantage comes with the realignment of the start of the batches (or better, the first baskets of batches) during operation. This will ensure that the power demand of every 15 min is predicted as accurately as possible for the next batch.

A similar approach can be taken when forecasting the power use of the LHs, as the prediction of their power demands showed great potential when using an LSTM. However, it is advised to perform individual grid searches for the final models of these aggregates. The LF shows a random and unpredictable course of power demand that the LSTM could not learn. Therefore, a stochastic approach is recommended for modelling the power demand of the LF. The input parameters of D were not able to explain the time course of the power demand. Therefore, a stochastic or more classical, regressive time-series approach should also be used in this case.

6 Summary and Future Outlook

Machine learning is a group of state-of-the-art technologies used for data preparation, analysis, and modelling. As part of the literary research in this work, seven main machine learning methods were identified and described: regression analysis, decision trees, Naïve Bayes, support vector machines, neural networks, clustering, and reinforcement learning. Additionally, 12 papers in the field of energy system modelling were assessed with respect to the different models used. The empirical part of this thesis consists of creating different neural networks to forecast the minute-resolved power demand of aggregates in the steel mill Marienhütte.

This Master's thesis was created as part of the project DSM_OPT, which aims to develop a demand side management tool box for companies to increase energy efficiency and optimise the integration of renewable energy sources. For this, accurate ways of forecasting the power demand are vital. This thesis' goal was to find an appropriate way to forecast the power demand of four aggregates of Marienhütte: electric arc furnace (EAF), ladle furnace (LF), ladle heaters (LHs), and dedusting (D). Due to its large power demand, a particular focus was put on the EAF. The forecasting horizon was chosen to be a minimum of one batch and ideally of one week.

According to the papers reviewed, neural networks are outstanding and frequently used when predicting the energy demand of steel mills. Long short-term memory networks (LSTMs) are special kinds of neural networks that, due to their internal architecture, are very suitable for predicting time-series data. The empirical part features six main methods of forecasting based on neural networks, for which an own nomenclature was introduced: Perfect Forecast LSTM, Perfect Forecast MLP, Recursive Forecast, Delayed Input Forecast, Multistep Forecast, and Phased Input Forecast.

The Perfect Forecast LSTM was used to see if and how accurate LSTM networks could in general predict the available time-series data of the Marienhütte and was trained for data of the EAF, LF, the vertical and booster fire LH, and D. The model's predictions accuracy for the EAF and the two LHs were satisfactory; however, the model cannot be used for the LF or D, as the models could not detect a correlation between the input parameters and their outputs.

The other forecasting techniques developed have the advantage of theoretically forecasting multiple time steps ahead and could therefore be implemented in an energy forecasting scenario for Marienhütte. Nonetheless, the prediction accuracy of the Recursive Forecast, Delayed Input Forecast, and Multistep Forecast were insufficient. Using a Phased Input Forecast and stochastically generating input from past measurements led to acceptable results when the batches of the predicted and true data was aligned regularly.

If a machine learning model is thus used to model the power demand of the different aggregates of the Marienhütte, it is recommended to utilise LSTMs for the EAF and LHs, while a stochastical approach should likely be employed for the LF and the D. As for the EAF, the recommendation is to employ a forecast with phased inputs and generate these inputs stochastically from past measurements. The forecasts should then be realigned after every batch to produce the most accurate predictions during operation.

A problem during the creation of the models was that the batches are independent of one another and the power demand of one batch does not necessarily depend on the power demand of the previous batch, as each batch is largely controlled automatically via the process software of Marienhütte. Additionally, even though the batches show regular patterns, their durations are always different. A problem that can arise during implementation of prediction models is that there are sometimes downtimes in production, which can neither be planned nor predicted. These problems cannot be solved without researching and fully understanding the process software that is used for operating the EAF.

To increase the accuracy of the final model for each aggregate, a grid search could be performed for each model over all possible hyperparameters. Additionally, multiple neural networks could be created and an ensemble method (e.g. bagging) could be applied. While this was not done in this thesis due to the large time requirements for a total of seven different models, it can be done when one forecasting method has been decided on for implementation.

Further research could be conducted in the space of time-series forecasting. While machine learning methods showed great potential for some aggregates, traditional or stochastical methods can and should also be used for comparison. Markov chains [49] and auto regressive integrated moving average (ARIMA) models [67], for example, have been used in the iron and steel and energy industry and can be compared to the neural networks used in this thesis. As machine learning could not predict the LF's and D's power demand, these traditional and stochastical methods should especially be applied to them, to see if superior results are possible.

For implementing the models in DSM_OPT, the most suitable forecasting horizon should be determined, as it showed that forecasting one week without knowing input parameters was not feasible. Furthermore, research on dependencies between different aggregates could be analysed, e.g. if a standstill of the EAF also leads to a, potentially time-delayed, lower power demand of the D. This information could improve the performance of the holistic model of Marienhütte Graz that will be implemented for the whole steel and rolling mill.

7 Bibliography

[1] National Academy of Sciences (2020) *Climate Change* [Online], Washington, D.C., National Academies Press. Available at https://nap.nationalacademies.org/catalog/25733/climate-change-evidence-and-causes-update-2020.

[2] Alessia De Vita, Izabela Kielichowska, Pavla Mandatowa, Pantelis Capros and Guillaume Dekelver (2018) *Technology pathways in decarbonisation scenarios* [Online]. Available at https://ec.europa.eu/energy/sites/ener/files/documents/2018_06_27_technology_pathways_-_finalreportmain2.pdf.

[3] Umweltbundesamt *Austria's National Inventory Report 2021: Submission under the United Nations Framework Convention on Climate Change and under the Kyoto Protocol* [Online], Vienna. Available at https://www.umweltbundesamt.at/fileadmin/site/publikationen/rep0761.pdf.

[4] Pulm, P. and Raupenstrauch, H. (2014) *Roadmap Industrie: Energieeffizienz in der Eisen- und Stahlindustrie,* Klima- und Energiefonds der österreichischen Bundesregierung.

[5] Kyriakides, E. and Polycarpou, M. (2006) 'Short Term Electric Load Forecasting: A Tutorial', in Chen, K. and Wang, L. (eds) *Trends in neural computation,* Berlin, Springer, pp. 391–418.

[6] Divina, F., García Torres, M., Goméz Vela, F. A. and Vázquez Noguera, J. L. (2019) 'A Comparative Study of Time Series Forecasting Methods for Short Term Electric Energy Consumption Prediction in Smart Buildings', *Energies*, vol. 12, no. 10, p. 1934 [Online]. DOI: 10.3390/en12101934.

[7] dilawar, u., Khaliq, A. and Kureshi, N. (2022) 'Evaluating Artificial Intelligence and Statistical Methods for Electric Load Forecasting', *International Journal of Innovations in Science & Technology*, 3 special Issue: 4, pp. 59–83 [Online]. Available at https://ideas.repec.org/a/abq/ijist1/v3y2022i4p59-83.html.

[8] Butt, F. M., Hussain, L., Mahmood, A. and Lone, K. J. (2020) 'Artificial Intelligence based accurately load forecasting system to forecast short and medium-term load demands', *Mathematical biosciences and engineering : MBE*, vol. 18, no. 1, pp. 400–425.

[9] Stahl- und Walzwerk Marienhütte GmbH (2022) *Marienhütte: Das Unternehmen* [Online], Graz. Available at https://www.marienhuette.at/das-unternehmen.

[10] Hong, T., Pinson, P., Wang, Y., Weron, R., Yang, D. and Zareipour, H. (2020) 'Energy Forecasting: A Review and Outlook', *IEEE Open Access Journal of Power and Energy*, vol. 7, pp. 376–388.

[11] Zhang, Y. (ed) (2010) *New Advances in Machine Learning* [Online], IntechOpen. Available at https://directory.doabooks.org/handle/20.500.12854/64827.

[12] Richter, S. (2019) *Statistisches und maschinelles Lernen*, Berlin, Heidelberg, Springer.

[13] Maleki, F., Ovens, K., Najafian, K., Forghani, B., Reinhold, C. and Forghani, R. (2020) 'Overview of Machine Learning Part 1: Fundamentals and Classic Approaches', *Neuroimaging Clinics of North America*, vol. 30, no. 4, e17-e32 [Online]. DOI: 10.1016/j.nic.2020.08.007.

[14] Bonaccorso, G. (2018) *Machine Learning Algorithms: Popular Algorithms for Data Science and Machine Learning* [Online], Birmingham, Mumbai, Packt. Available at https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5504925.

[15] Avati, A. (2019) *Bias-Variance Analysis: Theory and Practice* [Online], Stanford University. Available at http://cs229.stanford.edu/summer2019/ BiasVarianceAnalysis.pdf (Accessed 20 January 2022).

[16] Ardabili, S., Mosavi, A. and Várkonyi-Kóczy, A. R. (2020) 'Advances in Machine Learning Modeling Reviewing Hybrid and Ensemble Methods', Springer, Cham, pp. 215–227.

[17] Kozak, K. (2020) *Statistics using Technology*, 3rd edn, Flagstaff, LibreTexts.

[18] Alpaydin, E. (2014) *Introduction to Machine Learning*, 3rd edn, Cambridge, MIT Press.

[19] Rawlings, J. O., Pantula, S. G. and Dickey, D. A. (2001) *Applied regression analysis: A research tool* [Online], 2nd edn, New York, Berlin, Heidelberg, Springer. Available at http://web.nchu.edu.tw/~numerical/course1012/ra/Applied_Regression_Analysis_A_ Research_Tool.pdf (Accessed 25 January 2022).

[20] Dormann, C. F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., Marquéz, J. R. G., Gruber, B., Lafourcade, B., Leitão, P. J., Münkemüller, T., McClean, C., Osborne, P. E., Reineking, B., Schröder, B., Skidmore, A. K., Zurell, D. and Lautenbach, S. (2013) 'Collinearity: a review of methods to deal with it and a simulation study evaluating their performance', *Ecography*, vol. 36, no. 1, pp. 27–46 [Online]. DOI: 10.1111/j.1600-0587.2012.07348.x.

[21] Frochte, J. (2021) *Maschinelles Lernen: Grundlagen und Algorithmen in Python* [Online], 3rd edn, München, Hanser. Available at https://www.hanser-elibrary.com/doi/book/ 10.3139/9783446463554.

[22] Shalabh, S. *Regression Analysis* [Online], Kanpur, Indian Institute of Technology Kanpur. Available at http://home.iitk.ac.in/~shalab/regression/Chapter12-Regression-PolynomialRegression.pdf (Accessed 26 January 2022).

[23] Sharer, Z. (2018) *Curve Fitting* [Online], Johor Bahru, Universiti Teknologi Malaysia. Available at https://people.utm.my/zalilah/ (Accessed 3 February 2022).

[24] Priyanka Sinha (2013) ‘Multivariate Polynomial Regression in Data Mining: Methodology, Problems and Solutions’, *International Journal of Scientific and Engineering Research*, vol. 4, no. 12 [Online]. Available at https://www.researchgate.net /publication/264425037_Multivariate_Polynomial_Regression_in_Data_Mining_ Methodology_Problems_and_Solutions.

[25] Chen, G. (2020) *Polynomial Regression Models* [Online], San Jose, San José State University. Available at https://www.sjsu.edu/faculty/guangliang.chen/Math261a/ Ch7slides-polynomial-regression.pdf.

[26] Waissi, G. *Polynomial Regression - Model Testing* [Online], Arizone State University. Available at http://www.public.asu.edu/~gwaissi/ASM-e-book/module404.html (Accessed 27 January 2022).

[27] Susmita Ray (2019) ‘A Quick Review of Machine Learning Algorithms’, *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)* [Online]. Available at https://www.semanticscholar.org/paper/A-Quick-Review-of-Machine-Learning-Algorithms-Ray/ 8db8166249dfb94dd8d52f88d27917b5755ae049.

[28] Gajewicz-Skretna, A., Kar, S., Piotrowska, M. and Leszczynski, J. (2021) ‘The kernel-weighted local polynomial regression (KwLPR) approach: an efficient, novel tool for development of QSAR/QSAAR toxicity extrapolation models’, *Journal of cheminformatics*, vol. 13, no. 1, p. 9 [Online]. DOI: 10.1186/s13321-021-00484-5 (Accessed 10 February 2022).

[29] Čížek, P. and Sadıkoğlu, S. (2020) ‘Robust nonparametric regression: A review’, *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 12, no. 3 [Online]. DOI: 10.1002/wics.1492.

[30] Cutler, A., Cutler, D. R. and Stevens, J. R. (2012) ‘Random Forests’, in *Ensemble Machine Learning,* Springer, Boston, MA, pp. 157–175.

[31] Haltuf, M. (2014) *Support Vector Machines in Credit Scoring*, Prague, University of Economics in Prague [Online]. Available at https://svm.michalhaltuf.cz/ (Accessed 3 February 2022).

[32] Hemayet Ahmed Chowdhury, Md. Azizul Haque Imon, Anisur Rahman, Aisha Khatun and Md. Saiful Islam (2020) *A Continuous Space Neural Language Model for Bengali*

Language [Online]. Available at https://www.researchgate.net/publication/338621083_ A_Continuous_Space_Neural_Language_Model_for_Bengali_Language.

[33] Elgeldawi, E., Sayed, A., Galal, A. R. and Zaki, A. M. (2021) ‘Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis’, *Informatics*, vol. 8, no. 4, p. 79 [Online]. DOI: 10.3390/informatics8040079.

[34] Mundt, M., Johnson, W. R., Potthast, W., Markert, B., Mian, A. and Alderson, J. (2021) ‘A Comparison of Three Neural Network Approaches for Estimating Joint Angles and Moments from Inertial Measurement Units’, *Sensors (Basel, Switzerland)*, vol. 21, no. 13.

[35] van Veen, F. (2016) ‘The Neural Network Zoo’, *The Asimov Institute*, 14 September [Online]. Available at https://www.asimovinstitute.org/neural-network-zoo/ (Accessed 31 March 2022).

[36] O'Shea, K. and Nash, R. (2015) *An Introduction to Convolutional Neural Networks* [Online]. Available at https://arxiv.org/pdf/1511.08458.

[37] Manaswi, N. K. (2018) ‘RNN and LSTM’, in *Deep Learning with Applications Using Python,* Apress, Berkeley, CA, pp. 115–126.

[38] Olah, C. (2015) *Understanding LSTM Networks* [Online]. Available at https:// colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed 31 March 2022).

[39] Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. (1996) *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise* [Online], Institute for Computer Sccience, University of Munich. Available at https://www.aaai.org/Papers/ KDD/1996/KDD96-037.pdf (Accessed 21 January 2022).

[40] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction* [Online], Cambridge, London, MIT Press. Available at https://web.stanford.edu/class/psych209/Readings/ SuttonBartoIPRLBook2ndEd.pdf (Accessed 10 February 2022).

[41] Azhikodan, A. R., Bhat, A. G. K. and Jadhav, M. V. (2019) ‘Stock Trading Bot Using Deep Reinforcement Learning’, in Saini, H. S., Sayal, R., Govardhan, A. and Buyya, R. (eds) *Innovations in Computer Science and Engineering: Proceedings of the Fifth ICICSE 2017,* Singapore, Springer, pp. 41–49.

[42] Wei, H., Wang, Y., Mangu, L. and Decker, K. (2019) *Model-based Reinforcement Learning for Predictions and Control for Limit Order Books* [Online]. Available at https://arxiv.org/ pdf/1910.03743.pdf (Accessed 25 January 2022).

[43] Oh, E. and Wang, H. (2020) ‘Reinforcement-Learning-Based Energy Storage System Operation Strategies to Manage Wind Power Forecast Uncertainty’, *IEEE Access*, vol. 8, pp. 20965–20976.

[44] Mosavi, A., Salimi, M., Faizollahzadeh Ardabili, S., Rabczuk, T., Shamshirband, S. and Varkonyi-Koczy, A. (2019) ‘State of the Art of Machine Learning Models in Energy Systems, a Systematic Review’, *Energies*, vol. 12, no. 7.

[45] Hahn, H., Meyer-Nieberg, S. and Pickl, S. (2009) ‘Electric load forecasting methods: Tools for decision making’, *European Journal of Operational Research*, vol. 199, no. 3, pp. 902–907.

[46] Carlsson, L. S., Samuelsson, P. B. and Jönsson, P. G. (2019) ‘Predicting the Electrical Energy Consumption of Electric Arc Furnaces Using Statistical Modeling’, *Metals*, vol. 9, no. 9, p. 959.

[47] Kovačič, M., Stopar, K., Vertnik, R. and Šarler, B. (2019) ‘Comprehensive Electric Arc Furnace Electric Energy Consumption Modeling: A Pilot Study’, *Energies*, vol. 12, no. 11, p. 2142.

[48] Zhou, D., Gao, F., Guan, X., Chen, Z., Li, S. and Lu, Q. (eds) (2004) *Daily Electricity Consumption Forecast for a Steel Corporation Based on NNLS with Feature Selection*, Singapore, IEEE.

[49] Dock, J., Janz, D., Weiss, J., Marschnig, A. and Kienberger, T. (2021) ‘Time- and component-resolved energy system model of an electric steel mill’, *Cleaner Engineering and Technology*, vol. 4, p. 100223.

[50] Gajic, D., Savic-Gajic, I., Savic, I., Georgieva, O. and Di Gennaro, S. (2016) ‘Modelling of electrical energy consumption in an electric arc furnace using artificial neural networks’, *Energy*, vol. 108, pp. 132–139.

[51] Chen, C., Liu, Y., Kumar, M. and Qin, J. (2018) ‘Energy Consumption Modelling Using Deep Learning Technique — A Case Study of EAF’, *Procedia CIRP*, vol. 72, pp. 1063–1068.

[52] Carlsson, L. S., Samuelsson, P. B. and Jönsson, P. G. (2020) ‘Using Statistical Modeling to Predict the Electrical Energy Consumption of an Electric Arc Furnace Producing Stainless Steel’, *Metals*, vol. 10, no. 1, p. 36.

[53] Carlsson, L., Samuelsson, P. and Jönsson, P. (2019) ‘Using interpretable machine learning to predict the electrical energy consumption of an electric arc furnace’, *Stahl und Eisen (1881)*, vol. 139, no. 9, pp. 24–29 [Online]. Available at http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1505062&dswid=-6694.

[54] Klempka, R. and Swiatek, B. (2009) ‘In Day-Ahead Electricity Forecasting’, *Electrical Power Quality and Utilisation, 2009. EPQU 2009. 10th International Conference on*, pp. 1–5.

[55] Lange, S. *Evolutionary Algorithms* [Online], Freiburg, Albert-Ludwigs-Universität Freiburg. Available at https://ml.informatik.uni-freiburg.de/former/_media/documents/teaching/ss11/ml/04_evolution.pdf.

[56] Durrett, R. (2011) *Essentials of Stochastic Processes* [Online], Duke University. Available at https://services.math.duke.edu/~rtd/EOSP/EOSP2E.pdf (Accessed 31 March 2022).

[57] Abdi, H. *Partial Least Squares (PLS) Regression* [Online], The University of Texas at Dallas. Available at https://personal.utdallas.edu/~herve/Abdi-PLS-pretty.pdf (Accessed 11 February 2022).

[58] Balaram Panda (2019) *Hyperparameter Tuning,* University of Auckland [Online]. Available at https://www.researchgate.net/publication/340720901_Hyperparameter_Tuning (Accessed 10 April 2022).

[59] Heaton, J. (2017) *Heaton Research: The Number of Hidden Layers* [Online], St. Louis (Missouri), Heaton Research, Inc. Available at https://www.heatonresearch.com/2017/06/01/hidden-layers.html (Accessed 15 April 2022).

[60] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E. and Stoica, I. (2018) *Tune: A Research Platform for Distributed Model Selection and Training* [Online]. Available at http://arxiv.org/pdf/1807.05118v1.

[61] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht and Ameet Talwalkar (2018) ‘Massively Parallel Hyperparameter Tuning’ [Online]. Available at https://openreview.net/forum?id=S1Y7OOlRZ.

[62] *Time series forecasting* (2022) [Online]. Available at https://www.tensorflow.org/tutorials/structured_data/time_series (Accessed 7 May 2022).

[63] Chollet, F. (2018) *Deep learning with Python*, Shelter Island, Manning.

[64] Kingma, D. P. and Ba, J. (2014) *Adam: A Method for Stochastic Optimization* [Online]. Available at https://arxiv.org/pdf/1412.6980.

[65] Tawhid, A., Teotia, T. and Elmiligi, H. (2021) ‘Chapter 13 - Machine learning for optimizing healthcare resources’, in Kumar, P. (ed) *Machine Learning, Big Data, and IoT for Medical Informatics,* San Diego, Elsevier Science & Technology, pp. 215–239.

[66] Bokde, N. D., Yaseen, Z. M. and Andersen, G. B. (2020) ‘ForecastTB—An R Package as a Test-Bench for Time Series Forecasting—Application of Wind Speed and Solar Radiation Modeling’, *Energies*, vol. 13, no. 10, p. 2578 [Online]. DOI: 10.3390/en13102578.

[67] Xavier Serrano Guerrero, Luis-Fernando Siavichay, Jean-Michel Clairand and Guillermo Escrivá Escrivá (2020) ‘Forecasting Building Electric Consumption Patterns Through Statistical Methods’, in Botto-Tobar, M. (ed) *Advances in Emerging Trends and Technologies: Volume 2,* Cham, Springer International Publishing AG, pp. 164–175.

8 APPENDIX

8.1 Grid Search: Perfect Forecast LSTM – EAF

The following code was used to perform the grid search for the Perfect Forecast LSTM model of the EAF.

```
# Import libraries

import pandas as pd
import numpy as np
import matplotlib as mpl
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler
import ray
from ray import tune
from ray.tune.integration.keras import TuneReportCallback
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

np.set_printoptions(linewidth=1000,precision=5)
pd.set_option('display.max.columns', 15)
mpl.rcParams['figure.figsize']=(17, 8)
mpl.rcParams['axes.grid']=True

# Read data and choose input parameters

df=pd.read_csv('EAF_cont.csv', sep=';', encoding='cp1252',
               na_values=['#DIV/0!'])
df['timestamp_utc'] = pd.to_datetime(df['timestamp_utc'],
                                     infer_datetime_format=True)
print("ORIGINAL DF:")
for i in range(0,len(df.columns)):
    print(str(i)+": "+df.columns[i])
    i+=1
print("\t")

df.drop(df.columns[[]], axis=1, inplace=True) #6,7,8,9,10,11,
df.dropna(inplace=True)
df_input = df.drop(df.columns[[0]], axis=1)

print("DF COLUMNS USED FOR MODELLING:")
for i in range(0,len(df_input.columns)):
    print(str(i)+": "+df_input.columns[i])
    i+=1
print("\t")

print(len(df_input))

# Scale data and create TimeSeriesGenerators

win_length=1440
```

```
batch_size=64
num_features=len(df_input.columns)-1
num_epochs=30
print(num_features)

data=df_input.to_numpy()

train_set, test_set=train_test_split(data, test_size=0.1, shuffle=False)

train_set, val_set=train_test_split(train_set, test_size=0.11,
shuffle=False)

scaler=MinMaxScaler()
train_scaled=scaler.fit_transform(train_set)
val_scaled=scaler.transform(val_set)
test_scaled=scaler.transform(test_set)

x_train=train_scaled[:,1:]
y_train=train_scaled[:,0]

x_val=val_scaled[:,1:]
y_val=val_scaled[:,0]

x_test=test_scaled[:,1:]
y_test=test_scaled[:,0]

train_generator=TimeseriesGenerator(x_train, y_train, length=win_length,
                                      sampling_rate=1,
                                      batch_size=batch_size)
val_generator=TimeseriesGenerator(x_val, y_val,
                                      length=win_length, sampling_rate=1,
                                      batch_size=batch_size)
test_generator=TimeseriesGenerator(x_test, y_test,
                                      length=win_length,
                                      sampling_rate=1,
                                      batch_size=batch_size)

# Define an objective function

def train_model(config):

    n_nodes=config['n_nodes']
    n_layers=config['n_layers']

    model=tf.keras.Sequential()
    i=0
    while i < n_layers:
        if i==0:
            model.add(tf.keras.layers.LSTM(n_nodes,
input_shape=(win_length, num_features),
                                          return_sequences=True))
        else:
            model.add(tf.keras.layers.LSTM(n_nodes,return_sequences=True))
        i+=1
    if i==0:
        model.add(tf.keras.layers.LSTM(n_nodes, input_shape=(win_length,
num_features),
                                       return_sequences=False))
    else:
        model.add(tf.keras.layers.LSTM(n_nodes, return_sequences=False))

    model.add(tf.keras.layers.Dense(1, activation='linear'))

    model.compile(loss=tf.losses.MeanSquaredError(),
                  optimizer=tf.optimizers.Adam(),
                  metrics=['MAE', 'MSE', 'MAPE'])

    mycallbacks=[
        TuneReportCallback({'Mean Squared Error': 'MSE'}),
    ]

    model.fit(train_generator, epochs=num_epochs,
                        validation_data=val_generator,
                        shuffle=False,
                        verbose=1,
                        callbacks=mycallbacks)
    print("---DONE---")


# Define a search space

search_space = {
    'n_layers': tune.grid_search([0,1,2]),
    'n_nodes':
tune.grid_search([3,5,7,10,15,20,25,30,40,50,60,70,80,90,100])
}

asha_scheduler=tune.schedulers.ASHAScheduler(
    max_t=num_epochs,
    grace_period=1,
    reduction_factor=4,
    brackets=1
    )

# Start a Tune run and print the best result

if __name__ == "__main__":
    ray.init(num_gpus=1)
    analysis=tune.run(train_model,
                  config=search_space,
                  metric="Mean Squared Error",
                  mode="min",
                  resources_per_trial={"gpu": 1},
                  scheduler=asha_scheduler,
                  verbose=3
                  )
    print('Best Configuration:', analysis.get_best_config)
    print('Best Trial:', analysis.best_trial)
    print('Logdir of Best Configuration:', analysis.best_logdir)

    df_results = analysis.results_df
    df_results.to_csv('MA_PerfectForecast_LSTM_GridSearch_Results.csv')
```

8.2 Final Model: Perfect Forecast LSTM – EAF

The following code was used to create, train and verify the Perfect Forecast LSTM model of the EAF.

```
# Import libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

np.set_printoptions(linewidth=1000,precision=5)
pd.set_option('display.max.columns', 15)

# Read data and choose input parameters

df=pd.read_csv('EAF_cont.csv', sep=';', encoding='cp1252',
               na_values=['#DIV/0!'])
df['timestamp_utc'] = pd.to_datetime(df['timestamp_utc'],
                                     infer_datetime_format=True)
print("ORIGINAL DF:")
for i in range(0,len(df.columns)):
    print(str(i)+": "+df.columns[i])
    i+=1
print("\t")
df.drop(df.columns[[]], axis=1, inplace=True) #6,7,8,9,10,11,
df.dropna(inplace=True)
df.reset_index(inplace=True)
df_input = df.drop(df.columns[[0,1]], axis=1)
print("DF COLUMNS USED FOR MODELLING:")
for i in range(0,len(df_input.columns)):
    print(str(i)+": "+df_input.columns[i])
    i+=1
print("\t")

# Scale data and create TimeSeriesGenerators

win_length=1440
batch_size=64
num_features=len(df_input.columns)-1

data=df_input.to_numpy()

train_set, test_set=train_test_split(data, test_size=0.10, shuffle=False)
train_set, val_set=train_test_split(train_set, test_size=0.11,
shuffle=False)

scaler=MinMaxScaler()
train_scaled=scaler.fit_transform(train_set)
val_scaled=scaler.transform(val_set)
test_scaled=scaler.transform(test_set)
```

```
x_train=train_scaled[:,1:]
y_train=train_scaled[:,0]

x_val=val_scaled[:,1:]
y_val=val_scaled[:,0]

x_test=test_scaled[:,1:]
y_test=test_scaled[:,0]

train_generator=TimeseriesGenerator(x_train, y_train, length=win_length,
                                      sampling_rate=1,
                                      batch_size=batch_size)
val_generator=TimeseriesGenerator(x_val, y_val,
                                       length=win_length, sampling_rate=1,
                                       batch_size=batch_size)
test_generator=TimeseriesGenerator(x_test, y_test,
                                        length=win_length,
                                        sampling_rate=1,
                                        batch_size=batch_size)

# Create Single Model

model=tf.keras.Sequential()

model.add(tf.keras.layers.LSTM(70, input_shape=(win_length, num_features),
                               return_sequences=False))
model.add(tf.keras.layers.Dense(1, activation='linear'))

mycallbacks=[
   tf.keras.callbacks.ModelCheckpoint('checkpoint_bestmodel.keras',
                                      save_best_only=True,
                                      monitor='val_loss', mode='min'),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10,
                                     mode='min'),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                                         min_lr=1e-4, patience=2,
verbose=1)
]

model.compile(loss=tf.losses.MeanSquaredError(),
              optimizer=tf.optimizers.Adam(),
              metrics=['MAE', 'MSE', 'MAPE'])

print(model.summary())

# Train Model

history=model.fit(train_generator, epochs=100,
                            validation_data=val_generator,
                            shuffle=False,
                            callbacks=mycallbacks)

np.save('history_bestmodel_final.npy',history.history)

print("---DONE---")

# Make Predictions
```

```
model.load_weights(filepath='checkpoint_bestmodel_final.keras')

results=model.evaluate(test_generator, verbose=0)
predictions=model.predict(test_generator)

# Create final scaled dataset

total_length=len(data)
train_length=len(y_train)
val_length=len(y_val)
test_length=len(y_test)

df_pred_scaled=pd.concat([pd.DataFrame(predictions,
                                       columns=['Predicted Future
Scaled']),
                          pd.DataFrame(y_test[win_length:],
                                       columns=['True Future Scaled'])],
                         axis=1)

df_datetime=df.iloc[-test_length+win_length:,0].reset_index(drop=True)
df_final_scaled=pd.concat([df_datetime, df_pred_scaled], axis=1)

print(df_final_scaled)

df_final_scaled.to_csv('PerfectForecastLSTMEAF.csv')
```