Forecasting Power Consumption of Manufacturing Industries Using Neural Networks

Bachelor's Thesis

Lorenz Boguhn

March 31, 2020

Kiel University
Department of Computer Science
Software Engineering Group

Advised by: Prof. Dr. Wilhelm Hasselbring
Sören Henning, M.Sc.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 31. März 2020

Abstract

The reduction of power consumption should be reached for many ecologic and economic reasons. Since a large part of the power is consumed by the industrial sector, we propose to increase the energy efficiency and applying the DevOps approach. Forecasting the power consumption of manufacturing industries can increase the energy efficiency and also allows using anomaly detection systems and predictive maintenance for manufacturing industries. In this work, we designed multiple different model variations, using different neural networks, forecasting methods and features for the forecasting of three industrial power consumers. We also evaluated our different model variations with the aim to find the model variation with the highest accuracy. By providing the evaluations, we offer conclusions about different forecasting methods, features used and different neural networks for forecasting the power consumption of industrial power consumers. We also propose a forecasting microservice as a base for an anomaly detection and predictive maintenance.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays the power that is produced in Germany is consumed 50 % by the industry sector [Javied et al. 2016]. Worldwide, The manufacturing segment is responsible for around 33% of the total energy use and for around 38% of the CO2 emissions [Mohamed et al. 2019]. The reduction of the power consumption would lead to a better efficiency in ecological as well as in economic terms. Therefore, finding ways to optimize the energy consumption becomes more and more important.

Forecasting energy consumption is a way to solve this problem by increasing energy efficiency. Also, forecasts could be used to potentially restructure inefficient processes that use power [Shrouf et al. 2014]. Forecasts are a helpful tool to plan operations and their schedules. They can help to reduce the overall energy consumption as well as the energy consumption during load peaks.

If fluctuations are small in the energy demand, the energy grid system will be more efficient [Albadi and El-Saadany 2008]. For this reason, large-scale power consumers pay different energy costs than normal consumers. They have to pay a price for: each month, each kilowatt hour, and a maximum demand rate. The maximum demand rate depends on the maximal power demand of the consumer in a paying period. Therefore, load peaks generate significant costs for enterprises. Forecasts may show these load peaks before they occur and therefore may help operators to avoid or weaken them. The results of forecasts could also serve as reference values for alerting systems [Henning et al. 2019]. Alerting systems can use reference values from forecasts to detect failures and malfunctions and alert the operators to ensure a fast reaction [Shrouf et al. 2014]. Furthermore, forecasts are also required for predictive maintenance. Predictive maintenance is an alternative approach to conventional maintenance based on regular time intervals. In the conventional approach, machines are maintained unnecessary often. Discover defects and failures before they occur refer to predictive maintenance. The goal of predictive maintenance is to reduce the cost of maintenance, while avoiding defects of machines and degrading quality of products [Li et al. 2017].

In order to produce forecasts for the energy consumption, data of the previous energy consumption is needed. Fortunately, the digitization of the manufacturing industry leads to new abilities of industrial machines: They are increasingly intelligent, autonomous, and

able to communicate [Henning 2018]. This new level of automation in the industry is often referred to as Industry Internet of Things(IIoT) or Industry 4.0 [Lasi et al. 2014]. With Industry 4.0, manufacturing enterprises are increasingly able to perform: automated data collection, automated analysis of data, and automated visualization of data [Hasselbring et al. 2019].

Yet, many small and medium-sized enterprises (SME) hesitate to adopt Industry 4.0 solutions [Hasselbring et al. 2019]. They have to face major investment and maintenance costs, as well as lack of knowledge. An approach to tackle these challenges is Industrial DevOps [Hasselbring et al. 2019]. Industrial DevOps is an approach for transferring methods and culture of DevOps to industrial production environments. In Industrial DevOps, people of different departments and domains work together. It consists of two principles: a continuous adaptation and improvement process and lean organizational structures.

Together with the Titan Control Center, the Industrial DevOps approach is applicable on the energy management of the Industry 4.0 [Hasselbring et al. 2019]. The Titan Control Center is used to analyze power consumption data [Henning and Hasselbring 2019]. The power consumption data is integrated via multiple different sources into the Titan Control Center. Afterwards, it is aggregated, analyzed, and visualized by the Titan Control Center. The Titan Control Center works near real time and is part of the Titan Platform[1] which is presented in Section 2.6.

To extend the analysis capabilities of the Titan Control Center, the goal of this thesis is to develop a software component for forecasting power consumption. This forecast component can be integrated as a microservice into the Titan platform. This software component produces the forecasts by an artificial neural networks. Neural networks are able to approximate any continuous function to the desired accuracy [Hippert et al. 2001; Haykin 1998]. They are nonlinear and nonparametric methods and can therefore model complex nonlinear relationships [Park et al. 1991; Zhang et al. 1998]. Using artificial neural networks to produce forecasts has two major advantages: Firstly, artificial neural networks do not require any assumptions of the functional relationship between variables in advance. Secondly, artificial neural networks can generalize and infer. Given a sample of input and output vectors, artificial neural networks are able to automatically map the relationship between them. They therefore can learn relationships and patterns. Those can also be stored in the network's parameters [Park et al. 1991].

1.2 Goals

In order to design a power consumption forecasting software component, this thesis has four goals. For this reason, the first step is to design models for machine learning that can be used to generate predictions. In the next step one of the designed model variations has

[1] https://industrial-devops.org/

to be used to develop a forecast microservice. After developing such a service, it can be integrated into Titan. Afterwards the approach is evaluated.

G1: Development of a Machine Learning Model

To obtain a forecast model, machine learning and more precisely neural networks can be used [Hippert et al. 2001]. We have to build a model of the neural network we want to use. This model is needed since it determines which data should be predicted and to ensure a certain quality of the forecast. It defines how accurate the forecast can be and determines the results. In the model building process there are plenty of parameters to choose. They define the structure and capability of the network, ranging from the general network to the single neurons. The parameters are likely to be changed after some experimentation and have to be continuously evaluated to gain an accurate and precise model.

G2: Development of a Forecast Microservice

Designing the forecast approach as a microservice yields a better scalability [Hasselbring 2016]. For example, operators could deploy multiple instances of the forecast service running in parallel, each forecasting for a part of the power consumers. These could also run in a high-performance cloud if needed.

G3: Integration of the Forecast Microservice into Titan

Integrating the microservice into the Titan platform extends it by the capability to forecast. The forecasts could be used to calculate the approximate power consumption for the future. They can also show possible failures in machines if the forecasts differs from the actual consumption. That would indicate anomalies, which could be found automatically using forecasts. The Titan platform could then be used to predict possible load peaks in power consumption to support production operators in preventing or mitigating them.

G4: Evaluation of the Forecast Approach

To evaluate the forecast approach and its corresponding implementation in a microservice, real production data has to be used with the service. By using power consumption data acquired in the real manufacturing enterprise *Kieler Nachrichten*, we can check whether the forecast service produces reliable forecasts. In order to reach this goal, different forecasting models are evaluated. This models will be evaluated in different scenarios.

1.3 Document Structure

After this introduction, the remaining thesis is structured as follows: Chapter 2 contains a short introduction of the foundations and the used technologies, Chapter 3 explains how models using neural network for forecasting can be developed, Chapter 4 shows different variations of models, Chapter 5 contains the design and integration of a forecasting microservice, Chapter 6 describes an evaluation of the models, Chapter 8 discusses the related work and Chapter 9 gives conclusions about the thesis and a perspective about potential future work.

Chapter 2

Foundations and Technologies

This chapter presents required foundations and technologies used throughout this thesis.

2.1 Machine Learning

Machine learning is a subfield of artificial intelligence [Gron 2017]. It focuses on programs that can learn from data. Machine learning uses statistical models without explicit instructions. Systems for machine learning are trained using data [Chollet 2017]. After the training of such a system, it should be able to recognize and generalize patterns and inference information. This can be used to handle new and unknown information and to get a deeper insight of what patters are in the data. It is used in a lot of fields such as classification, object identification in images, or time series forecasting.

2.1.1 Artificial Neural Networks

Artificial neural networks, often simply referred to as neural networks, are one method of machine learning, which got a lot of attention in recent years. Neural networks are inspired by the biological neurons [Haykin 1998; Hippert et al. 2001; Din and Marnerides 2017]. This is shown in Figure 2.2. A neural network is a parallel and distributed processor, which is built of simple process units. Human brains are composed of a number of interconnected simple processing elements called neurons [Zhang et al. 1998]. All neurons get an input signal. This is the total information they can receive from the other neurons. In artificial neural networks, a connection between two neurons has a specific weight. The inputs are then processed through an activation function. This produces the transformed output signal for other neurons.

Artificial neural networks consists of multiple layers as shown in Figure 2.1. They usually have one input layer and one output layer. Between the input and the output, there can be any amount of layers. These layers are referred to as hidden layers. All layers consist of cells.

In the following subsections different cell types are described. A neural network obtains knowledge of its environment by learning [Haykin 1998]. The gathered knowledge of a neural network is stored in the weights of the interconnections. Therefore, learning of a neural network is the process of modifying the weights. For the learning of a neural

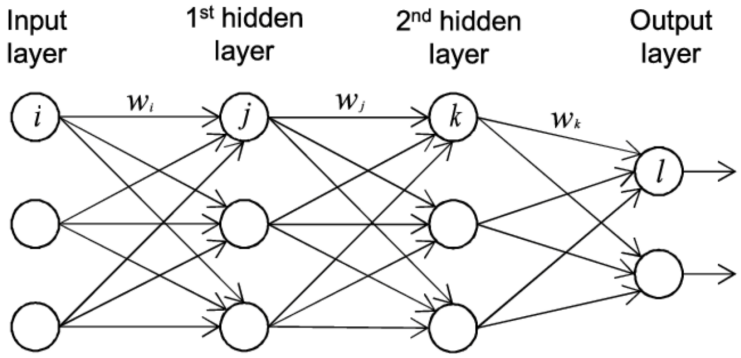


Figure 2.1. A deep neural network [Vieira et al. 2017]

network, to do a specific task, a measurement of how far the network is away from the expected results is needed. This is done by the loss function. The loss function computes a loss score by calculating the distance between the network output and the actual value. This actual value is often referred to as label. The score is then used by the network optimizer to adjust the network accordingly. An optimizer is the implementation of the Backpropagation-algorithm [Chollet 2017]. The used loss function in this thesis is the mean-average-error(MAE) function [Willmott and Matsuura 2005]:

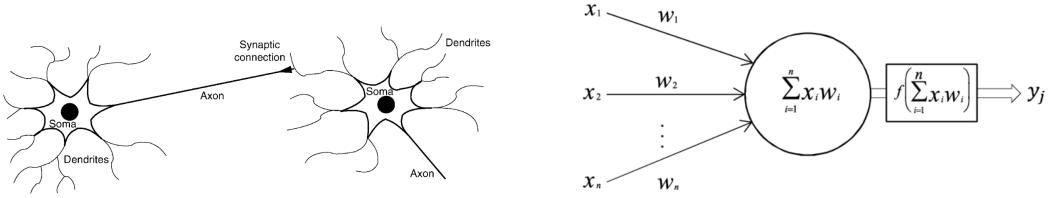
$$\frac{1}{n} \times \sum_{i=0}^{N} |x - x_{pred}|$$

Here, the x is the prediction. x_{pred} is the feature value that should be the output. A feature represents one of the properties of an object of a problem. The MAE function is chosen to get a good average forecasting. The lower a loss value, the higher is the accuracy of the network. If the loss score on the training data is lower than on new data, the neural network is overfitting. Overfitting is on of the main problems of neural networks [Gron 2017; Chollet 2017]. Besides overfitting, a neural network can also be underfitting. Underfitting means that the neural network is not able to generalize enough and therefore has a high loss score.

Network in which the output of the neuron are only flowing in one direction are called Feedforward Neural Networks (FNN) [Gron 2017]. Neural networks that have an internal loop are referred to as Recurrent Neural Networks (RNN) [Chollet 2017].

Dense Cells

Dense Cells are a linear operation on the input vector [Chollet 2017]. The result of the linear operation is passed into the activation function that yields the output. The mathematical model behind the neuron in Figure 2.2 is a dense cell. Multiple Dense Cells form a densely connected layer [Gron 2017].



(a) Biological Neuron [Rozenberg et al. 2011] **(b)** Mathematical Neuron [Vieira et al. 2017]

Figure 2.2. The mathematical neuron picked up the most significant features of a biological neuron.

LMST - Long Short Term Memory

The Long Short Term Memory (LSTM) is a specialized RNN architecture [Jozefowicz et al. 2015]. In LSTM neurons differ from the normal neuron architecture. They can learn to identify whether an input is important [Gron 2017]. Besides, they can store this information and use it if needed. They can also learn how long to store such inputs and drop the input if needed. This information is controlled by the gates. There are three different kinds of gates: the input gate, the forgot gate, and the output gate. LSTM cells are, using their gates, able to learn long term dependencies [Gron 2017]. The LSTM cells are used in the structure of some of the neural networks that are designed for the forecasting.

GRU - Gated Recurrent Units

Gated recurrent units are very similar to the LMST [Hope et al. 2017]. They are not as powerful as LSTM cells but not as computational expensive as LSTM cells [Chollet 2017]. Compared to an LSTM, they have less parameters. Gated recurrent units outperformed the LSTM cells on a lot of tasks [Jozefowicz et al. 2015]. They also have a memory mechanism.

2.1.2 Deep Learning

The central task in machine learning is to transform input data to output data [Chollet 2017]. In order to do this, the learning process changes the inner representation of the data. If the learning of a network is successful, it has found useful inner representations of the data. In this context useful means closer to the expected output. Abstract representation of the input data is used to get better abstraction for new data. Thus, learning can be perceived as the automated searching process for better representations of data.

The term "deep" in deep learning references the usage of multiple inner representations successively. Therefore using multiple successive layers of increasingly useful representations of data, is called deep learning.

2.1.3 Tensors

All current machine learning systems use multidimensional arrays as a basic data structure [Chollet 2017; Abadi et al. 2016]. These multidimensional arrays are called tensors. Tensors have three key attributes: number of axes, shape, and datatype. The number of axis represents the number of dimensions a tensor has. The shape of the tensor is defined by a tuple of integers. This tuple describes how many entries a tensor has at each axis. All contents of the entries of a tensor have a specified datatype. A tensor that only contains a single number is called a scalar or 0-dimensional tensor. A tensor that contains an array of data is called a vector or 1-dimensional tensor. A vector has only one axis. It can contain multiple entries along its axis. A matrix or 2-dimensional tensor is a description for an array of vectors. It has two axis often called rows and columns. A 3-dimensional tensor would be an array that contains matrices and so on.

A time series is usually encoded as a 3-dimensional tensor. They are build in the following pattern: Sample, Timestamp, Feature. The time series tensor is an array that contains multiple samples. In this samples there are multiple features with different values for different points in time.

2.2 Forecasting Methods using Artificial Neural Networks

The forecasting models used are multivariate or univariate. A univariate forecasting model only describes dependencies one one variable. The univariate forecasting model is restricted to one variable and can therefore not capture complex dependencies between multiple variables. Multivariate forecasting models describe dependencies of several variables. If data have dependencies between the different variables, the multivariate forecasting model may be able to generalize those. Therefore, the results of the multivariate forecasting model may be better.

There are also different approaches to which kind of outputs a model should produce. If the forecasting model predicts only one value from an input, it is referred to as single-step forecasting model. Otherwise, if the forecasting model is able to predict multiple values that are the potential future values, the model will be referred to as multi-step model.

Depending on the choice of model, the inputs for the neural network have different shapes. For an univariate single-step model, an input vector contains the selected feature as input vector and an output scalar that contains values from the column that is predicted. For the univariate multi-step model, there is a set of multiple input vectors that contain all the selected features as input vectors and an output scalar that holds the value that is predicted. If the output is also a vector, instead of a scalar, the model would be a multi-step model.

2.3 The Deep Learning API Keras

Keras[1] is a high-level API for building and training deep learning models [Chollet 2017]. It makes fast modeling possible. Keras does not handle low-level operations such as tensor manipulation and differentiation. It relies on a backend engine, that is capable of tensor operations [Chollet 2017]. Keras can run on three different backend engines: TensorFlow, Theano, and the Microsoft Cognitive Toolkit. The same code can run on the CPU and on the GPU using Keras.

The TensorFlow framework and Keras are used in the Forecasting microservice. The neural networks in this thesis are designed and build using Keras. Additionally, the training, the predictions and the evaluation of the neural networks are performed using Keras. Therefore the Keras API is needed for this thesis.

2.4 The Machine Learning Framework TensorFlow

TensorFlow[2] [Abadi et al. 2016; Gron 2017] is an open-source machine learning framework. It is a complex library that offers distributed numerical computations using data flow graphs [Chollet 2017]. A computation in TensorFlow is described by a directed graph, which consists of a set of nodes and represents a dataflow computation. In order to use TensorFlow, a computational graph has to be constructed. The computational graph is defined in Python and then translated by TensorFlow to optimized C++ code [Gron 2017]. Tensorflow divides the computational graph into smaller chunks and allows to execute the chunks on multiple CPUs and GPUs in parallel. It also supports distributed computing, enabling training neural networks on multiple servers. TensorFlow uses the Eigen[3] library for tensor operations if running on the CPU. If it is running on GPU, it uses the NVIDIA CUDA Deep Neural Network library[4] (cuDNN).

The forecasting approach uses machine learning and TensorFlow can perform and optimize all needed vector operations.

2.5 The Microservice Architectural Pattern

The Microservice architectural pattern is a pattern for building software. Software is built in the microservice pattern, if it consists of multiple different functional units [Ebert et al. 2016; Kang et al. 2016]. These units can, independently from each other, fail, be scaled, maintained, and reused. These units are referred to as microservices. The Microservice architectural pattern tries to overcome the problems of limited scalability of monolithic architectures [Hasselbring 2016; Hasselbring and Steinacker 2017]. They are a full-stack

[1] https://keras.io/
[2] https://www.tensorflow.org//
[3] http://eigen.tuxfamily.org/
[4] https://developer.nvidia.com/cudnn

implementation for a business capability [Hasselbring 2016]. The use of the pattern provides scalability and adaptability [Hasselbring and Steinacker 2017; Balalaie et al. 2016]. It also yields fast delivery cycles through the integration of development and operations aspects of the DevOps approach [Ebert et al. 2016].

The Titan Control Center (see Section 2.6.2) is built in the microservice architectural pattern. Each business capability is built in a microservice. Thus, for an integration of the forecasting microservice into the Titan Control Center, the microservice structure is necessary for the forecast service.

2.6 The Industrial DevOps Platform Titan

The Titan project is a research project [Titan Project 2018] that investigates the digitalization of small and medium-sized enterprises (SME). The project developed a role model that describes the cooperation between different departments [Hasselbring et al. 2019]. The Industrial DevOps platform Titan is a platform for agile process integration and automation [Henning et al. 2019; Hasselbring et al. 2019]. It offers monitoring for production environments and data analysis in real-time. The following features are provided by Titan: data integration, data analysis, visualization. The Titan Platform can support the integration of the DevOps approach. It consists of multiple different components that will be described in this section. The forecast service is integrated in the Titan platform as an additional service.

2.6.1 Titan Flow Engine

The Titan Flow Engine applies the principles of flow-based programming. Flow-based programming is a pattern that describes applications not as deterministic processes [Morrison 2010]. They are described as asynchronous "black box" processes that communicate over data steams. The configuration of the Titan components is done via a visual modeling language [Hasselbring et al. 2019]. This language is provided by the Titan Flow Engine. It enables the integration of sensors and allows configuration using a visual modeling language. This enables persons trained in modeling to work together with domain experts and perform configurations or changes. For each integration of a tool or infrastructure a modular software component called brick is created. The bricks are not limited to integrate systems. Bricks can also perform the collection, manipulation or creation of data. So, for example, they can execute tasks such as transforming or filtering data. Bricks have an output port and an input port and can be connected by them. Bricks can be connected and combined in various ways. Those combined bricks then can form a new structure as a new application. In Titan, a network of multiple connected bricks referred to as flow. So there is the possibility to model all sorts of business and production processes with a flow.

The Titan Flow Engine supplies the Titan platform with data. Therefore it is essential for the forecast service.

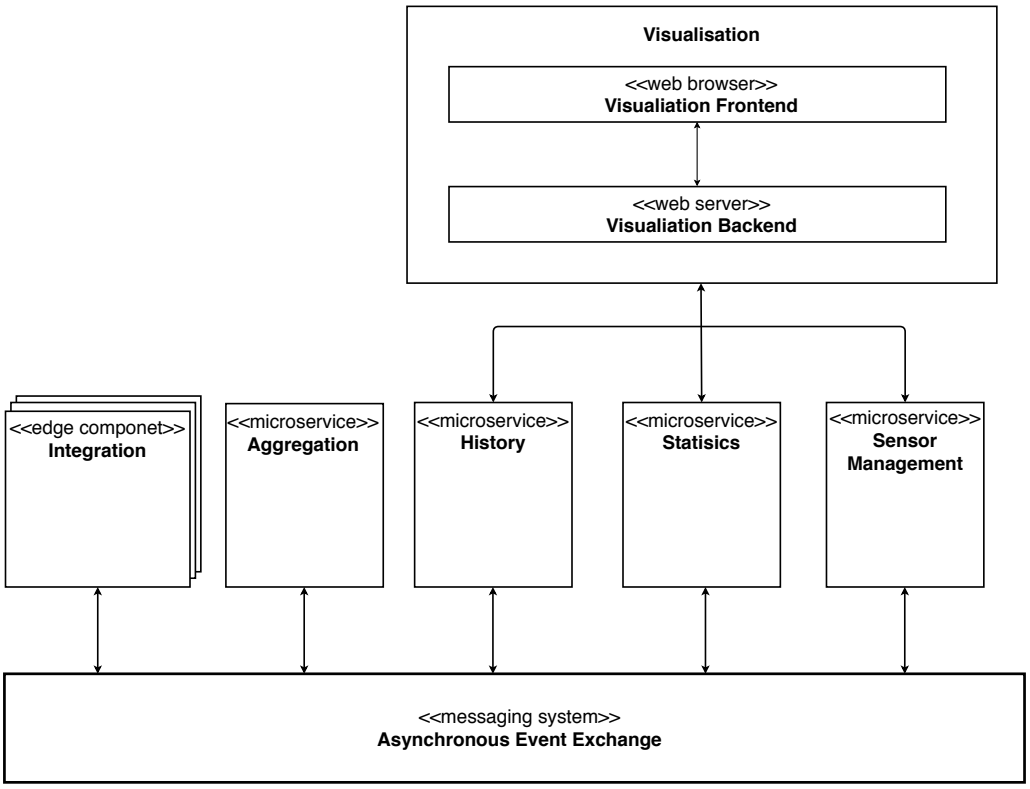


Figure 2.3. The Titan architecture based on [Henning et al. 2019]. Except for the Integration, all other mircoservices form together with the visualization component the Titan Control Center.

2.6.2 Titan Control Center

The Titan Control Center is a distributed implementation of a software system that offers monitoring and analysis [Henning et al. 2019; Henning 2018]. The Titan Control Center is implemented in the microservice design pattern. In Figure 2.3 the different components are shown. There are four different microservices: Aggregation, History, Statistics and the Sensor Management. Additionally, there is a Visualization component in the Titan Control Center (see **??**). They are all shown in Figure 2.3. The Aggregation service collects data and provides aggregated data. The History service manages sensor data the access to it. This counts for single sensor data as well as for aggregated data for groups of sensors. The Statistics service builds and provides statistical data. The Sensor Management service manages system-wide configurations. It manages configurations like a hierarchical model that specifies which sensors exist and specifies groups. The measurements of sensor groups

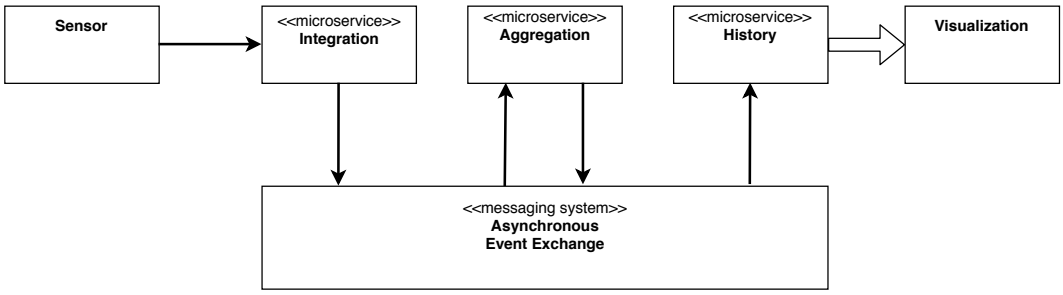


Figure 2.4. The Titan Control Center event exchange. The History service is an example and can be replaced by any of the other microservices.

can be aggregated. The communication of the services works via the asynchronous message system Apache Kafka as well as via HTTP/REST (see Section 2.6.3).

2.6.3 Control Center Event Exchange

The communication of the microservices of the Titan Control Center works mainly via the asynchronous message system Apache Kafka. Data is first measured by a sensor. The sensor then sends the data via various ways to the individual integration. Devices and machines are usually from different manufactures [Henning et al. 2019]. Therefore the data supply of these machines varies widely and the integrations of sensors are build individually. The integration then handles the sensor data by its own specific logic and adds the sensor data along with the sensor identifier to the Kafka topic input. The integration is therefore a Kafka producer (see Section 2.8). All the microservices are Kafka consumers and some of them are also producers. The Control Center event exchange is shown in Figure 2.4. Services like the History Service can process the data with their own logic. For example the History Service can store the data. The communication between the services and the Visualization is done via REST/HTTP. If a sensor that is in a hierarchical group publishes a new measurement, the Aggregation Service also produces a new value [Henning et al. 2019]. It calculates the new value for the hierarchical group and publishes the values to the output topic.

2.7 The *Kieler Nachrichten Druckerzentrum* Data Set

The *Kieler Nachrichten Druckerzentrum* Data Set contains the energy consumption of a real-world production environment. It was collected at the *Kieler Nachrichten Druckerzentrum*[5] over 5 years, from July 6, 2014 to January 18, 2019. The data set contains power consumption measurements for 6 different machines, recorded every 15 minutes. These machines are

[5]https://kn-druckzentrum.de/

represented by strings as shown in Table 2.1. The data set contains the power consumption of the following machines: two air condition machines, one air compressor, the rotary printing press as well as the power consumption for the outside lighting and the offices. The head of the data set is shown in Table 2.2. It represents the data set containing the machine names, the timestamps in milliseconds since 1970, and the current power consumption in watt. The table has 926830 rows. The data set was created by using the Titan Control Center in a previous data analysis within the scope of the Titan project.

In Figure 2.5 the power consumption of all machines in January 2015 is visualized. The y-axis shows the power consumption in watt and the x-axis the date in days. Figure 2.5 reveals that there are multiple patterns in the data set. The first pattern is the weekly consumption of e-druckluft-e2, e-kalte-1, and e-buro-aufenthalt. The e-druckluft-e2 machine has the highest peaks. These peaks differ in their maximal heights from week to week. Besides, e-aussenbeleuchtung-sud-nea shows periodical consumption, since its consumption depends on the sunrise and the sunset.

The forecast microservice should be capable of forecasting power consumption data of manufacturing industries. For this reason, the *Kieler Nachrichten Druckerzentrum Data Set* is used to show this capability of the forecasting service.

Table 2.1. Machines of *Kieler Nachrichten Druckerzentrum Data Set*

machine id strings	machine	ID
e-aussenbeleuchtung-sud-nea	outside lights	m0
e-buro-aufenthalt	office lights	m1
e-druckluft-e2	air compressor	m2
e-kalte-1	air conditioner 1	m3
e-kalte-2	air conditioner 2	m4
e-rotation-e4	rotary printing press	m5

Table 2.2. First five lines of the data set

	0	1	2
0	e-kalte-1	1402504200000	94800.001413
1	e-kalte-1	1402505100000	93200.001389
2	e-kalte-1	1402506000000	93200.001389
3	e-kalte-1	1402506900000	92800.001383
4	e-kalte-1	1402507800000	92000.001371
...	...	...	...

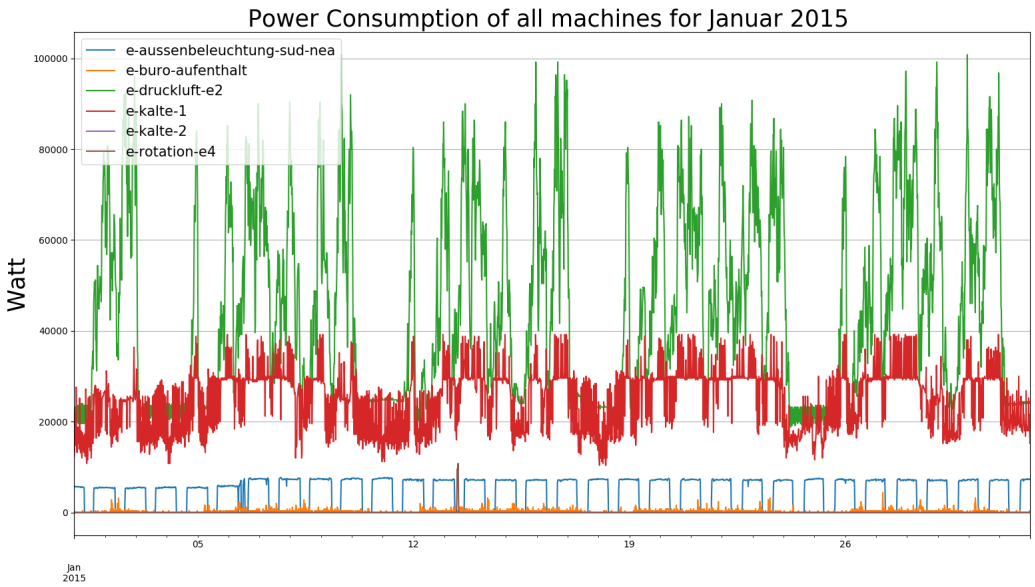


Figure 2.5. Power Consumption from the *Kieler Nachrichten Druckzentrum*

2.8 The Distributed Messaging System Kafka

Apache Kafka is a distributed publish-subscribe messaging system [Wang et al. 2015; Narkhede et al. 2017]. It is a messaging system designed for scalability and a high throughput.

In an architecture using Kafka, there are different types of processes: broker, consumer, and producer. When Kafka is running on a server cluster, the servers are called brokers. Producers can publish messages to a specific topic to a broker. The messages are stored and provided as key-value pairs together with a timestamp and its topic by the broker. Inside topics, the data can be divided into different partitions. The partitions can be replicated and distributed among the brokers. The consumer can subscribe at a broker to specific topics and will then get all the messages regarding what they subscribed to.

Titan uses the Apache Kafka framework for asynchronous communication between the different microservices [Henning et al. 2019]. In order to produce a forecast, data is needed. This data is read by the forecast microservice using Kafka. It is necessary for reading the data and is therefore needed for the forecast service and the thesis.

Chapter 3

Design of a Machine Learning Model

In order to forecast data using neural networks, a model has to be designed first. This model determines the capabilities of the forecast service. Therefore the development of a suitable model will be described in this chapter. This includes the data preprocessings that have to be done and the way of structuring neural networks.

3.1 Data Preprocessing

To work with the data set, we have to load it first. With the pandas[1] framework, the data can be loaded and manipulated. After loading the data, it has to be manipulated. These manipulations also referred to as preprocessing, ensure that the data can be fed into a neural network. After the preprocessing, the data is expected to have an enhanced quality. As shown in Table 2.2, the data is stored in table consisting of the columns machine name, timestamp, and measured active power consumption. In the following, the preprocessing will be described in more detail.

3.1.1 Restructuring

The first manipulation is the restructuring of the table structure. The timestamp is set as table index. Furthermore, it is converted to a more human-readable form such as 2014-06-11 16:30:00. Afterwards the data can be re-sampled using pandas, resulting in one data point for every 15 minutes. Entries are add for the missing timestamps containing NaN for all machines. This has to be done since some measurements are missing in the data set. In the new structure, the power data of all machines are in the columns as shown in Table 3.1. Afterwards the missing data points are filled using data interpolation. This data interpolation works by approximating the data points using the neighboring data points. The pandas framework is also used here.

3.1.2 Feature Engineering

The features of the data set, currently contained in the data set, are the power consumption of each machine. In order to forecast data, some additional information as features could

[1] https://pandas.pydata.org/

Table 3.1. First five lines of the data set after applying the data transformation steps of Section 3.1.1

	0	1	2	3	4	5
0	m1	m2	m3	m4	m5	m6
2014-06-11 16:30:00	0.0	0.0	34400.000513	94800.001413	0.000000	8400.000125
2014-06-11 16:45:00	0.0	0.0	43600.000650	93200.001389	0.000000	8400.000125
2014-06-11 17:00:00	0.0	0.0	46400.000691	93200.001389	0.000000	8800.000131
2014-06-11 17:15:00	0.0	0.0	48800.000727	92800.001383	0.000000	8400.000125
2014-06-11 17:30:00	0.0	0.0	47200.000703	92000.001371	0.000000	8400.000125
...	...	...	...	...	...	...

be helpful. For each measurement, the day in the year, day of the week, and hour of the day it was recorded is added. Therefore all features currently available are the power consumptions of each machines, the day in the year, the day of the week, and the hour of recordings. After adding this information the next step is the data normalization. We expect better results by generalizations through the added features. The more data with an underlying pattern there is, the better this pattern can be detected. Depending on the current needs, additional features could be added (see Section 4.4).

3.1.3 Training, Validation, and Test Sets

In the next step, the data is split into three sets: a training set, a validation set, and a test set. These sets consist of pairs of tensors. One part of the pair is the input tensor and the other one is the output tensor that is also called target or label. The input tensor is fed into the neural network and the resulting prediction is compared to the label. During the learning phase, the network parameters are adjusted according to the results of the comparison and the used learning algorithm. For this adjustments the data from the training set is used. The validation set is used after every finished training epoch, to provide an overview how much the neural network adapted to the training data set and if it overfitts. An epoch is the number of times a neural network sees the complete data set.

The test set is used to unbiasedly evaluate the final model predictions. In this way, it can be shown that the neural network not only remembers the training data and the corresponding correct labels.

As described above, the *Kieler Nachrichten Druckerzentrum Data Set* is split as follows: 70% training data, 15% validation data, and 15% test data. This means that from the 161443 rows, 116011 are in the training data, 24216 in the validation data, and 24216 in the test data. To select the sets, we take the rows 0 to 116011 for the training data, the rows 116012 to 140227 for the validation data and rows 140228 to 161443 for the test data. This split is realized by defining three different generators that are further described in Section 3.1.5. Each of these is responsible for one of the sets, which is done by setting the start and end point according to the choices above.

3.1.4 Data Normalization

The next step is a normalization of the data. It is not necessary as functional requirement for a model using neural networks, but it will help the neural network to adjust its parameters in the learning process. This normalization speeds up the learning process and enhances the accuracy of the results a neural network will output. For normalization the following formula is used:

$$x = \frac{data - mean}{std}$$

The *std* is the standard deviation of the given column. The *mean* is the mean of a column. The *data* is the current value in a row of a column. Finally, x represents the new value. It is very important to calculate the standard deviation and mean from the training set.

3.1.5 Data Generation

To train a neural networks with data, the data needs to be a (b,(l,f),t)-tensor, where b is the batch size, l is the length of each sample, f is the number of features, and t the tensor containing the labels. In this case the batch size is the number of samples in one batch. A sample is a ((length,features),targets) - tensor. The length describes how many values of the features are taken. The features can be understand in this context as the columns in the data, and the targets are the corresponding labels. Therefore, the data has to be constructed in a way that is has this shape. This can be done by generating the needed sets by using the *TimeseriesGenerator*[2] from Keras. It is threadsafe and therefore there can be multiple threads generating the input data. The *TimeseriesGenerator* than can be used to define generators for all three sets. These generators yield the data in the needed shape as NumPy arrays.

To define a *TimeseriesGenerator* the following parameters are needed: a data set, a length value, a delay value, a start and an end index, a target set, a shuffle value, and a batch size value. An example usage is shown in Listing 3.1. x is an array that contains the numbers from zero to 50 in the first feature. The other features are the numbers incremented by one to four. Next, y is an array that contains the numbers from four to 58 The generator is defined with a batch size of two, therefore it yields samples with the size two. These samples contain tuples with three arrays, each containing five entries, and two arrays, each containing two entries. The first three arrays contain the data from the x array. This is what the length parameter describes. The other two arrays contain the data from the y array.

Listing 3.1. Example usage of the *TimeseriesGenerator*

```
1 x = np.array([[i,i+1,i+2,i+3,i+4] for i in range(50)])
2 y = np.array([[i+4,i+5,i+6,i+7,i+8] for i in range(50)])
3
4 train_gen = keras.preprocessing.sequence.TimeseriesGenerator(x,
5                                 targets=y,
```

[2] https://keras.io/preprocessing/sequence/

```
                                length=3,
                                start_index=0,
                                end_index=50,
                                shuffle=True,
                                batch_size=2)

print(train_gen[0])
# Results in:
#(array([[[0, 1, 2, 3, 4],[1, 2, 3, 4, 5],[2, 3, 4, 5, 6]],
#        [[1, 2, 3, 4, 5],[2, 3, 4, 5, 6],[3, 4, 5, 6, 7]]]),
# array([[ 7,  8,  9, 10, 11], [ 8,  9, 10, 11, 12]]))
```

In Listing 3.2, the concrete use of the *TimeseriesGenerator* is shown. The input for the generators is the normalized-data that data can holds all features needed. The target is depending on the forecasting model. This target holds data of one selected machine shifted in the future. It either holds one value or multiple. It depends on the shifting number, how far into the future the predictions of the neural network will be.

Listing 3.2. Training data generator using the *TimeseriesGenerator*

```
target = normalized_data.iloc[:,target_column].shift(periods=(-shift))
output = 1
normalized_data= normalized_data.iloc[:-shift,:]
target = target.iloc[:-shift]

train_gen = keras.preprocessing.sequence.TimeseriesGenerator(normalized_data,
                                targets=target,
                                length=length,
                                start_index=0,
                                end_index=(val_start-1),
                                shuffle=True,
                                batch_size=batch_size)
```

3.2 Defining and Training a Neural Network

For our forecasting neural networks we use the Keras API. The input layer of a neural network has to be defined using the shape of the training data. This is important, since with an shape that is to small not all the data could be feed the neural network. That would result in losing information. Therefore the shape is defined by the length of the samples and the count of features used. The output layer determines the shape of the output. There can be several other layers betweem the input and the output layer. These layers can have different numbers of cells in it. An example definition is shown in **??**

After defining the neural network structure, it has to be compiled. We define an optimizer and a loss function for this. The optimizer specifies how the loss value from the loss function will be used to adjust the network [Chollet 2017]. For the models, the *Adam* optimizer is used. The used loss function is the *MAE* function. Since the values are all normalized, the normalized *MAE* is obtained. A normalized *MAE* for a feature can be denormalized to gain the concrete *MAE* by multiplying the normalized *MAE* with the feature std.

Listing 3.3. Defining and compiling of a neural network using the Adam optimizer and the MAE.

```
1 shape=(length,features)
2
3 model = keras.Sequential()
4 model.add(keras.layers.GRU(12, input_shape=shape))
5 model.add(keras.layers.GRU(30, activation='sigmoid',return_sequences=True))
6 model.add(keras.layers.GRU(30, activation='sigmoid'))
7 model.add(keras.layers.Dense(output))
8 model.compile(optimizer=keras.optimizers.Adam(), loss='mae')
```

To train a neural network, the following parameters have to be defined: the training set, a number of epochs, and the validation set. When using generators, the training and the validation sets have to be *training* and *validation* generators. The number of Epochs represents the number of times the training set is used to adjust the neural network. The validation set is used after every epoch to yield a loss value. This loss value calculated on the validation set can be used for comparisons. These comparisons can help to detect whether the model overfitts or underfitts. Additionally, since the *TimeseriesGenerator* from Keras is used, some multiprocessing options can be used additionally. For the design of the models, the *fit* function[3] from Keras, shown in Listing 3.4, is used. The *fit* function returns a dictionary that holds the loss values for the training and the validation. This dictionary can be used to visualize the progress the neural network made during the training. It allows to quickly see if the neural network overfitts or underfitts.

Listing 3.4. Training a model using generators

```
1 history = model.fit(x=train_gen,
2                     epochs=EPOCHS,
3                     use_multiprocessing=True,
4                     workers=8,
5                     max_queue_size=20,
6                     validation_data=val_gen)
```

[3] https://keras.io/models/model/

3.3 Predictions and Accuracy

After compiling and training a neural network it it may be used to produce predictions by the means of the *predict* function in Keras. The function has to be supplied with a vector of normalized history data as input. This will yield a normalized prediction. To get a denormalized prediction, the inverse of the normalization formula is used:

$$data = (x \times std) + mean$$

Afterwards the prediction can be used further. The forecast service uses this predict function to for its forecasts.

An neural network can be evaluated in terms of accuracy using the *evaluate* function from Keras and the test set or the corresponding test generator. This is shown in Listing 3.5. This *evaluation* function returns the loss value for the test set. It is used in Chapter 6. That loss value may be used for comparisons that show if the neural network is overfitting or underfitting. If the loss of the training is higher than the loss of the validation, the model variation is presumably overfitting. Otherwise, if the the validation loss is higher than the training loss the model is presumably underfitting. The model variation is near to the best possible model variation, if the difference in both loss values is very small. The test loss is an indicator of how the model perform on unseen data. Therefore, if the test loss is higher than the training and validation loss, the model variation is presumably over fitting. A model can be further fine tuned using the knowledge of the loss values.

Listing 3.5. Evaluating a model using the test set

```
1 results = model.evaluate_generator(test_gen, test_steps/batch_size)
```

Chapter 4

Forecasting Model Variations

In order to design and find an accurate forecasting model, we designed a lot of different model variations arising from the model in Chapter 3. For this reason the different forecast model variations are described in this chapter. The models vary in the choice of machine to forecast, in the choice of the forecasting horizon, of the network, of additional features, and of the forecasting approach.

4.1 Power Consumers

As described in Section 2.7, the *Kieler Nachrichten Druckzentrum* Data Set contains the data of 6 different machines. In the scope of this thesis, three of them are used in model variations, namely the air compressor, the outside lighting, and one of the air conditioning machines. They have the identifiers: e-druckluft-e2, e-aussenbeleuchtung-sud-nea, and e-kalte-1. A power consumer is referred to as target in the following, if it is the power consumer that is forecasted in the model variation.

The Air Compressor

The air compressor is used as target in the models as it is consuming power mainly depending on the performance of the production. It is the largest consumer compared to the others. It is expected that models predicting the power consumption of this machine will have the lowest accuracy.

The Outside Lighting

The outside lighting is used as target in models as it consumes power depending mainly on external influences. It is turned on depending on the outside light situation. The outside lighting, for example, gets turned on in the sunset and turned off in the sunrise. We expect that Models that forecast the power consumption of the lighting yield a better accuracy than models forecasting one of the other power consumers.

The Air Conditioner Machine

The first air conditioner machine is used as target, as its power consumption depends on external and internal influences. It is consuming the second largest power consumer. The accuracy of models that forecast the power consumption of this machine are expected to be between the lighting and the air-compressor.

4.2 Time Parameters

There are two different time parameters that influence a model variation. The first is the forecasting horizon. It determines how far the prediction is in future. For example, eight hours in the future. The model variations discussed in this thesis have the forecasting horizon of one hour, two hours, eight hours, and 24 hours. Secondly, the period of time of historical data that is the input of a neural network for one prediction. For this input, the period of one week is chosen. Since the measurements are taken every 15 minutes a length of $4 \times 24 \times 7 = 672$ for every input is chosen.

4.3 Forecasting Methods

As described in Section 2.2 there are multiple methods of forecasting using artificial neural networks. In the scope of this thesis all four of them are used.

The Univariate Single Step Forecasting Method

The Univariate Single Step forecasting method is used in few of the model variations for a comparison to the multivariate forecasting model variations. This kind of method uses the data of one machine as a feature and produces forecasts for the energy consumption data of this machine for one point in time. In initial experiences the multivariate model variations yielded better results.

The Univariate Multi Step Forecasting Method

The Univariate Multi Step forecasting method is also used in a few model variations, for the same reason as described above. It uses data of one machine as feature and produces forecasts of the energy consumption data for one machine for multiple points in time.

The Multivariate Single Step Forecasting Method

The next forecasting method is the Multivariate Single Step method. It uses all features and produces forecasts of the energy consumption data for one machine for one point in time. We use this method on a lot of model variations, since forecasting with multivariate model

potentially leads to better results. Since the multivariate model can capture dependencies between multiple variables, it can recognize patterns and generalize them. They therefore have more information about the data which could lead to a higher accuracy of the forecasts.

The Multivariate Multi Step Forecasting Method

The last forecasting method is the multivariate multi step forecasting method. It uses all features and produces forecasts for the energy consumption data of one machine for multiple points in time. We use it on the same number of model variations as the Multivariate Single Step method.

4.4 External Features

In addition to the time features (see Section 3.1.2), there are external features that could lead to better forecasts. Therefore values for external features are collected and used in the models variations. With these external features, we expect the model variations to predict with higher accuracy. The data preprocessing remains the same after adding the features as described in Section 3.1.

4.4.1 Public Holidays as a Feature

For a newspaper organization public holidays are an important factor. On a public holiday, most newspaper publishers don not publish a new newspaper issue. Therefore it is of importance for the power consumption of the printing machines if the following day is a public holiday. Since the machines will not be used the day before the public holiday to print a new issue. For this reason, we add the holiday feature. This feature holds a boolean value. It is created using the *holidays*[1] python package. If the feature is a 1, the day is a holiday, otherwise it is 0.

In order to consider that the printing machines are in use the day before the public holiday, we use a lagged feature. This feature has the same values like the holiday feature described above, except that it is delayed for one day. The lagged feature could be useful to predict the energy consumption of the e-druckluft-e2 machine. Therefore, it is expected that model variations using this features will yield a higher accuracy.

4.4.2 Outside Temperature as a Feature

Another feature that could be useful is the air temperature. If the air temperature is very high, some machines have to be cooled during the production process. Therefore the air

[1] https://pypi.org/project/holidays/

temperature should have an influence on the power consumption of the air conditioner machine, for example.

We collect this feature using the *Deutsche Wetter Dienst*. This organization offers a free use of their weather data collection. The feature holds a float variable that represents the temperature at this time. Unfortunately, some of the data is corrupt and some of the measurements are missing. Therefore an interpolation as described in Section 3.1.1 is used on this data. Additionally, the measurements are only taken every hour. Therefore it has to be re-sampled to every 15 minutes and the missing time period is filled using the last valid temperature.

4.5 Different Neural Networks

More complex neural networks are able to express more complex issues. In order to explore this, we designed four different network that use different cells in their structure. All of them have the same number of input and output cells. The number of input cells depends on how many features are used and the output cells determine how much values one prediction contains. Therefore the number of output cells are depending on the step model (see Section 2.2).

The first is a neural network that uses Dense cells. In in the following this is called the Dense Neural Network. The implementation of it is shown in Listing 4.1. It uses the *relu* function as activation function and has only two layers. At first the input is flattened into a one dimensional array. The next layer is the input layer, which is followed by the output layer.

Listing 4.1. A Neural Network out of Dense Cells

```
1 model = keras.Sequential()
2 model.add(keras.layers.Flatten(input_shape=shape))
3 model.add(keras.layers.Dense(32, activation='relu'))
4 model.add(keras.layers.Dense(output)
```

The second neural network is a network using GRU cells. It is referred to in the following as GRU Neural Network and the concrete implementation is shown in Listing 4.2. It consists of four layers: one input layer, two hidden layers and one output layer. For the activation function the *sigmoid* function is used.

Listing 4.2. First Network with GRU in Structure

```
1 model = keras.Sequential()
2 model.add(keras.layers.GRU(features,return_sequences=True,input_shape=shape))
3 model.add(keras.layers.GRU(30, activation='sigmoid',return_sequences=True))
4 model.add(keras.layers.GRU(30, activation='sigmoid'))
5 model.add(keras.layers.Dense(output))
```

The next model is using LSTM cells and is in the following referred to as LSTM Neural Network. Its implementation corresponds to Listing 4.2, except that the GRU cells are exchanged with LSTM Cells.

As last neural network we use a modified version of the GRU Neural Network. It uses a technique called dropout, since this can help if neural networks are overfitting. By using dropout on a layer, it randomly drops an amount of features by setting them to zero. This can be useful to disturb the learning of random patterns that are not helpful for the network.

4.6 Summary

After setting the decisions for the model parameters, there are three possibilities for the machine, four for the time period, four for the network, and two for the step model. Therefore there are $3 \times 4 \times 4 \times 2 = 96$ model variations. Additionally, there are 16 model variations with the univariate forecasting and 24 model variations with only selected features. This makes a total of $96 + 16 + 24 = 136$ models. All of those are evaluated in Chapter 6 These model variations are all trained for 20 epochs, since this already takes, in a few variations, up to eight hours to complete the training.

Chapter 5

Design of a Forecast Microservice

In this section design and implementation of a microservice, which forecasts power consumptions of manufacturing industry is described. This forecast microservice uses model variations from Chapter 4 for the forecasts and is integrated in the Titan Control Center.

5.1 Architecture of the Microservice

The forecasts produced by the microservice, are forecasts of the power consumption of industrial machines. Our power consumption forecast component is designed according to the microservice architectural pattern. We designed the microservice using a docker container and Docker[1] is used for this. It allows us to define a container, that can run on any server. In this container all the needed software is preinstalled. Containerization also prohibits the interfering of different services. Since the containerization is used, the forecasting mircoservice component can be executed on a Kubernetes[2] cluster, as described in [Henning and Hasselbring 2019]. We design the service to use a configuration file. This is done to enable the service to be flexible for changes in the environment and to be able to quickly change the used neural network.

To produce forecasts using neural networks, historical data is needed as input. In order to retrieve this data the service has to be able to communicate with other services that could provide this data. For this reason, it uses the asynchronous publish-subscribe message system Kafka.

The forecast microservice is designed in the Pipe-and-Filter architectural style. It is an often used style that offers the potential combination of high modularity and high throughput [Wulf et al. 2017]. A Pipe-and-Filter architecture consists of pipes and filters. Pipes are connections between filters. Filters have one data input and one data output. Filters perform one processing step on the data between their input and output.

The flow of data in the microservice is shown in Figure 5.1. The service consists of four filters: the *Reader* filter, the *Normalizer* filter, the *Forecast* filter, and the *Writer* filter. There are three pipes in the microservice. One between the *Reader* and the *Normalizer*, one between the *Normalizer* and the *Forecast*, and one between the *Forecast* and the *Writer*.

[1] https://www.docker.com/
[2] https://kubernetes.io/de/

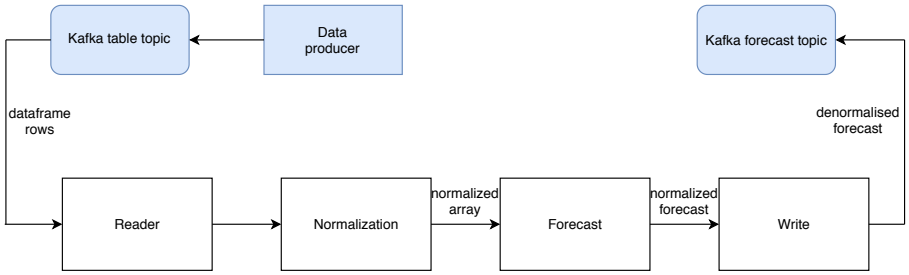


Figure 5.1. The flow of data between the four filters of the forecast microservice. The blue components are not part of the forecast microservice.

The Reader filter is responsible for the reading of data from Kafka. Responsible for the normalization and the storing is the *Normalizer* filter The *Forecast* filter is responsible for producing the forecasts. The denormalization and the of the forecast and the writing of forecasts to Kafka is done by the *Writer* filter.

The neural networks used for the forecasting are loaded from the microservice. Therefore, the neural networks have to be constructed and trained before they can be used in the microservice. This means that the individual forecasting capabilities of a forecasting microservice depends on the internally used neural network.

5.2 Integration into the Titan Control Center

The integration of the forecast microservice into the current Titan Control Center architecture is shown in Figure 5.2. There are multiple overlaying microservices shown, since multiple instances of the microservice can run in parallel. These multiple instances are used to predict the power consumptions of different machines or aggregated power consumptions at the same time. The data necessary for the forecasts is received over the Kafka server of the Titan Control Center. Since the current implementation of the Titan Control Center does not support a configurable simulation of receiving live data, we wrote a script that allows to read *dataframes* that are formatted in the scheme shown in Section 3.1.1 and publish them to a topic called *table*. This script is the *Data producer* in Figure 5.1. To integrate the microservice into the Titan Control Center, it has to be added to the *docker-compose* file of the Titan Control Center. Additionally, the two topics *forecast* and *table* have to be added to the Kafka topic in the file. The forecasts microservices can consume data from the Kafka server and produce their forecasts. Afterwards, these are published in the *forecast* topic. In the future, the forecasts can be used in anomaly detection services or other future components of the Titan Control Center.

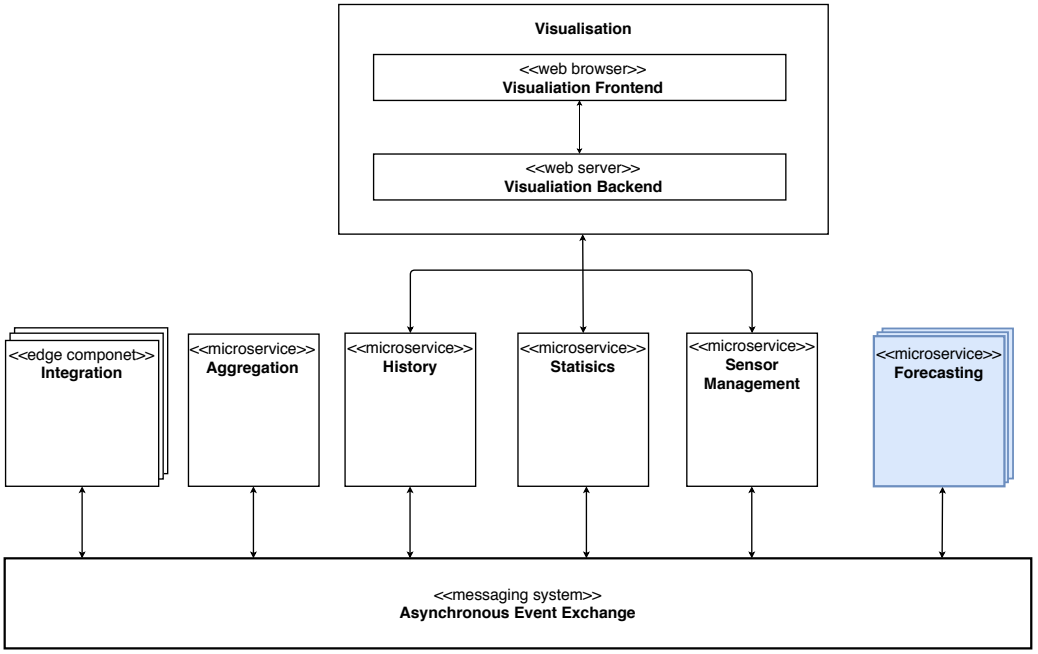


Figure 5.2. Titan architecture extended by the forecast microservice derived [Henning and Hasselbring 2019]. The forecasting microservice is highlighted in blue.

5.3 Implementation of the Forecast Microservice

The forecast service is implemented in Python version 3.6.8. To containerize the application, we defined a dockerfile. This file configures the setup of the container and the starting of the forecast microservice. In this step, we pull an official image of TensorFlow and install the needed dependencies. Also the needed files are copied in the virtual disc space of the container. This configuration is a YAML[3] file that contains the neural network and the further settings.

The four filters (seeSection 5.1) are implemented as four different threads. For the pipes queues from the python library are used. Initially after starting the service, the settings are set according to the in the handed configuration file. Afterwards, it defines a Kafka *consumer*. The *main* method defines the queues, defining the threads, and starting the treads. The queues are: *ReaderQueue*, *TransformationQueue*, *WriterQueue*.

The *Reader* filter is started with an topic and a queue. The used topic is the *table* topic. It connects to the configured Kafka server and starts to listen to the configured topic using the defined *consumer*. The configured topic is the *table* topic. If a new message is published

[3]https://yaml.org/

in the table topic, it puts the values from the message in its queue.

The *Normalizer* filter is started with the two queues: *ReaderQueue* and *TransformationQueue*. It takes elements from *ReaderQueue* and puts elements into *TransformationQueue*. Each element is added to the internal to the internal memory *dataframe*. The internal memory holds the last 672 elements, since this number of elements is needed to produce a forecast. Afterwards, it checks if there are enough entries in the memory *dataframe* for a new forecast. If there are, it normalizes the values using the formula described in Section 3.1.4. Then the normalized values are converted to a NumPy array and the array is put in *TransformationQueue*.

The *Forecast* filter loads a neural network from a file in the start using Keras. It is started with the two queues: *TransformationQueue* and *WriterQueue*. It takes an array out of *TransformationQueue* and expands its dimension by one. This has to be done since the neural network expects input data to be in batches. Afterwards, it calls the *predict*-function from Keras with the array as input. This yields a new forecast that is send to the writer via putting it in *WriterQueue*.

The *Writer* filter constructs a Kafka *producer*. This *producer* is then connected to the Kafka server defined in the configurations. The writer has the queue *WriterQueue*. From this queue it takes an array containing a forecast. Afterwards it is denormalizes it using the formula described in Section 3.3. With the denormalized forecasts, a message is build that holds a dict containing the denormalized forecasts at the key *forecasts* as a list. After construction, this message is published in the Kafka topic *forecast* using the *producer*.

Chapter 6

Evaluation

This chapter contains the evaluation of different multivariate model variations and usefulness of the external features in model variations. Additionally, it contains a discussion of the results and possible threats to the validity are discussed.

6.1 Methodology and Setup

Kaggle[1] is an online machine learning competition organizator. It offers the possibility to run notebooks on a cloud computing infrastructure. Additionally, there are a lot of data sets and tutorials freely available. The Kaggle infrastructure[2] is used to train and evaluate the different model variations. It allows parallel computing, since multiple variations could be trained and evaluated in different notebooks[3] at the same time.

In this evaluation the results of the training and the evaluations of the model variations of Chapter 4 are shown. The training and evaluation of the model variations is executed as described in Chapter 3. For most of the model variations, training and evaluation are executed only once. Since the shuffle mode in the training generator is used, the reproduction of exactly the same result is unlikely. The model results are compared to each other and if the difference between their results was too high, the compared models were run multiple times again. Afterwards the results that had the smallest difference to the average are considered.

As mentioned in Chapter 2, the MAE is used as the error metric. The loss values are computed using the error metric on the normalized values. Therefore the loss values represent the relative error. Section 6.1 shows how the absolute error is calculated from the relative error. Table 6.1 provides an overview of which absolute error corresponds to which relative error for each evaluated machine. The absolute error of the outside light is of minor significance compared to the other two errors.

$$loss \times std_{machine} = error_{inwatt}$$

[1] https://www.Kaggle.com/
[2] https://www.Kaggle.com/docs/notebooks#technical-specifications
[3] https://jupyter.org/

Table 6.1. The error values for each power consumer in watt. A loss value of 1.0 corresponds to the standard deviation.

Loss Values	Error of Outside Lighting	Error of Air Compressor	Error of Air Conditioner
0.1	315	2156	3365
0.2	630	4311	6731
0.3	945	6467	10096
0.4	1260	8622	13461
0.5	1575	10778	16827
0.6	1890	12933	20192
0.7	2205	15089	23558
0.8	2520	17244	26923
0.9	2835	19400	30288
1.0	3150	21555	33654

6.2 Definition of a Baseline

In order to have a comparison to the neural network models, we evaluate a naive approach. This naive approach serves as baseline. The naive approach described in this section uses past values as predictions. For a prediction, the values from different time periods in the past are used. For example, the values from exactly 24 hours ago. This approach is rather naive, since it can not capture dependencies between the different features. It is also computed on the normalized test data set. The results using this approach and the MAE are shown in Table 6.2.

Table 6.2. Naive Approach

Time	Outside Lighting	Air Compressor	Air Conditioner
1h	0.1801	0.3150	0.1336
2h	0.3515	0.4493	0.1925
8h	1.3042	0.8794	0.4528
24h	0.0467	0.8842	0.3428

6.3 Evaluation of the Model Variations

In this section, results of the model variations for the three power consumers (see Section 4.1) containing all features are shown. Therefore, for each of the three machines, the results of multivariate model variations using the single step and the multi step forecasting approach (see Section 2.2) are shown. For every machine, we provide two tables: One containing

the values of the single step version of the models and one contains the values for the multi step version of the model. This allows to compare both types of models. To allow quick comparisons between the results, we also provide two bar graphs for every table. These bar graphs contain the loss values of the evaluation of the model variants on the test set. Additionally for the air compressor and the air conditioning, the results of univariate model variations are shown. Each table contains the loss values from the training, the validation, and the evaluation on the test set.

6.3.1 Multivariate Model Variations for the Air Compressor

The first model variations to be evaluated are the variations for the the air compressor. Section 4.1 describes why this machine is expected to yield the worst results in comparison to the other machines.

The results of the training and evaluation are shown in **??** and Figure 6.2. Surprisingly, the naive approach is better in predicting the power consumption of this machine in the time period of one hour in the future. All variations for this period yielded larger errors on the test set. In the other time periods, the single model variations beat the naive approach in accuracy. In every other forecasting horizon, the model variations using neural networks yielded lower loss values and therefore have a higher accuracy.

In summary, the results of the multi step variations are superior to the results of the single step model variations. For the 24 hour and 8 hour model variations they are significantly better. One cause of this is that the predictions for the shorter time periods are also contained in the predictions of the longer time periods and the predictions for the shorter time periods are more accurate. In comparison to the naive approach, the single step model variations are also more accurate.

The validation loss of all model variations is higher than their training loss. This indicates that all model variations overfitt in general. The possible causes for this are described in Section 6.5 All model variations for this machine have variations that perform better than the naive approach. Therefore there must be some underlying patterns in the power consumption of the air compressor. The neural networks are able to derive this patterns to a certain part and use it for their predictions.

6.3.2 Multivariate Model Variations for the Outside Lighting

The second model variations to be evaluated are the variations that yield predictions for the outside lighting. This consumer consumes power independently from the general power consumption of the production.

Figure 6.3 (a) and Figure 6.4 (a) shows that the first three time periods of the single step variations of this model variations yielded lower loss values than the naive approach. For the one hour forecasting horizon, it is the GRU and the GRU-Drop variant, for two hours it is the Dense variant and for eight hours all variants performed better. In the 24 hour period

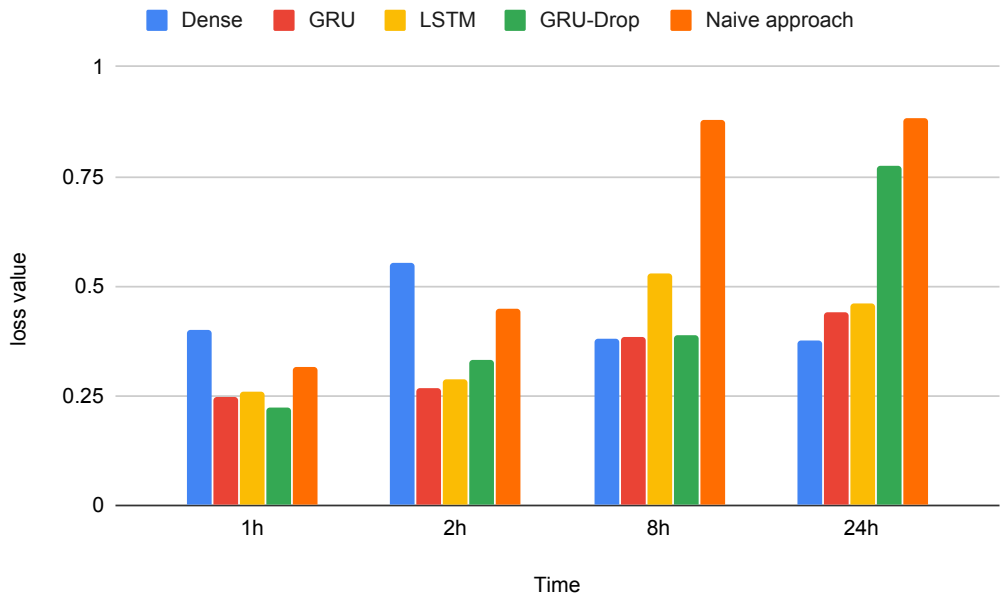
Time	Type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1071	0.2713	0.2714	0.4327
	validation	0.9514	0.5599	0.3760	0.3737
	test	0.4664	0.3829	0.3354	0.3969
2h	train	0.1057	0.2853	0.2672	0.4173
	validation	0.4606	0.3889	0.2952	0.4837
	test	0.3176	0.4393	0.6839	0.3806
8h	train	0.1203	0.3157	0.3079	0.4031
	validation	0.3191	0.3885	0.4379	0.5215
	test	0.4098	0.5426	0.5145	0.3461
24h	train	0.1426	0.3063	0.3047	0.4478
	validation	0.8780	0.5291	0.4670	0.8295
	test	0.4482	0.6983	0.8370	0.6620

(a) Single step variations results

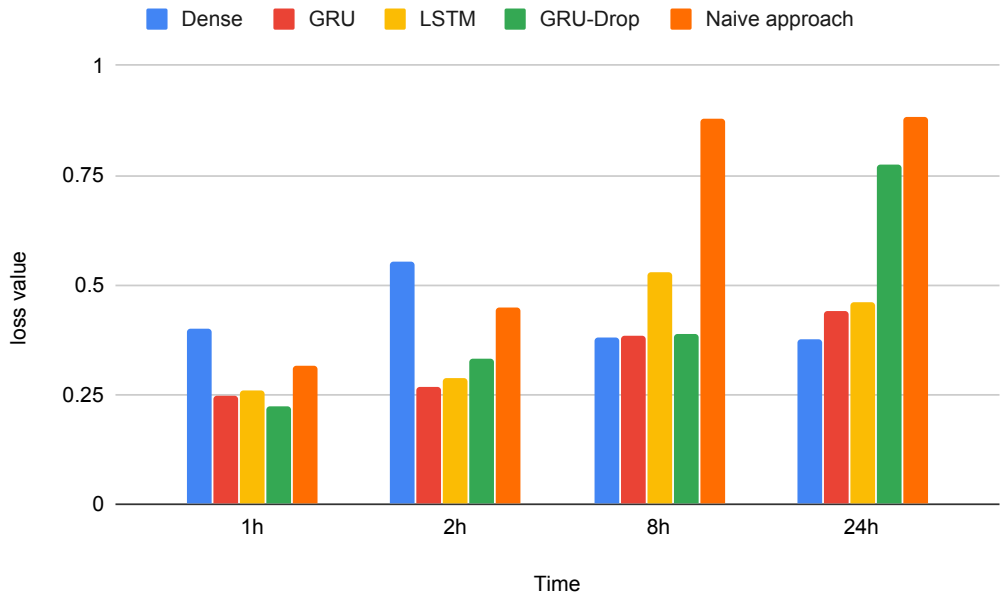
Time	Type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1241	0.2119	0.2061	0.3960
	validation	0.3060	0.2826	0.2926	0.3427
	test	0.3999	0.2481	0.2577	0.2246
2h	train	0.1372	0.2408	0.2387	0.3344
	validation	0.4878	0.4206	0.4188	0.4700
	test	0.5528	0.2682	0.2868	0.3325
8h	train	0.1955	0.2712	0.2646	0.4396
	validation	0.4203	0.3688	0.4737	0.4708
	test	0.3792	0.3847	0.5282	0.3869
24h	train	0.2241	0.3033	0.3113	0.6230
	validation	0.5234	0.5611	0.6083	0.6625
	test	0.3773	0.4411	0.4607	0.7752

(b) Multi step variations results

Figure 6.1. Results of the multivariate model variations for the air compressor: e-druckluft-e2.



(a) Single step



(b) Multi step

Figure 6.2. The test loss results of the evaluation of the model variations of the air compressor on the test set. The x-axis shows the time and the y-axis the loss value.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.0478	0.0897	0.0548	0.0603
	validation	0.1871	0.1224	0.0955	0.1224
	test	0.2105	0.1175	0.2277	0.1175
2h	train	0.0585	0.0756	0.0752	0.2680
	validation	0.0656	0.1481	0.0766	0.2525
	test	0.0970	0.6827	0.4437	0.7061
8h	train	0.0550	0.0793	0.0697	0.0793
	validation	0.2021	0.0021	0.0033	0.0021
	test	0.1265	0.2227	0.2288	0.2227
24h	train	0.0587	0.0664	0.0686	0.1893
	validation	0.4103	0.0226	0.0598	0.1044
	test	0.1922	0.0769	0.0750	0.1116

(a) Single step variations results

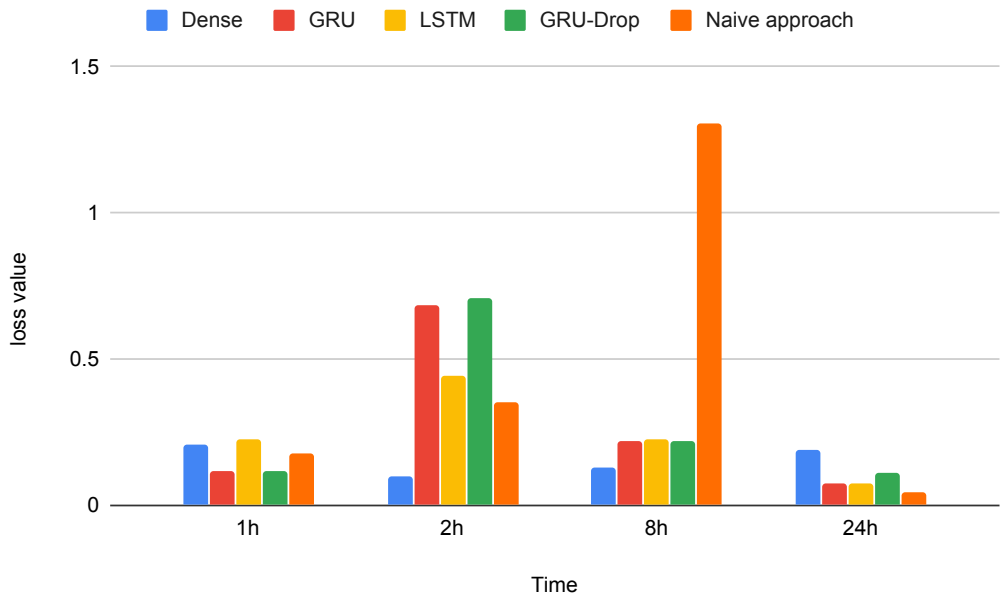
Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.0590	0.0779	0.0534	0.0966
	validation	0.0985	0.0499	0.0329	0.0723
	test	0.2309	0.2377	0.4136	0.2352
2h	train	0.0618	0.0533	0.0584	0.3028
	validation	0.1260	0.0581	0.0935	0.4805
	test	0.2128	0.1104	0.2459	0.4354
8h	train	0.0734	0.5259	0.0804	0.3117
	validation	0.1273	0.4779	0.0812	0.8455
	test	0.1085	0.4391	0.0760	0.1708
24h	train	0.0963	0.0668	0.0932	0.6445
	validation	0.0987	0.0463	0.1005	0.4056
	test	0.1155	0.0685	0.1028	0.4000

(b) Multi step variations results

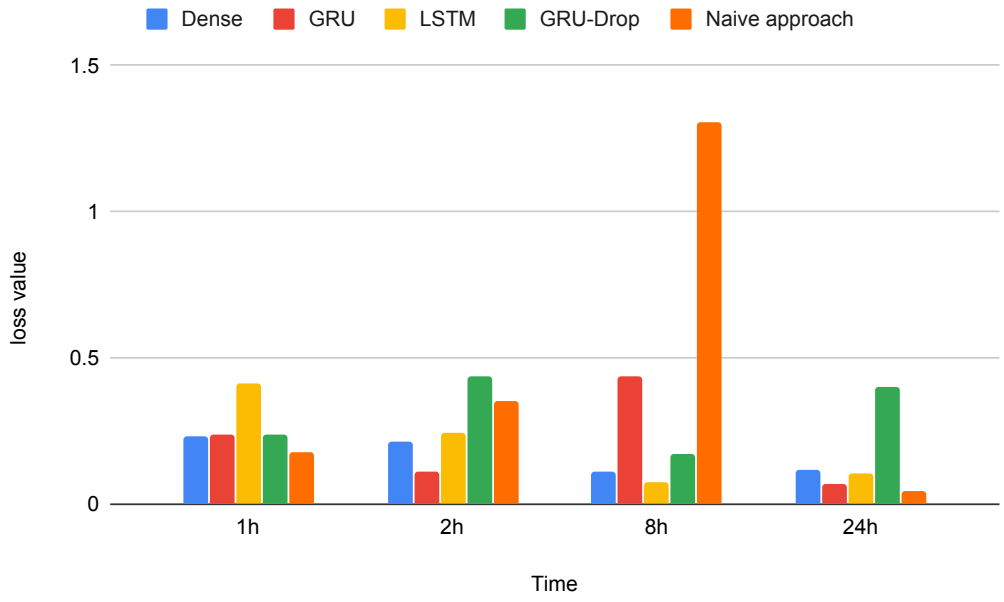
Figure 6.3. Results of the multivariate model variants for the outside lighting: e-aussenbeleuchtung-sud-nea

no model variation was able to perform better. The one hour variants all are overfitting. Their training loss values are higher than their validation loss values.

The results of the multi step model variants are presented in Figure 6.3 b). It shows that changing a model variation from single step to multi step is creating different results. In comparison to the single step variants, the multi step variant could not reach a higher accuracy for the 1 hour time period. The multi step model variations have a higher accuracy at the two hour and the eight hour periods than the naive approach.



(a) Single step



(b) Multi step

Figure 6.4. The test loss results of the evaluation of the model variations of the outside lighting on the test set. The x-axis shows the time and the y-axis the loss value.

The naive approach yielded lower loss values in the 24 hours time period. Therefore the networks failed to derive a pattern that is better than the simple naive approach for this forecasting horizon. Also the networks failed to derive a pattern that represents the naive approach.

6.3.3 Multivariate Model Variations for the Air Conditioner Machine

The results of model variations predicting the power consumption of the air conditioning machine, e-kalte-1 are shown in Figure 6.5 and Figure 6.6.

They show that the training loss values for a model variants are larger than the validation or test loss values. The model variations have a very small loss value in the training set and therefore their accuracy is better than the naive model in all forecasting horizons.

In Figure 6.5 (b), the results of the multi step variants are shown. This table shows that both model variants yielded similar results. In comparison to the single step model variants, the multi step variants have a slightly smaller loss value. Therefore the multi step variants have a better accuracy. In every case except for the dense variants, the training loss is higher than the validation and the test loss.

The results of the accuracy of this variations are surprising. They show a much higher accuracy than expected. The training loss values in both kinds of variations are higher than the validation and test loss values. This indicates that the training set was harder to predict than the validation and the test set. This explains the low test and validation loss values. The neural networks were able to derive the pattern that was in the data for this machine. This enables them to forecast the power consumption data of the air conditioning with a very high accuracy.

6.3.4 Univariate Model Variations for the Air Compressor

In Table 6.3 the results of the univariate single step model variations that predict the power consumption of the air compressor are shown. In Figure 6.7, the model variations are shown next to the multivariate single step variations and the naive approach. There is a total of eight model variations with this kind. The different model variations produced results that are not far apart from each other.

Since the multivariate multi step model variations yielded lower loss values than the multivariate single step model variations, it is expected that the univariate multi step model variations would yield also lower loss values than the univariate single step models.

6.3.5 Univariate Model Variations for the Air Conditioning

In Table 6.4 the results of the univariate model variations for the air condition machine are shown. Figure 6.8 shows the results next to the corresponding multivariate model variations results. The loss values of the univariate variations are, except for the variations

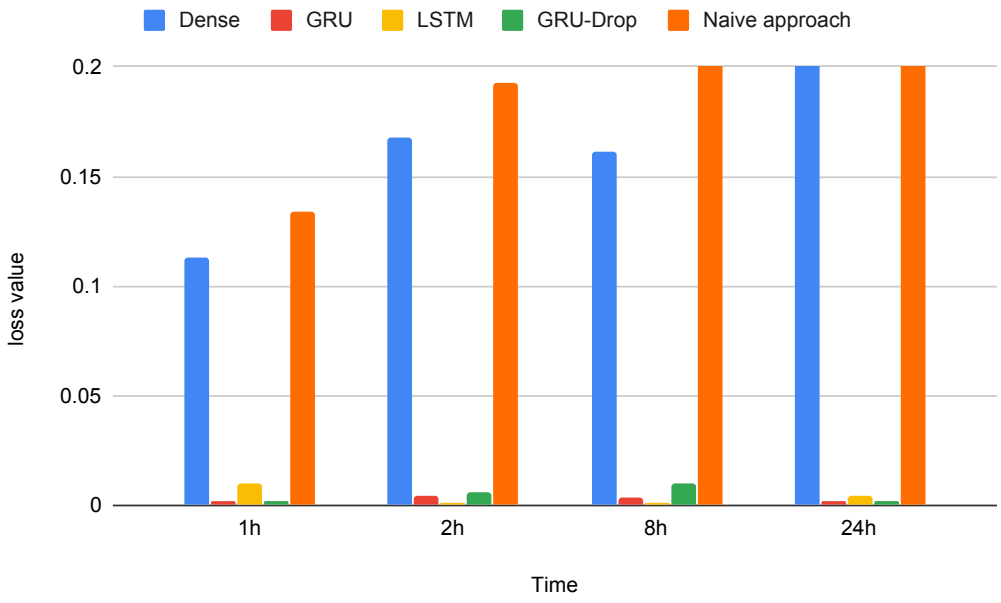
Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.0661	0.1120	0.1116	0.2041
	validation	0.0773	0.0011	0.0089	0.0092
	test	0.1130	0.0022	0.0103	0.0023
2h	train	0.0780	0.1236	0.1262	0.2843
	validation	0.1735	0.0009	0.0110	0.0110
	test	0.1679	0.0042	0.0009	0.0063
8h	train	0.0841	0.1527	0.1441	0.2250
	validation	0.5152	0.0044	0.0041	0.0013
	test	0.1614	0.0035	0.00083	0.00966
24h	train	0.0975	0.1669	0.1680	0.3182
	validation	0.3552	0.0039	0.0029	0.0014
	test	0.2576	0.0019	0.0044	0.0023

(a) Single step variations results

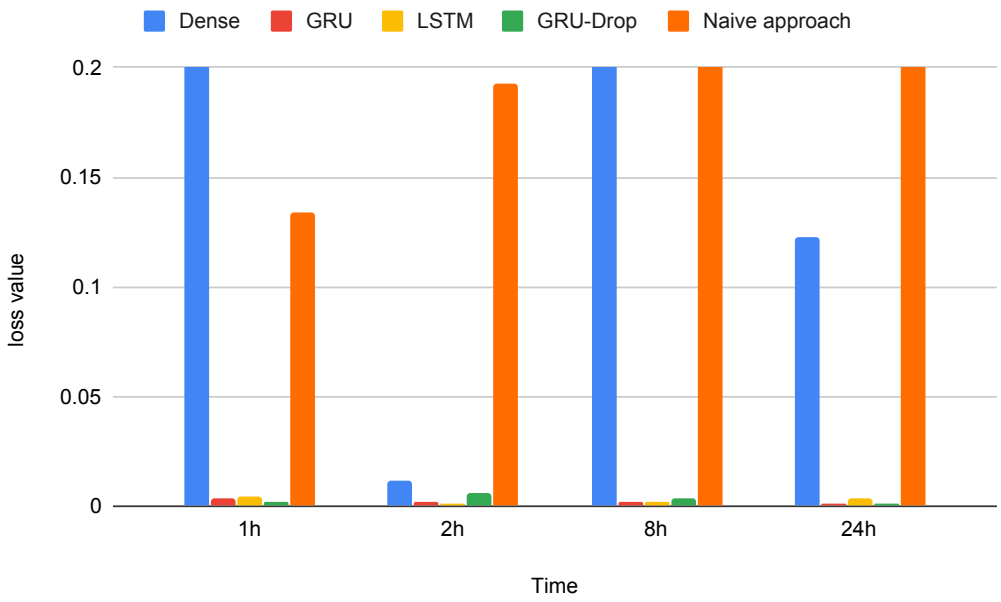
Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.0816	0.0976	0.0967	0.2716
	validation	0.0876	0.0018	0.0051	0.0021
	test	0.2552	0.0037	0.0040	0.0023
2h	train	0.1033	0.1079	0.1075	0.2087
	validation	0.1132	0.0018	0.0015	0.0074
	test	0.0114	0.0021	0.0012	0.0058
8h	train	0.1125	0.1373	0.1373	0.2811
	validation	0.0441	0.0017	0.0017	0.0022
	test	0.2099	0.0016	0.0016	0.0032
24h	train	0.1321	0.1553	0.1685	0.3014
	validation	0.0284	0.0015	0.0030	0.0015
	test	0.1227	0.0013	0.0032	0.0014

(b) Multi step variations results

Figure 6.5. Results of the multivariate model variants for the air conditioning machine: e-kalte-1



(a) Single step



(b) Multi step

Figure 6.6. The test loss results of the evaluation of the model variations of the air conditioning on the test set. The x-axis shows the time and the y-axis the loss value. The y-axis has been cut off at 0.2

Table 6.3. Results of the Univariate Single Step Model variations for m2: e-druckluft-e2. On the left side there is the time from 1h to 24h. On the top are the different Neural Networks (see Section 4.5) of each variation.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1667	0.2907	0.3461	0.3504
	validation	0.3778	0.3405	0.4297	0.3819
	test	0.3984	0.3829	0.5015	0.3851
2h	train	0.1752	0.3151	0.4543	0.3665
	validation	0.5614	0.5005	0.6172	0.3734
	test	0.3579	0.4176	0.6534	0.3701

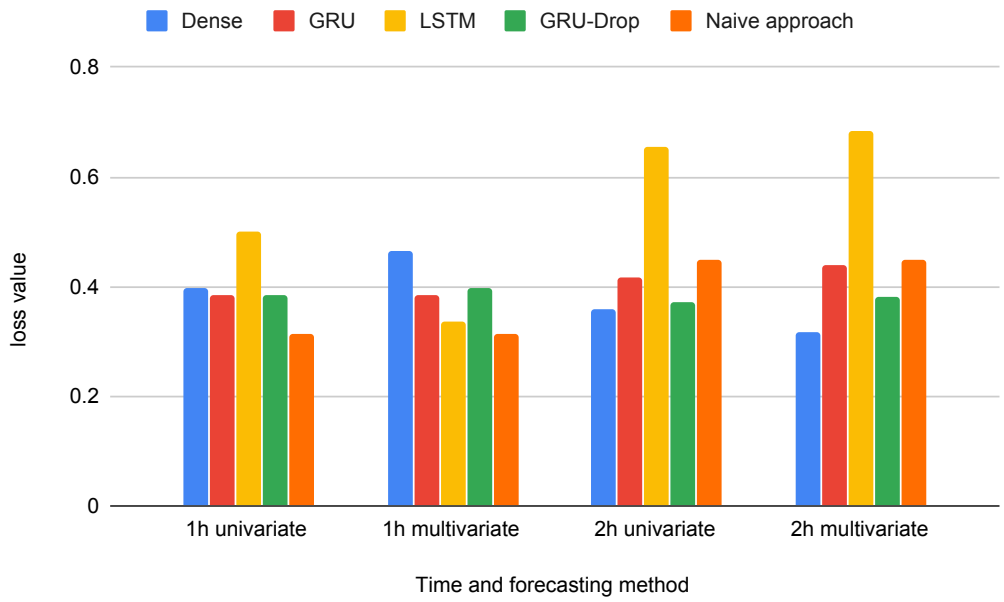


Figure 6.7. Results of the single step univariate model variations next to the results of the single step multivariate model variations of the air compressor.

using the dense neural network, higher than the loss values of the multivariate variations. Therefore the multivariate model variations have an higher accuracy.

Table 6.4. Results of the univariate single step model variations for the air conditioning machine: kälte.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1196	0.1302	0.1308	0.1952
	validation	0.0713	0.0055	0.0081	0.1121
	test	0.0713	0.0055	0.0081	0.1121
2h	train	0.1272	0.1545	0.1609	0.2200
	validation	0.0027	0.0207	0.0116	0.1105
	test	0.0026	0.0207	0.0115	0.1104

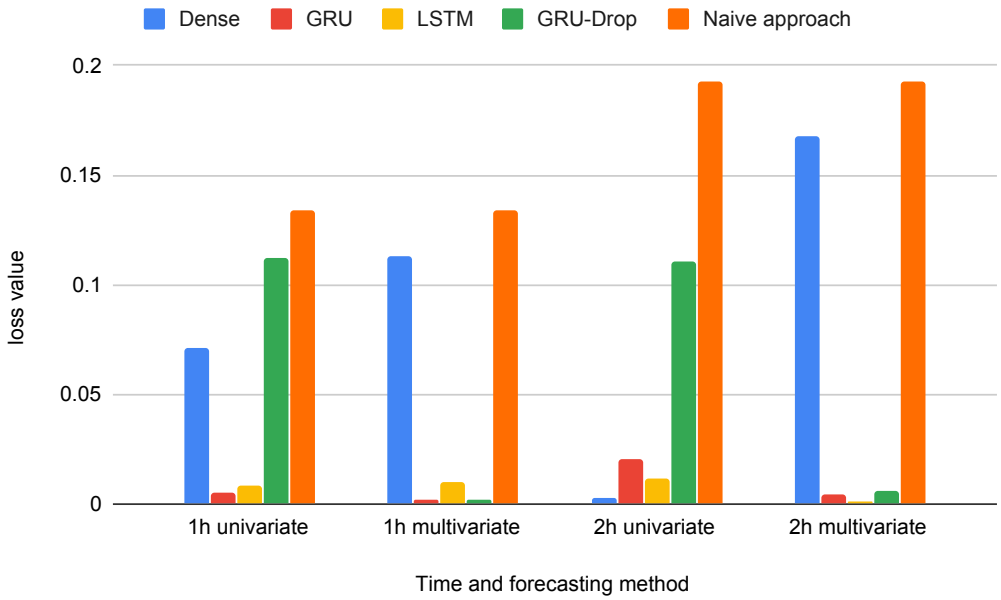


Figure 6.8. Results of the univariate single step model variations next to the corresponding multivariate model results and the results of the baseline

6.3.6 Conclusions

One result is, that no neural network was the best for all machines and forecasting horizons. In most of the cases, the GRU or the LSTM neural networks have the highest accuracy, however in some cases other neural networks or the baseline yielded a lower loss value. For forecasting the power consumption of the air compressor, the GRU or the LSTM neural network would be the best choice in terms of accuracy. The baseline would be the best

choice for the outside light, and for the air conditioner the GRU or the LSTM neural network would be the best choice.

Also between single step and multi step, there is no approach that always yielded better results. In most models, the multi step model variations yielded slightly lower loss values. The multi step forecasts of larger forecasting horizons contain the forecasts of the shorter forecasting periods. Since in the shorter forecasting horizons, most of the model variations performed better, this could sightly lowered the loss of the multivariate model variations. For forecasting the power consumption of the air compressor the multi step model variations would be the better choice. For the outside light it depends on the forecasting horizon and for the air conditioning, the multi step model would be the better choice.

In terms of the forecasting horizons, there is also no forecasting horizons that was always the best. Most of the time, the model variations using a shorter foresting horizon yielded a lower loss value and thus they have a higher accuracy. However, for the air conditioner, this is not true. The loss values for the 24 hour forecasting horizons are lower than the loss values for the other forecasting horizons.

In the comparison between the univariate and the multivariate mode variations, the dense neural network is performing better on the univariate model variations. This is the case since the dense neural network can not capture complex dependencies. Therefore it can not distinguish which of the features of the input is important for the prediction. As a result, the accuracy of the dense neural network gets worse the more input is provided. For the other neural networks, the difference between the univariate and the multivariate was very small. In the most cases, the multivariate model variations yielded a lower loss value, so we come to the conclusion their accuracy is higher in most cases. This is why we conclude, that the multivariate forecasting method should be used for forecasting models.

The expectations for the model accuracy for the air compressor were partly right . In terms of the absolute error the forecasting models for this consumer yielded the lowest accuracy. In terms of the relative error to the baseline, the models for the outside light yielded the lowest accuracy, except of the 8 hour forecasting horizon. The expectations for the model accuracy for the outside lighting were partly right. The accuracy of the model variations forecasting power consumption of the outside lighting is very high, however the baseline has an even higher accuracy. The expectations for the air conditioning were wrong. It is the power consumer that can be forecasted with the highest accuracy.

6.4 Evaluation of External Features

For the evaluation of the external features, model variations without them are designed. The results of these model variations are shown in this section. The usefulness of the external features can be shown by comparing them to model variations that are designed with all features. The model variations used here are multi step models for forecasting the air compressor. We refer the model variations using all features as normal variations in the

following.

6.4.1 Time Features

In Table 6.5, the results of model variations not using the time features (see Section 3.1.2) are shown. Figure 6.9 shows the results side by side with the results of the normal variations and the baseline. The model variations not using the time features reach a better accuracy than the naive approach. The loss values for the 1 hour variations is higher than the loss values in the normal variations (see Figure 6.1). Therefore the time feature is helpful for the 1 hour time period. In the 2 hour time period, the Dense variations has a higher accuracy than the normal Dense and GRU-Drop variations. The other two networks show a higher accuracy in the normal variations. All of the networks are overfitting since the train loss of every model is lower than the corresponding validation loss.

Table 6.5. Results of Model variations for m2: e-druckluft-e2 without the time based features. On the left side there is the time period. On the top are the different neural networks (see Section 4.5) of each variation.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1112	0.2264	0.2218	0.3128
	validation	0.5083	0.3217	0.2952	0.2486
	test	0.2746	0.2981	0.2558	0.2586
2h	train	0.1356	0.2791	0.2809	0.4025
	validation	0.3959	0.5411	0.3958	0.4700
	test	0.3047	0.3304	0.3010	0.2453

6.4.2 Public Holiday Features

In Table 6.6 the results of the model variants not using the external holiday features (see Section 4.4.1) are shown. Figure 6.10 shows the results side by side with the results of the normal variations and the baseline. In the 1 hour period, the Dense and the GRU network variants have a higher accuracy than the normal variants. The holiday features do not increase the accuracy of the model variants with the 2 hour forecasting horizons. The test loss values of these variants are lower than or nearly equal to the corresponding normal variants. Therefore, the holiday feature is not helpful in this time period. The training loss for most of this model variants is lower than the validation loss and therefore most of these model variants are also overfitting.

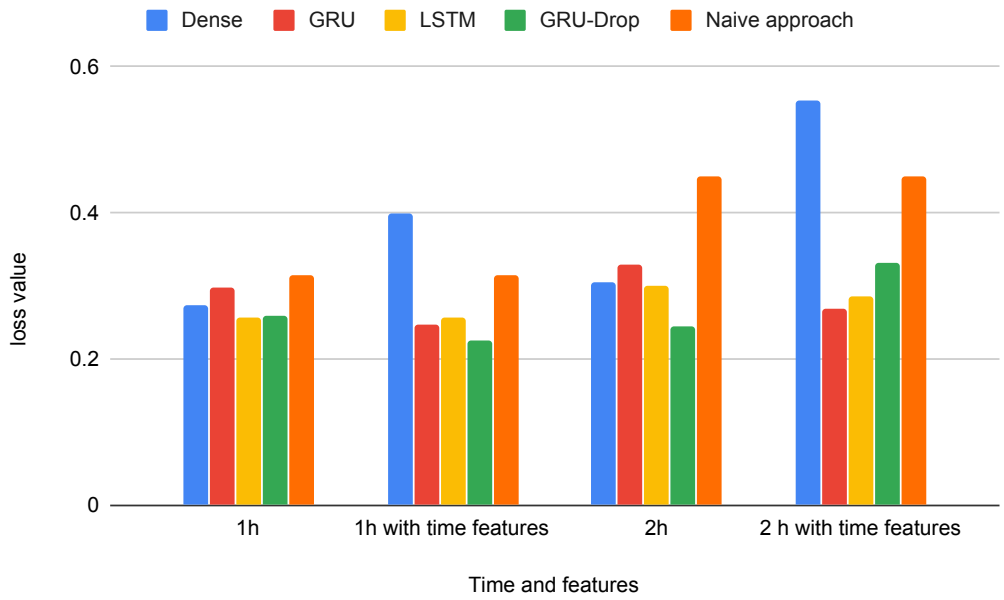


Figure 6.9. The results of the baseline and of the model variations not using the time features side by side with the model variations using them.

Table 6.6. Results of Model variations for m2: e-druckluft-e2 without the holiday features.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1107	0.2052	0.2091	0.3887
	validation	0.3410	0.2284	0.3659	0.2443
	test	0.3344	0.2271	0.2675	0.2251
2h	train	0.1375	0.2502	0.2401	0.4230
	validation	0.4125	0.2310	0.2770	0.4611
	test	0.2027	0.2069	0.2895	0.2693

6.4.3 Temperature Feature

The model variants not using the external temperature feature (see Section 4.4.2) yielded the results shown in Table 6.7. In Figure 6.11, the results are also shown next to the result of the normal variants and the baseline. In the 1 hour period the model variants yielded better accuracy than the normal model variations. The only exception is the GRU-Drop model. In the 2 hour period, the Dense and GRU from the variations without temperature have lower test loss values than the normal variations and thus they have a higher accuracy.

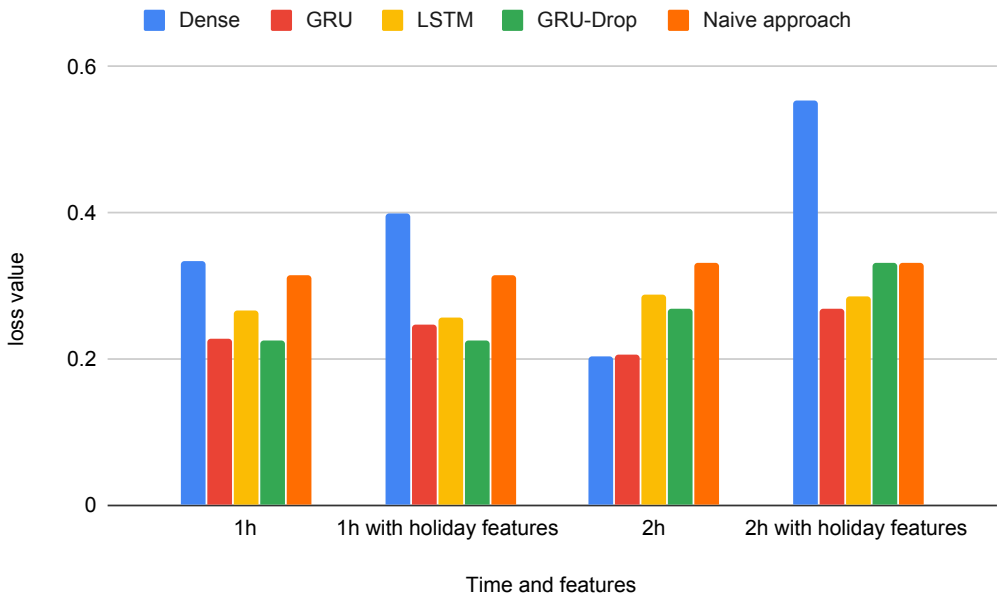


Figure 6.10. The results of the baseline and of the model variations not using the holiday features side by side with the model variations using them.

The other two models were better in the normal variations. Using the temperature feature the difference in the loss value is only large for the Dense neural network. For the other networks, the temperature let them yield a slightly lower loss for the 1 hour and a slightly higher loss for the 2 hour forecasting horizon. All of the model variations, except the 1 hour GRU-Drop, are also overfitting since their train loss is lower than their validation loss.

Table 6.7. The results of the baseline and of model variations for the air compressor: e-druckluft-e2 without the temperature feature. On the left side there is the time from 1h to 24h. On the top are the different Neural Networks (see Section 4.5) of each variation.

Time	type of loss	Dense	GRU	LSTM	GRU-Drop
1h	train	0.1106	0.2162	0.2128	0.3913
	validation	0.3576	0.3403	0.2473	0.3304
	test	0.2880	0.2944	0.2315	0.2497
2h	train	0.1300	0.2456	0.2451	0.3897
	validation	0.7009	0.5573	0.3182	0.3713
	test	0.3032	0.2524	0.3006	0.1964

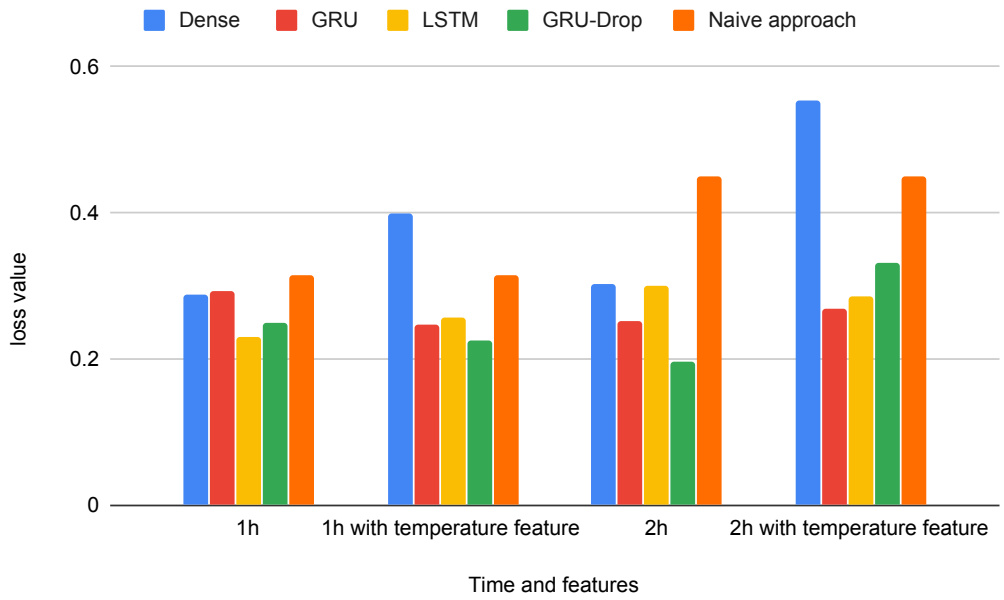


Figure 6.11. The results of the model variations not using the temperature feature side by side with the model variations using them.

6.4.4 Conclusions

The model variations using the Dense neural network yielded higher loss values in the multivariate model variations. As stated in Section 6.3.6, this could be due to the Dense neural network not capturing complex dependencies as well as the other used neural networks. For this reason the Dense network is excluded from the following conclusions.

The time features have proven to slightly increase the accuracy of the model variants. Therefore it is a good choice to use this feature in model variations. The use of external holiday features has not improved the accuracy of the model variations. Opposing to our expectations, the accuracy got slightly worse. This could be put down to the fact that the *Kieler Nachrichten Druckzentrum* do not only print newspaper issues [4]. A few other orders could be scheduled the day before the public holiday and thus prevented the neural networks from generalizing the context described in Section 4.4.1. Therefore it is not a good choice to use these features in models to predict the power consumption of the air compressor. The use of the external temperature feature has not improved the accuracy of the models variations using the 2 hour time period forecasting horizon. In the 1 hour time period forecasting model variations, the use of the external temperature feature slightly

[4]https://kn-druckzentrum.de/unsere-produkte/unsere-produkte-druck/up-druck-akzidenz/

increased the accuracy. So we can not conclude that the usage of the temperature feature is helpful for all model variations of the air compressor.

6.5 Discussion of Overfitting

There are multiple reasons why some of the model variants are the overfitting. One of the reasons could be that the data set with roughly 160000 samples is not very large. Additionally, the data set contains some outliners and errors which are fixed by an interpolation described in Section 3.1.1. Perhaps replacing the error values by sequences from the past would reduce overfitting. All model variants were trained for 20 epochs and not stopped when they reached their optimal loss values. It could also be that they did not even reached their optimal loss value. Another reason for overfitting could be an unfortunate cut between the sets. For example, the split into train, validation and test set influenced the model variants that forecast the power consumption of the air conditioning. The air conditioning was turned off in the time period that is contained in the validation set. Furthermore the neural networks could be too complex for the data set. If this was the reason, the overfitting could be stopped by reducing the model complexity. This process is called regularization [Chollet 2017].

6.6 Threats to Validity

There are a few possible threats to the validity of the results. The first is about the used shuffling option during the training of the neural networks. This leads to a random distribution of all samples in the training set in the batches. Therefore the neural network is trained with another sequence of samples if the training is done again. This leads to different adjustments and generalizations of the network and thus to other accuracy values. Since the networks are trained for 20 epochs, the difference will not be very high but existent. Moreover, the training and evaluation of most of the models is only executed once as described in Section 6.1 So they might reach reach a different accuracy if trained again.

The next threat is the execution of the code on external hardware. Since we have no insight in the cloud servers of Kaggle, we cannot guarantee that the results are not corrupted by the platform.

Chapter 7

Related Work

We present related work in this Chapter. It is classified into the following topics: energy load forecasting, power consumption forecasting, and Titan related work.

7.1 Energy Load Forecasting

Jian Zheng et al. [2017] shows that LSTM based neural networks are capable of forecasting electric load in Smart Grids. There, the univariate forecasting approach is used. The forecasting using LSTM based recurrent neural networks is compared to Seasonal Autoregressive Integrated Moving Average (SARIMA), a feed forward neural network with a single hidden layer and lagged inputs (NNETAR), Support Vector Regression (SVR), and a nonlinear autoregressive neural network model with exogenous inputs (NARX). The approaches were first tested on a airline passengers data set [Faraway and Chatfield 1998] and afterwards on an electric load data set that was collected in a engineering school building. The Root-Mean-Squared-Error (RMSE) and the Mean-Average-Percentage-Error (MAPE) are used for the evaluation. The LSTM-based neural network outperforms the other approaches.

Nataraja et al. [2012] shows different models for short term load forecasting. This models are an Autoregressive, an Autoregressive Moving Average, and an Autoregressive Integrated Moving Average model. The models have been applied on an energy demand pattern. The ARIMA model outperformed the others. In contrast to this thesis, the used methods for the forecasting are statistical approaches. Furthermore, the data is an energy demand pattern and not a power consumption data set.

Din and Marnerides [2017] shows forecasting short term power loads deep neural networks. The data used for this forecasting is taken from a report of the electricity consumption in New England[1]. For this reason Feed-Forward Deep Neural Networks (FF-DNN) and Recurrent Deep Neural Networks (R-DNN) are used. In the paper it is shown that with the use of time-frequency feature analysis and DNN together, a high forecasting accuracy is obtainable. The timefrequency feature analysis covers features like weather, time, holidays, and lagged electricity loads. Temperature, time and holidays are also used in this thesis. In contrast to this thesis, the forecasts are for the future power loads of a power grid.

[1] https://www.iso-ne.com/isoexpress/web/reports/load-and-demand

7.2 Power Consumption Forecasting

Chujai et al. [2013] shows the analyzing of time series data containing the electric power consumption of individual households by ARMA and ARIMA. The following forecasting periods were used: daily, weekly, monthly, and quarterly. RMSE was used for the analysis. The results showed that the ARIMA model performs better forecasts for all periods. In the paper, the forecasts also cover the future power consumption. In contrast to the data used in this thesis, the future power consumption contains the power consumption of whole households and not the power consumption of industrial machines. Statistical methods are used for the forecasting

CHAE et al. [2015] uses artificial neural networks to forecast sub-hourly the electricity usage in commercial buildings. The model produces forecasts for one day ahead in a 15 minutes resolution. In order to develop this model a feature importance analysis was carried out first. Resulting from this analysis the day type indicator, time-of-day, HVAC set temperature schedule, outdoor air dry-bulb temperature, and outdoor humidity were identified to be the most important features for predictions. Afterwards nine different machine learning algorithms were examined and the artificial neural network algorithm was chosen, since it yielded the best results. The used neural network is a multi layer Feed-Forward Network. The developed model can predict with an accuracy of 5% using the Mean-Average-Percentage error. In contrast to this thesis only one-day-ahead forecasts are evaluated. The data used was also from commercial buildings and not from a manufacturing industry. Furthermore, the Random Forest algorithm was used to gather the importance of the features. Contrary, in this thesis the actual impact of the features on the neural networks itself was used to gather the importance of the features. The features used for the forecasts are similar to the ones used in this thesis.

Sas [2006] describes the peak-load management in steel plants. In order to forecast the peak load a mathematical formula for the industrial machines is created. Afterwards, different schedules for the nessesary processes are developed. One of this is identified as optimal solution. In contrast to this thesis, only the load peak problem is tackled. By the mathematical formulation of the process the schedule is inflexible for changes in the production and the approach cannot be used for anomaly detection. Also the problem solution is very specialized for the analyzed factory.

7.3 Titan

Hansen [2019] explores an energy status data set from an industrial environment by a visual analysis and a forecasting for the power consumption. The exploration of the HIPE data set [Bischof et al. 2018] is done using the Titan Control Center. First the data is feed in the Titan Control Center and afterwards visually explored and analyzed. For the forecasting the Prophet [Taylor and Letham 2017] framework is used and the usage of neural network is described. The implementation of the ussage of neural networks was not successful.

Prophet uses a statistical method for the forecasting. The implementation of models using neural networks was unsuccessful and therefore we can not compare the results of the model variations of this thesis to the results of Hansen [2019].

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The overall goal of this thesis was to design a power consumption forecasting software component. This overall goal was divided in four subgoals in Section 1.2. To fulfill these goals, we described the design of machine learning models. After designing and implementing different model variants, we designed a forecasting microservice. This microservice produces forecasts for the power consumption of one machine. It could be integrated into Titan. We evaluated the variants of the models in terms of accuracy. In this evaluation, we compare the machine learning model variations to a naive approach. Furthermore, we evaluated different forecasting method and compared their results. This evaluation revealed that in the most of the model variations the GRU and the LSTM neural network performed better than the other. However, this is not the case for all situations and therefore we can conclude that no neural network was the best of all. In the comparison between the forecasting methods, the multi step variants yielded better results in most cases. Furthermore, most of the time the multivariate model variations yielded a lower loss value than the univariate model variations. Therefore the multivariate model variations have a higher accuracy. In terms of the forecasting horizon, there also was no horizon that always yielded the best accuracy. Additionally, we evaluated the different model features with regard to their usefulness for the model accuracy. Most of the time the approach using neural networks yielded better results than a naive approach. Therefore we recommend to tackle the task of forecasting power consumption in manufacturing industries with neural networks.

The training and evaluation of the neural networks was done using python scripts. We provide these scripts in a package [Boguhn 2020]. Additionally, there are the results of the training and evaluation as tables.

8.2 Future Work

In the following different aspects for future work that could yield interesting results are described.

8.2.1 Future Work in Forecasting Power Consumption

In terms of forecasting power consumption, future work could cover aspects such as forecasting the aggregated power consumption or forecasting the difference of the future value and the current value. Also multiple different time periods as input and as forecasting horizon are interesting and can be covered in future work.

The forecasting of power consumption in general allows future work in terms of monitoring and anomaly detection. The forecasting values can be used to detect machine malfunctions or other anomalies. All in all, future work could cover anomaly detection by using neural networks for forecasts.

8.2.2 Future Work in Using Neural Networks for Power Consumption

In terms of forecasting power consumption using neural networks there are a lot of promising aspects. Future work could include the evaluation of the models designed in this thesis on different data. Different sizes of the neural networks used for the forecasting also offers the opportunity for future work. The designed models could be further optimized for specialized needs in future work. Additionally, they could be trained and evaluated using different optimizer and error criteria. The neural networks could also be modified to use different activation functions. Different outputs of the models, for example a forecast for all machines for the next 15 minutes as forecast could be interesting. The results of this would be helpful to design models with a higher forecasting accuracy.

Most of the model variations had the problem of overfitting. Therefore, we conclude, that for a concrete forecasting problem, multiple forecasting model variations should be designed and evaluated in their accuracy until the needed accuracy is reached. Therefore future work could include additional techniques against overfitting and their applications on neural networks forecasting power consumption.

Bibliography

[Abadi et al. 2016] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: a system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pages 265–283. (Cited on pages 8, 9)

[Albadi and El-Saadany 2008] M. Albadi and E. El-Saadany. A summary of demand response in electricity markets. *Electric Power Systems Research* 78 (Nov. 2008), pages 1989–1996. (Cited on page 1)

[Balalaie et al. 2016] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables DevOps: migration to a cloud-native architecture. *IEEE Software* 33.3 (May 2016), pages 42–52. (Cited on page 10)

[Bischof et al. 2018] S. Bischof, H. Trittenbach, M. Vollmer, D. Werle, T. Blank, and K. Böhm. Hipe: an energy-status-data set from industrial production. In: June 2018, pages 599–603. (Cited on page 50)

[Boguhn 2020] L. Boguhn. *Thesis Artifacts for: Forecasting Power Consumption of Manufacturing Industries Using Neural Networks*. Mar. 2020. (Cited on page 53)

[CHAE et al. 2015] Y. CHAE, R. Horesh, Y. Hwang, and Y. Lee. Artificial neural network model for forecasting sub-hourly electricity usage in commercial buildings. *Energy and Buildings* 111 (Nov. 2015). (Cited on page 50)

[Chollet 2017] F. Chollet. *Deep learning with Python*. 1st. Greenwich, CT, USA: Manning Publications Co., 2017. (Cited on pages 5–9, 19, 48)

[Chujai et al. 2013] P. Chujai, N. Kerdprasop, and K. Kerdprasop. Time series analysis of household electric consumption with arima and arma models. *Lecture Notes in Engineering and Computer Science* 2202 (Mar. 2013), pages 295–300. (Cited on page 50)

[Din and Marnerides 2017] G. M. U. Din and A. K. Marnerides. Short term power load forecasting using deep neural networks. In: *2017 International Conference on Computing, Networking and Communications (ICNC)*. 2017, pages 594–598. (Cited on pages 5, 49)

[Ebert et al. 2016] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. DevOps. *IEEE Software* 33.3 (May 2016), pages 94–100. (Cited on pages 9, 10)

[Faraway and Chatfield 1998] J. Faraway and C. Chatfield. Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 47.2 (1998), pages 231–250. (Cited on page 49)

[Gron 2017] A. Gron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 1st. O'Reilly Media, Inc., 2017. (Cited on pages 5–7, 9)

[Hansen 2019] A. Hansen. Exploring an energy-status-data set from industrial production. Bachelor thesis. Kiel University, Sept. 2019. (Cited on pages 50, 51)

[Hasselbring and Steinacker 2017] W. Hasselbring and G. Steinacker. Microservice architectures for scalability, agility and reliability in e-commerce. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. Apr. 2017, pages 243–246. (Cited on pages 9, 10)

[Hasselbring 2016] W. Hasselbring. Microservices for scalability: keynote talk abstract. In: *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ICPE '16. Delft, The Netherlands: ACM, 2016, pages 133–134. (Cited on pages 3, 9, 10)

[Hasselbring et al. 2019] W. Hasselbring, S. Henning, B. Latte, A. Mobius, T. Richter, S. Schalk, and M. Wojcieszak. Industrial DevOps. In: Mar. 2019, pages 123–126. (Cited on pages 2, 10)

[Haykin 1998] S. Haykin. *Neural networks: a comprehensive foundation*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. (Cited on pages 2, 5)

[Henning and Hasselbring 2019] S. Henning and W. Hasselbring. Scalable and reliable multi-dimensional aggregation of sensor data streams. In: *2019 IEEE International Conference on Big Data (Big Data)*. Dec. 2019, pages 3512–3517. (Cited on pages 2, 27, 29)

[Henning et al. 2019] S. Henning, W. Hasselbring, and A. Möbius. A scalable architecture for power consumption monitoring in industrial production environments. In: *2019 IEEE International Conference on Fog Computing (ICFC)*. June 2019, pages 124–133. (Cited on pages 1, 10–12, 14)

[Henning 2018] S. Henning. Prototype of a scalable monitoring infrastructure for Industrial DevOps. Master thesis. Kiel University, Department of Computer Science, Aug. 2018. (Cited on pages 2, 11)

[Hippert et al. 2001] H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural networks for short-term load forecasting: a review and evaluation. *IEEE Transactions on Power Systems* 16.1 (Feb. 2001), pages 44–55. (Cited on pages 2, 3, 5)

[Hope et al. 2017] T. Hope, Y. S. Resheff, and I. Lieder. *Learning tensorflow: a guide to building deep learning systems*. 1st. O'Reilly Media, Inc., 2017. (Cited on page 7)

[Javied et al. 2016] T. Javied, T. Rackow, R. Stankalla, C. Sterk, and J. Franke. A study on electric energy consumption of manufacturing companies in the german industry with the focus on electric drives. *Procedia CIRP* 41 (Dec. 2016), pages 318–322. (Cited on page 1)

[Jian Zheng et al. 2017] Jian Zheng, Cencen Xu, Ziang Zhang, and Xiaohua Li. Electric load forecasting in smart grids using long-short-term-memory based recurrent neural network. In: *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. Mar. 2017, pages 1–6. (Cited on page 49)

[Jozefowicz et al. 2015] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pages 2342–2350. (Cited on page 7)

[Kang et al. 2016] H. Kang, M. Le, and S. Tao. Container and microservice driven design for cloud infrastructure DevOps. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. Apr. 2016, pages 202–211. (Cited on page 9)

[Lasi et al. 2014] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. Industry 4.0. *Business & Information Systems Engineering* 6.4 (Aug. 2014), pages 239–242. (Cited on page 2)

[Li et al. 2017] Z. Li, Y. Wang, and K.-S. Wang. Intelligent predictive maintenance for fault diagnosis and prognosis in machine centers: Industry 4.0 scenario. *Advances in Manufacturing* 5 (Dec. 2017), pages 1–11. (Cited on page 1)

[Mohamed et al. 2019] N. Mohamed, J. Al-Jaroodi, and S. Lazarova-Molnar. Leveraging the capabilities of Industry 4.0 for improving energy efficiency in smart factories. *IEEE Access* 7 (2019), pages 18008–18020. (Cited on page 1)

[Morrison 2010] J. P. Morrison. *Flow-based programming, 2nd edition: a new approach to application development*. Scotts Valley, CA: CreateSpace, 2010. (Cited on page 10)

[Narkhede et al. 2017] N. Narkhede, G. Shapira, and T. Palino. *Kafka: the definitive guide real-time data and stream processing at scale*. 1st. O'Reilly Media, Inc., 2017. (Cited on page 14)

[Nataraja et al. 2012] C. Nataraja, M. Gorwar, G. Shilpa, and S. Harsha J. Short term load forecasting using time series analysis: a case study for karnataka, india. *International Journal of Engineering Science and Innovative Technology* 1 (Jan. 2012), pages 45–53. (Cited on page 49)

[Park et al. 1991] D. C. Park, M. A. El-Sharkawi, R. J. Marks, L. E. Atlas, and M. J. Damborg. Electric load forecasting using an artificial neural network. *IEEE Transactions on Power Systems* 6.2 (May 1991), pages 442–449. (Cited on page 2)

[Rozenberg et al. 2011] G. Rozenberg, T. Bck, and J. N. Kok. *Handbook of natural computing*. 1st. Springer Publishing Company, Incorporated, 2011. (Cited on page 7)

[Sas 2006] A. Sas. Peak-load management in steel plants. *Applied Energy* 83 (May 2006), pages 413–424. (Cited on page 50)

[Shrouf et al. 2014] F. Shrouf, J. Ordieres, and G. Miragliotta. Smart factories in Industry 4.0: a review of the concept and of energy management approached in production based on the Internet of Things paradigm. In: *2014 IEEE International Conference on Industrial Engineering and Engineering Management*. Dec. 2014, pages 697–701. (Cited on page 1)

[Taylor and Letham 2017] S. Taylor and B. Letham. Forecasting at scale. *The American Statistician* 72 (Sept. 2017). (Cited on page 50)

[Titan Project 2018] Titan Project. *The Industrial DevOps platform for agile process integration and automatision*. Accessed: 2020-03-26. 2018. URL: https://industrial-devops.org. (Cited on page 10)

[Vieira et al. 2017] S. Vieira, W. Pinaya, and A. Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: methods and applications. *Neuroscience & Biobehavioral Reviews* 74 (Jan. 2017). (Cited on pages 6, 7)

[Wang et al. 2015] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein. Building a replicated logging system with Apache Kafka. *Proc. VLDB Endow.* 8.12 (Aug. 2015), pages 1654–1655. (Cited on page 14)

[Willmott and Matsuura 2005] C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research* 30.1 (2005), pages 79–82. (Cited on page 6)

[Wulf et al. 2017] C. Wulf, W. Hasselbring, and J. Ohlemacher. Parallel and generic pipe-and-filter architectures with teetime. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 2017, pages 290–293. (Cited on page 27)

[Zhang et al. 1998] G. Zhang, B. Eddy Patuwo, and M. Y. Hu. Forecasting with artificial neural networks: the state of the art. *International Journal of Forecasting* 14.1 (1998), pages 35–62. (Cited on pages 2, 5)