Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Angewandte Informatik

Bachelorarbeit

von

Sebastian **Steindl**

Integration von Python-basierten Vorhersagemodellen in ein Monitoring-Webdashboard

Integration of Python-based predictive models into a monitoring webdashboard

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Angewandte Informatik

Bachelorarbeit

von

Sebastian **Steindl**

Integration von Python-basierten Vorhersagemodellen in ein Monitoring-Webdashboard

Integration of Python-based predictive models into a monitoring webdashboard

Bearbeitungszeitraum: von 28. Oktober 2019
bis 27. März 2020

1. Prüfer: Prof. Dr. Brunner

2. Prüfer: Prof. Dr.-Ing. Dominikus Heckmann

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Steindl, Sebastian**

Studiengang: **Angewandte Informatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Integration von Python-basierten Vorhersagemodellen in ein Monitoring-Webdashboard

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 12. März 2020

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Steindl, Sebastian**
Studiengang: Angewandte Informatik
Aufgabensteller, Professor: Prof. Dr. Brunner
Durchgeführt in (Firma/Behörde/Hochschule): Capgemini Germany, Nürnberg
Betreuer in Firma/Behörde: Dr. Eldar Sultanow
Ausgabedatum: 28. Oktober 2019 Abgabedatum: 27. März 2020

Titel:

Integration von Python-basierten Vorhersagemodellen in ein Monitoring-Webdashboard

---

Zusammenfassung:

In dieser Arbeit sollte mittels Deep Learning ein Modell erstellt werden, mit dem sich die Auslastung eines Servers in einem Rechenzentrum vorhersagen lässt. Dafür sollte evaluiert werden, welche Modell-Architektur für diese Zeitreihenvorhersage am besten geeignet sind. Dafür wurde zunächst eine theoretische Untersuchung durchgeführt und anschließend ein Vergleich der geeigneten Modell-Architekturen angestellt. Es stellte sich heraus, dass ein *Convolutional Neural Network* (CNN) mit einer *Long Short Term Memory* (LSTM) Schicht die besten Prognosen erzeugt. Es wurde untersucht, wie sich die Hauptkomponentenanalyse zur Dimensionsreduktion auf die Modelle auswirkt. Dies zeigte, dass die Effekte stark von der Architektur abhängen. Die System-Architektur einer Webanwendung wurde so erweitert, dass es möglich ist, neben der historischen Serverauslastung auch Prognosen anzuzeigen. Die Visualisierungen der aufgezeichneten Daten wurden um die Vorhersagen erweitert. Die Arbeit stellt schließlich eine Möglichkeit dar, wie der Lebenszyklus des Prognosemodells in einem produktiven System mit stetiger Auslastungsmessung integriert werden kann.

Schlüsselwörter: Vorhersagemodelle, Monitoring-Webdashboard, Zeitreihenvorhersage, Deep Learning, Serverauslastung.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Ausgangssituation 1
1.2 Zielsetzung 2
1.3 Gliederung der Arbeit 2

2 Einführung in die Zeitreihenanalyse in der Statistik **4**
2.1 Definition einer Zeitreihe 4
2.1.1 Arithmetisches Mittel 4
2.1.2 Erwartungswert 5
2.1.3 Varianz 5
2.1.4 Kovarianz 5
2.2 Stationäre Prozesse 6
2.3 Das Komponentenmodell 7
2.3.1 Trendbereinigung 7
2.3.2 Saisonbereinigung 8
2.4 AR-, MA- und ARMA-Modelle 8
2.4.1 AR(p)-Modelle 9
2.4.2 MA(q)-Modelle 11
2.5 ARMA(p,q)-Modelle 11
2.6 ARIMA(p,d,q)-Modelle 12
2.6.1 Vorgehensweise zur Zeitreihenvorhersage mit ARIMA-Modellen 14

3 Python für Machine-Learning und Data Science **15**
3.1 Relevante Python-Bibliotheken 16
3.1.1 Numpy 16
3.1.2 Pandas 16
3.1.3 statsmodels 16
3.1.4 Matplotlib 16
3.1.5 Tensorflow 16
3.1.6 Keras 17
3.1.7 Scikit-learn 17

4 Der Data Science Prozess **18**
4.1 Fragestellung 19
4.2 Datenbeschaffung (1) 19

4.3 Datenexploration (1) . . . 19
4.4 Datenbeschaffung (2) . . . 19
4.5 Datenexploration (2) . . . 20
4.5.1 Rücksprache mit Datenbankadministrator . . . 22
4.6 Datenbeschaffung (3) . . . 22
4.7 Datenexploration (3) . . . 23
4.7.1 Vereinheitlichung der Timestamps . . . 23
4.7.2 Merkmale als Spalten . . . 24
4.7.3 Keine fehlenden Merkmalswerte . . . 24
4.8 Datenmodellierung . . . 25
4.9 Visualisierung . . . 25

5 Grundlagen für die Prognosenerstellung mit Machine-Learning 27
5.1 Was ist Machine-Learning? . . . 27
5.2 Statistische Methoden oder Machine-Learning? . . . 27
5.3 Machine-Learning-Klassen . . . 28
5.3.1 Supervised Learning . . . 28
5.3.2 Unsupervised Learning . . . 28
5.3.3 Reinforcement Learning . . . 29
5.3.4 Entscheidung für eine Klasse . . . 29
5.4 Künstliche Neuronale Netze . . . 29
5.4.1 Das Perzeptron nach Rosenblatt . . . 30
5.4.2 Aufbau von KNNs . . . 31
5.4.3 Lernen mittels Gradientenabstiegsverfahren . . . 34
5.4.4 Backpropagation . . . 37
5.5 Über- und Unteranpassung eines Künstlichen Neuronalen Netzes . . . 38
5.6 Regularisierung . . . 39
5.6.1 Dropout-Regularisierung . . . 40
5.6.2 Regularisierung durch Early Stopping . . . 40
5.7 Standardisierung . . . 41
5.8 Univariate und multivariate Vorhersagen . . . 42

6 Zeitreihenprognose mit Neuronalen Netzen 43
6.1 Evaluation verschiedener Architekturen Neuronaler Netze . . . 43
6.1.1 Multilayer Perceptron (MLP) . . . 43
6.1.2 Convolutional Neural Network (CNN) . . . 44
6.1.3 Recurrent Neural Network(RNN) . . . 46
6.1.4 Long Short Term Memory (LSTM) . . . 47
6.1.5 Gated Recurrent Unit (GRU) . . . 50
6.2 Datenvorbereitung für multivariate KNNs . . . 52
6.2.1 Featuregenerierung aus den Zeitstempeln . . . 52
6.2.2 Hauptkomponentenanalyse (PCA) . . . 52
6.2.3 Ablauf der Datenvorbereitung . . . 53
6.3 Implementierung der KNNs . . . 54
6.3.1 Grundlegende Designentscheidungen . . . 55
6.3.2 Implementierung des MLP . . . 57

6.3.3 Implementierung des CNN . . . 57
6.3.4 Implementierung des LSTM- und GRU-Netzes . . . 59
6.4 Modelloptimierung . . . 59
6.4.1 Kreuzvalidierung . . . 60
6.4.2 Rastersuche . . . 61
6.4.3 Ergebnisse der Hyperparameteroptimierung . . . 62
6.5 Trainings-, Validierungs- und Testdaten . . . 62
6.6 Modellbewertung . . . 63
6.6.1 Möglichkeiten zur Fehlerbestimmung . . . 63
6.6.2 Fehlerauswertung der erstellten Modelle . . . 65
6.6.3 Interpretation der Fehlerauswertung . . . 66
6.6.4 Entscheidung für ein Modell . . . 67
6.6.5 Modell für zweite Serverinstanz . . . 68

7 Möglichkeiten zur Integration der Prognosen in die Webanwendung 72
7.1 Durchführen des Machine-Learning im Browser . . . 72
7.1.1 Tensorflow.js . . . 72
7.1.2 WebAssembly . . . 73
7.1.3 Iodide . . . 73
7.1.4 Pyodide . . . 74
7.2 Bereitstellung der Prognosewerte . . . 74
7.3 Entscheidung für einen Ansatz . . . 75

8 Umsetzung des gewählten Ansatzes 77
8.1 Die Webanwendung DC-Cubes . . . 77
8.2 Verwendete Technologien . . . 77
8.2.1 React . . . 78
8.2.2 D3.js . . . 78
8.2.3 three.js . . . 79
8.2.4 Solr . . . 79
8.2.5 Node.js . . . 80
8.3 Backend-Architektur . . . 80
8.4 Konzept zur Bereitstellung der Prognosewerte . . . 80
8.5 Visualisierung der Prognosen in der Webanwendung . . . 80
8.5.1 Ausgangssituation . . . 81
8.5.2 Erweiterung um Prognose . . . 81
8.6 Technische Umsetzung . . . 82
8.6.1 Speichern der historischen Daten . . . 82
8.6.2 Erzeugen und Speichern der Prognosen . . . 82
8.6.3 Laden der Prognose in die Webanwendung . . . 83
8.6.4 Neue Prognosen erstellen . . . 83
8.6.5 Anzeigen der Prognose aktivieren . . . 84
8.6.6 Anzeigen der Prognose im 2d-Chart . . . 84
8.6.7 Anzeigen der Prognose im 3d-Chart . . . 85
8.6.8 Merge-Core . . . 85

9 Zusammenfassung und Ausblick **86**
9.1 Zusammenfassung dieser Arbeit 86
9.2 Lebenszyklus der Prognosemodelle 87
9.3 Ausblick 87

Literaturverzeichnis **89**

Abbildungsverzeichnis **97**

Tabellenverzeichnis **99**

Listingverzeichnis **100**

A Abbildungen **101**

B Tabellen **107**

C Listings **109**

D Inhaltsverzeichnis der CD **115**

Glossar

ACF: Autocorrelationfunction. **10**, 11 API: Application Programming Interface. 17, 73, 75
AR: Autoregressive. 9, 14, 11
ARIMA: Autoregressive integrated moving average. 45, 4, 12, 14
ARMA: Autoregressive moving average. 8, 11, 12, 14, 28
AWS: Amazon Web Services. 75
CNN: Convolutional Neural Network. **43**, 44, 45, 53, 54, 57, 58, 62, 65, 66, 67, 68, 69, 71, 86
CPU: Central processing unit. 4, 19, 22, 25, 69, 73, 77
CRUD: Create, Read, Update, Delete. 79
CSS: Cascading Style Sheets. 73
DOM: Document Object Model. 78
DSV: Delimiter-separated values. 16, 20
GPU: Graphics processing unit. 37, 73
GRU: Gated recurrent unit. **50**, 51, 53, 54, 59, 62, 65, 66, 67
HTML: Hypertext Markup Language. 73, 74, 78
HTTP: Hypertext Transfer Protocol. 82, 83, 84
IT: Informationstechnologie. 1, 2, 25, 77, 87
KNN: Künstliches Neuronales Netz. **29**, 31, 32, 34, 35, 37, 38, 39, 42, 43, 44, 52, 53, 54, 55, 57, 59, 62
LSTM: Long Short Term Memory. 45, **47**, 48, 49, 50, 51, 53, 54, 58, 59, 60, 62, 65, 66, 67, 86
MA: Moving average. 8, **11**, 12, 14
OCR: Optical caracter recognition. 28, 75
PACF: Partial Autocorrelationfunction. **10**
PCA: Principal components analysis. **52**, 53, 54, 65, 82, 86
REST: Representational State Transfer. 80, 83
RNN: Rekursives Neuronales Netz. 38, 45, **46**, 47, 50, 74
SVG: Scalable vector graphics. 78, 79, 84

Kapitel 1

Einleitung

Dieses Kapitel stellt zunächst die Ausgangssituation dar, formuliert dann die daraus entstehende Zielsetzung dieser Arbeit und legt schließlich die Gliederung der Arbeit dar.

1.1 Ausgangssituation

Rechenzentren spielen seit eine zentrale Rolle bei Organisationen und Unternehmen, um diverse Dienste zur Verfügung zu stellen und Daten zu speichern. Mit der in den letzten Jahrzehnten steigenden Bedeutung des Internets und der IT wuchsen auch die Datenmengen und die Bedeutung der Daten nahm zu. Gleichzeitig erhöhten sich damit die Ansprüche an die Netzwerkstrukturen, die diese Daten speichern und zur Verfügung stellen. Diese Server sollen möglichst ununterbrochen erreichbar sein und immer höhere Geschwindigkeiten ermöglichen [42]. Bei der Erreichbarkeit besteht für den reibungslosen Ablauf eines Betriebs meist der Anspruch der sogenannten „Hochverfügbarkeit“. Das heißt nach [80], dass die Server nahezu stetig erreichbar sein und die Ausfallquote weniger als ein Prozent betragen soll. Weiter weißt der Autor darauf hin, dass je größer die Anzahl der verwendeten Geräte ist, desto komplizierter und teurer ist es auch, dieses Ziel zu erreichen. Zudem ist der Betrieb eines Rechenzentrums an sich mit nicht vernachlässigbaren Kosten verbunden. Diese setzten sich hauptsächlich aus den Aufwendungen für die Hardware und den Stromkosten zusammen [36]. Daher ist es auch aus dieser Sicht für Großorganisationen mit großen Rechenzentren von wirtschaftlichem Interesse, die Infrastruktur effizient und mit geringen Ausfallzeiten zu betreiben. Nach [36] wird dies aber nur selten erreicht. Dementsprechend ist die Überwachung und Optimierung von Rechenzentren von großer Bedeutung.
Der für diese Arbeit vorliegende Datensatz stammt von der Bundesagentur für Arbeit. Mit rund 9000 Servern, von denen die Hälfte virtuell ist, die in zwei redundanten Rechenzentren betrieben werden, liegt eine der größten IT-Landschaften Deutschlands vor [3].

1.2 Zielsetzung

Technologien aus den Bereichen der Künstlichen Intelligenz und Data Science können bei mehreren Aspekten eines Rechenzentrums eine Unterstützung sein. Laut [51] konnte Google durch deren Einsatz die Energiekosten für ihre Rechenzentren um 40% senken. Ein durch statistische Vorhersagen angereichertes Monitoring kann außerdem zu erhöhter Effizienz führen und dabei helfen, Ausfallzeiten zu minimieren [1]. Im Zuge dieser Arbeit sollen basierend auf realen Belastungsdaten mittels Machine Learning Prognosen über die zukünftige Auslastung eines Servers getroffen werden. Die Daten stammen dabei aus der IT-Struktur der Bundesagentur für Arbeit. Diese Vorhersagen können als Grundlage zur Optimierung und Verhinderung von Netzwerkausfällen dienen. Dafür ist es nötig, die Ergebnisse zu visualisieren. So können die Prognosen vom Menschen besser verstanden und analysiert werden und es wird eine einfachere Kommunikation ermöglicht [86, S. 1 f.]. Diese Visualisierung soll in eine prototypische Web-Anwendung, die zum Monitoring der oben genannten IT-Struktur konzipiert ist, integriert werden. Dabei soll kenntlich gemacht werden, ob der Benutzer gerade Monitoringdaten der Vergangenheit oder die prognostizierten Werte betrachtet. Nach dem Lesen dieser Arbeit kennt der Leser Grundlagen zur Zeitreihenvorhersage in der Statistik und für die Prognosenerstellung mit tiefen Künstlichen Neuronalen Netzen. Er kennt Architekturen beziehungsweise Arten von Neuronalen Netzen, die sich für die Verarbeitung von Sequenzen, zu denen Zeitreihen gehören, anbieten sowie deren besonderen Eigenschaften, die sie dafür geeignet machen. An einem konkreten Datensatz kann er den Prozess zur Prognosenerstellung nachverfolgen. Schließlich lernt er Leser Möglichkeiten kennen, mit denen man die Zeitreihenvorhersagen in eine Webanwendung integrieren kann. Dem Leser wird eine Implementierung der Integration sowie die verwendete Visualisierung in der Webanwendung vorgestellt.

1.3 Gliederung der Arbeit

Das folgende Kapitel gibt eine Einführung in die Zeitreihenanalyse in der Statistik und stellt eine Modellklasse vor, die für Zeitreihenvorhersagen verwendet werden kann. Anschließend wird in Kapitel 3 erläutert, warum sich Python als Programmiersprache für Machine-Learning und Data Science eignet. Dabei werden einige für diese Bereiche und diese Arbeit relevante Bibliotheken vorgestellt. Daraufhin wird in Kapitel 4 ein Prozess erläutert, der die Vorgehensweise bei einem Data Science Projekt allgemein aber auch in dieser Arbeit, beschreibt. Im darauf folgenden Kapitel werden einige Grundlagen zur Prognosenerstellung mittels Machine-Learning vorgestellt, bevor dann in Kapitel 6 auf konkrete Modell-Architekturen eingegangen wird. In diesem Kapitel wird auch die Hauptkomponentenanalyse und deren Auswirkungen auf die verschiedenen Netz-Architekturen dargestellt. Anschließend wird die Implementierung und Optimierung der Modelle beschrieben. Daraufhin wird das Ergebnis des Trainierens analysiert und darauf basierend eine Modell-Architektur ausgewählt. Für die ausgewählte Architektur werden die Vorhersagemodelle erstellt, deren Integration in das Monitoring-Webdashboard in den Kapiteln 7 und 8 behandelt wird. Dafür werden in Kapitel 7 zunächst zwei grundlegende Ansätze vorgestellt und verglichen. Danach

wird in Kapitel 8 die Umsetzung eines der beiden Ansätze für den konkreten Fall dieser Arbeit beschrieben. Schließlich werden in Kapitel 9 die Ergebnisse dieser Arbeit zusammengefasst und eine Empfehlung für den Lebenszyklus des Prognosemodells sowie ein Ausblick für das Monitoring-Webdashboard gegeben.

Kapitel 2

Einführung in die Zeitreihenanalyse in der Statistik

In diesem Kapitel werden zunächst Verfahren zur Zeitreihenanalyse aus der Statistik behandelt. Zunächst wird definiert, was man unter einer Zeitreihe versteht und es werden einige Kennzahlen eingeführt. Danach werden Zeitreihen theoretischer betrachtet und das Komponentenmodell erläutert. Schließlich wird das Zeitreihenmodell ARIMA, sowie die Modelle, aus denen sich ARIMA zusammensetzt, vorgestellt.

2.1 Definition einer Zeitreihe

Eine Zeitreihe lässt sich definieren als „zeitlich geordnete Beobachtungswerte des jeweils gleichen Sachverhaltes“ [84]. Nach der Definition in [57] kann noch hinzugefügt werden, dass es sich um eine diskrete Sequenz mit endlichem Zeitinterval handelt. Eine Zeitreihe entsteht demnach dadurch, dass man *etwas* in bestimmten, im Idealfall regelmäßigen, Zeitabständen misst. In einem Rechenzentrum gibt es viele Metriken, die nach diesem Prinzip erhoben werden. Misst und speichert man beispielsweise die CPU-Auslastung eines Servers alle n Minuten, entsteht durch diese Diskretisierung eine Zeitreihe. Im Folgenden werden einige wichtige Kennzahlen erklärt.

2.1.1 Arithmetisches Mittel

Das arithmetische Mittel $\bar{x}$ ist die Summe der Zeitreihenwerte x_i dividiert durch deren Anzahl N und stellt somit den mittleren Wert der Zeitreihe dar. Es lässt sich berechnen über

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.1}$$

[43, S. 47].

2.1.2 Erwartungswert

Der Erwartungswert $E(X)$ einer Zufallsvariable X berechnet sich aus den Werten, die X annimmt, x_i, unter Berücksichtigung ihrer Eintrittswahrscheinlichkeit $P(X = x_i)$. Die x_i werden auch Realisationen genannt. Er wird berechnet als

$$E(X) = \mu(x) = \sum_i x_i P(X = x_i) \tag{2.2}$$

[40]. Die Realisationen werden also gewichtet betrachtet. Ein einfaches Beispiel zur Veranschaulichung ist das Werfen eines sechsseitigen Würfels. Jede Augenzahl hat die Wahrscheinlichkeit $P(X = x_i) = \frac{1}{6}$ für $i = 1..6$. Somit ergibt sich für den Erwartungswert der gewürfelten Augenzahl: $E(X) = \sum_{i=1}^{6} i * \frac{1}{6} = 3.5$. Wenn man das arithmetische Mittel der Augenzahlen aus n Würfen bildet, wird man also - bei ausreichend großem n - einen Wert erhalten, der sehr nahe bei 3.5 ist. Man kann den Erwartungswert also als den zu erwartenden Mittelwert betrachten.

2.1.3 Varianz

Die empirische Varianz s^2 ist ein Maß dafür, wie stark die Werte x_i um das arithmetische Mittel schwanken:

$$s^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{2.3}$$

[85, S. 3]. Je geringer die Varianz, desto ähnlicher ist die Kurve einer waagrechten Linie im zweidimensionalen Raum.

Für Zufallsvariablen muss die Wahrscheinlichkeitsverteilung berücksichtigt werden. Man spricht dann nur von Varianz und berechnet diese über den Erwartungswert als

$$Var(X) = \sigma^2 = E\left(X - E(X)^2\right) \tag{2.4}$$

[29, S. 232].

Die Varianz einer Zufallsvariable X gibt demnach den Erwartungswert der Abweichung zwischen X und dem Erwartungswert $E(X)$ an. Wenn also n Werte einer Zufallsvariable X gezogen werden, erwartet man, dass diese eine Abweichung von $Var(X)$ von ihrem Erwartungswert haben.

2.1.4 Kovarianz

Die empirische Kovarianz c misst die Abhängigkeit zweier Werte x_i und y_i:

$$c = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) \tag{2.5}$$

und kann für die Abhängigkeit der τ aufeinander folgendenen Werte einer Zeitreihe eines stationären Prozesses angepasst werden, sodass

$$c = \frac{1}{N} \sum_{t=1}^{N-\tau} (x_t - \bar{x})(x_{t+\tau} - \bar{x}) \tag{2.6}$$

[85, S. 4 f.]. Diese Formel gilt für stationäre Prozesse, da hier $\bar{x}$ konstant ist (siehe Abschnitt 2.2) und so für beide Faktoren verwendet werden kann. Bei großer empirischer Kovarianz und somit Abhängigkeit zwischen den Punkten, bedeutet ein hoher Wert in x_t einen hohen Wert in $x_{t+\tau}$ respektive ein niedriger Wert in x_t auch einen niedrigen Wert in $x_{t+\tau}$.

Analog zur Varianz muss auch für die Kovarianz die Wahrscheinlichkeitsverteilung berücksichtigt werden, wenn man sie für Zufallsvariablen betrachten möchte. Sie ergibt sich für die Zufallsvariablen X und Y als

$$Cov(X, Y) = E\left((X - E(X))(Y - E(Y))\right) \tag{2.7}$$

[29, S. 324].

2.2 Stationäre Prozesse

In der Statistik wird eine Zeitreihe nach [85, S. 90 f.] als eine Realisation eines stochastischen Prozesses gesehen. Ein stochastischer Prozess ist dabei eine Folge von Zufallsvariablen. Die Zeitreihe entsteht dann, indem der zugehörige stochastische Prozess einem eingetretenen, zufälligen Ergebnis eine Zahlenreihe zuordnet [85, S. 90 f.]. Der stochastische Prozess ist also diejenige Vorschrift, die die Zeitreihe erzeugt hat. Somit kann es für jeden stochastischen Prozess mehrere Realisationen, also mehrere Zeitreihen, geben. Da die stochastischen Prozesse aus Zufallsvariablen bestehen, ist im Folgenden mit Varianz und Kovarianz nicht die jeweils empirische Art gemeint, sondern die für Zufallsvariablen nach Gleichung (2.4) beziehungsweise (2.7).
Selbst wenn man den stochastischen Prozess kennen würde, der eine Zeitreihe, die ja nur aus Beobachtungen eines begrenzten Zeitraums besteht, erzeugt hat, wäre die Vorhersage dennoch nicht deterministisch [85, S. 91]. Schließlich handelt es sich um Zufallsvariablen. Allerdings lässt sich der stochastische Prozess zu einer Zeitreihe ohnehin nicht einfach so finden. Selbst wenn man nur die Maßzahlen des Erwartungswertes und der Varianz betrachten möchte, wäre dies schwierig. Bei einer Zeitreihe der Länge N, müssten dafür nach [84, S. 100] N Erwartungswerte sowie N Varianzen geschätzt werden. Wenn man nun festlegt, dass der Erwartungswert und die Varianz im zeitlichen Verlauf gleich sind, reduziert sich der Aufwand von N Erwartungswerten + N Varianzen = $2N$ zu schätzenden Werten auf 1 Erwartungswert + 1 Varianz = 2 zu schätzende Werte [85, S. 100]. Ein stochastischer Prozess, der die Anforderung des konstanten Erwartungswertes erfüllt heißt „mittelwertstationär“ [85, S. 100]. Des Weiteren bezeichnet man einen mittelwertstationären Prozess als „kovarianzstationär“ wenn die Kovarianzen zweier Werte, also deren Ähnlichkeit zueinander, nur von deren zeitlichen Abstand abhängt [85, S. 100]. Mit der Kovarianzstationarität geht einher,

dass auch die Varianz konstant ist. Ist ein stochastischer Prozess sowohl mittelwert- als auch kovarianzstationär, bezeichnet man ihn auch als „schwach stationär“ [85, S. 100]. Bei einem solchen schwach stationären Prozess kennen wir nun für die Zeitreihen gewisse Eigenschaften - den Erwartungswert, die Varianz und Kovarianzen -, die auch gültig sind, wenn die Zeitreihe um s Zeitschritte verschoben wird [85, S. 100]. Dies ist für die Vorhersage der Zeitreihe entscheidend, da eine Vorhersage für s Zeitschritte einem Verschieben um s Einheiten in die Zukunft entspricht.

2.3 Das Komponentenmodell

Einige Besonderheiten von Zeitreihen sind aus dem Alltag bekannt. Betrachtet man beispielsweise die Konjunktur eines Landes, wird man Schwankungen feststellen. Diese Schwankungen können zum Beispiel an der Jahreszeit liegen. Die Konjunktur vieler typischer Urlaubsländer hängt stark vom Tourismus und so auch von der Jahreszeit ab. Die Zeitreihe der Konjunktur besitzt also eine sogenannte Saisonalität. Wenn die gemessenen Werte aber unabhängig von diesen saisonalen Schwankungen auf lange Sicht steigen oder fallen, liegt zusätzlich ein Trend vor. Diese Eigenschaften widersprechen den Forderungen nach konstantem Erwartungswert und konstanter Kovarianz der eben definierten schwachen Stationarität. Auf den Kontext eines Rechenzentrums übertragen kann man davon ausgehen, dass beispielsweise die Dauer einer Datenbankabfrage mit der Uhrzeit und dem Wochentag schwankt. Wenn die Auslastung des Servers an einem Montag um 09:00 Uhr deutlich höher ist, als an einem Sonntag um 03:00 Uhr, wird sich dies auch auf die Abfragedauer auswirken. Nimmt die gespeicherte Datenmenge über die Zeit zu, kann hierdurch ein steigender Trend bei der Abfragedauer entstehen. Es wird klar, dass nicht alle stochastischen Prozesse (schwach) stationär sind. Um die Zeitreihenanalyse dennoch durchführen zu können, beschränkt man sich auf stationäre Prozesse und bereinigt die Zeitreihen entsprechend von Saisonalität und Trend, um die Stationarität herzustellen.
Das Komponentenmodell definiert Merkmale einer Zeitreihe, die bearbeitet werden können, um Stationarität zu erreichen [85, S. 9]. Im folgenden sollen die Komponenten Trend und Saison beleuchtet werden. Des Weiteren existiert die Konjunkturkomponente für eventuell unregelmäßige, mehrjährige Schwankungen sowie die Restkomponente, die unerklärbare Einflüsse beinhaltet [85, S. 9]. Abbildung A.1 (siehe Anhang) zeigt die Zerlegung der Zeitreihe cpuusage_ps in die Komponenten Trend, Saisonalität und Rest bei wöchentlicher Frequenz und Abbildung 2.3 für eine tägliche Frequenz. Dadurch wird ersichtlich, dass in der Zeitreihe zwar eine klare Saisonalität vorliegt, allerdings auch Schwankungen, die nicht erklärbar sind. Die tägliche Saisonalität könnte sich daraus ergeben, dass auf diesem Server jeden Tag zu einer festgelegten Uhrzeit eine Aufgabe ausgeführt wird, die die Auslastung erhöht.

2.3.1 Trendbereinigung

[85, S. 9] definiert den Trend als „langfristige, systematische Veränderung des mittleren Niveaus einer Zeitreihe“. Dies lässt sich am Temperaturverlauf in Deutschland veranschaulichen. Der Aufwärtstrend in der mittleren Tagestemperatur, der in Abbildung

2.1 klar ersichtlich ist, ist Teil der globalen Klimaerwärmung.

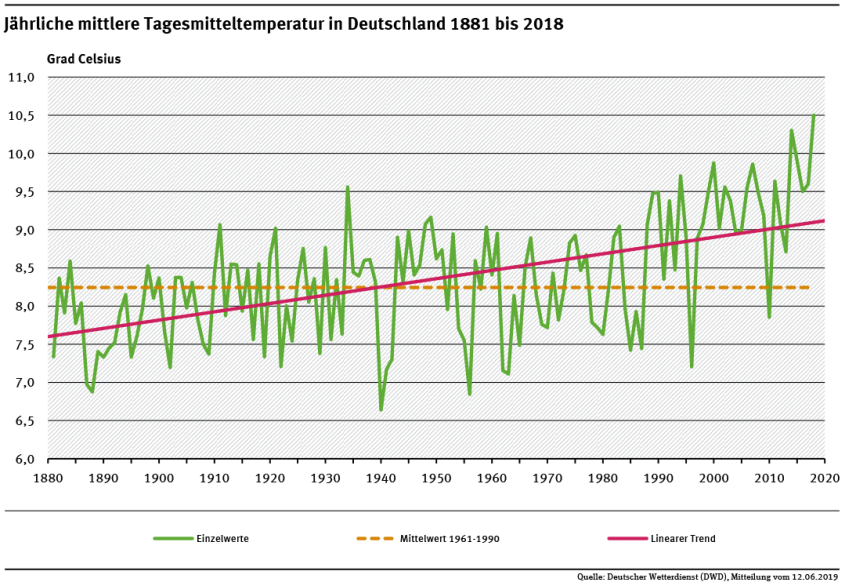

Abbildung 2.1: Der Verlauf der mittleren Tagestestemperaturen in Deutschland für den Zeitraum von 1881 bis 2018 aus [104]. Der steigende Trend ist klar zu erkennen.

Die Trendelimination stellt einen wichtigen Aspekt der Zeitreihenanalyse dar. Wird der Trend nicht vollständig eliminiert, überwiegen die Reste und überdecken andere Komponenten. Andererseits kann zu starke Bereinigung zur Verfälschung der Zeitreihe führen [60, S. 10].

2.3.2 Saisonbereinigung

Die Saisonkomponente einer Zeitreihe ist die „jahreszeitlich bedingte Schwankungskomponente, die sich relativ unverändert jedes Jahr wiederholt“ [85, S. 9]. Ein Beispiel für eine solche Saisonalität lässt sich bei der Zahl der Arbeitslosen in Deutschland erkennen (siehe Abbildung 2.2). Ob für die Serverauslastung eine jährliche Schwankung für gewisse Metriken existiert, lässt sich aus dem vorliegenden Datensatz von vier Wochen nicht sagen. Jedoch ist eine tägliche Saisonalität festzustellen (siehe Abbildung 2.3). Die Saisonalität führt zusammen mit dem Trend und Zufallseinflüssen zur Variation einer Zeitreihe [85, S. 52]. Für das Beispiel der Konjunkturzeitreihe von oben ist oft der Trend von Interesse, weswegen die Zeitreihe saisonbereinigt betrachtet wird.

2.4 AR-, MA- und ARMA-Modelle

Die im Folgenden besprochenen Modelle gehen davon aus, dass es sich um einen stationären Prozess handelt [84, S. 45].

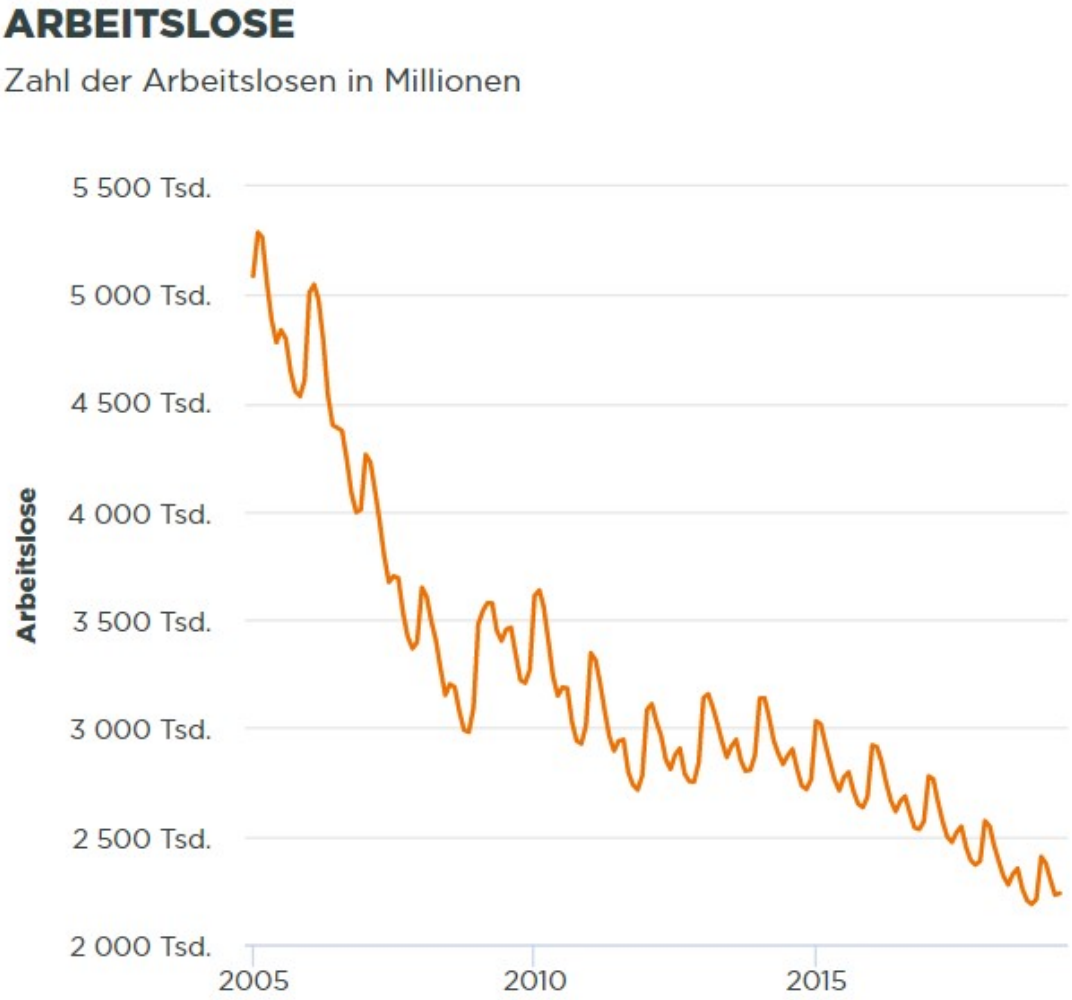

Abbildung 2.2: Zahl der Arbeitslosen in Deutschland von 2005 bis 2019 aus [58]. Es ist erkennbar, dass in den kälteren Wintermonaten die Arbeitslosigkeit regelmäßig ansteigt. Außerdem lässt sich ein Abwärtstrend feststellen.

2.4.1 AR(p)-Modelle

Ein $AR(p)$-Modell ist ein autoregressives Modell p-ter Ordnung [60, S. 84]. Man bezeichnet es als autoregressiv, da sich der Wert zum Zeitpunkt t aus einer Linearkombination der vorhergegangenen Werte ergibt [98, S. 79]. Es ist intuitiv, dass bei den Werten einer Zeitreihe eine Abhängigkeit zu den vorherigen Werten besteht. Jedoch ist nicht klar, wie viele Zeitschritte in die Vergangenheit eine solche Abhängigkeit besteht. Das $AR(p)$-Modell stellt die Zeitreihe durch die in der Zeit verschobene Zeitreihe selbst dar. Der Parameter p des $AR(p)$-Modells gibt an, bis zu welchem Zeitpunkt in der Vergangenheit die Werte berücksichtigt werden sollen [84, S. 45]. Ein AR(p)-Prozess Y_t

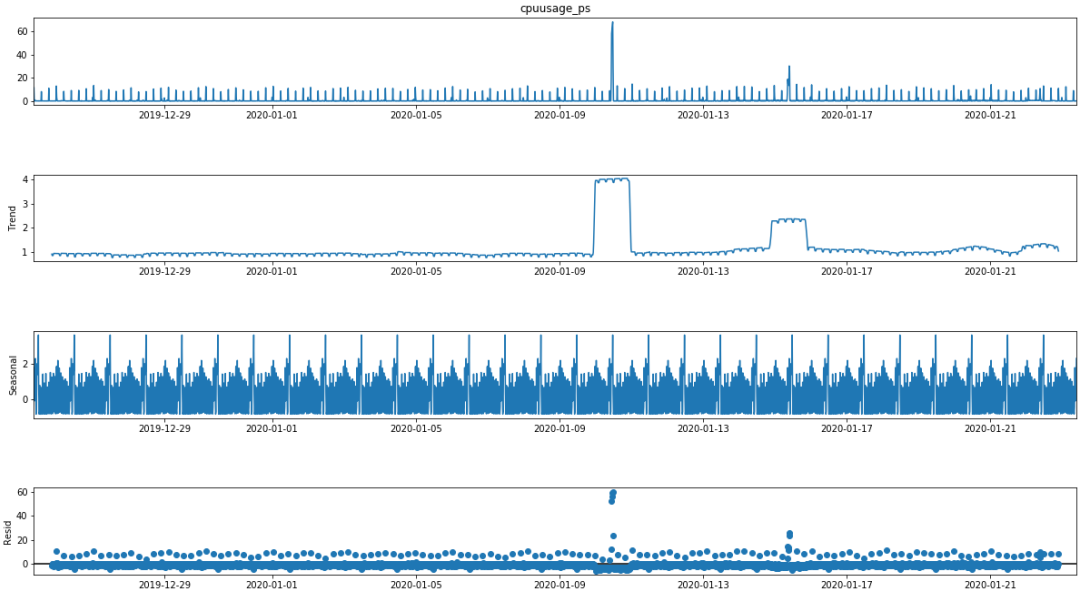

Abbildung 2.3: Zerlegung der Zeitreihe cpuusage_ps in die Komponenten Trend, Saisonalität und Rest mit täglicher Frequenz. Es kein klarer Trend erkennbar, allerdings lässt sich eine tägliche Saisonalität festzustellen. Dennoch liegen mit der Rest-Komponente unerklärliche Einflüsse vor.

lässt sich nach [84, S. 45] durch folgende Gleichung definieren:

$$Y_t = \alpha_0 + \alpha_1 Y_{t-1} + ... + \alpha_p Y_{t-p} + \varepsilon_t. \tag{2.8}$$

Dabei ist Y_t ist ein stochastischer Prozess und ε_t ein White-Noise-Prozess mit Erwartungswert $E(\varepsilon_t) = 0$, der auch „Innovationen“ genannt wird [84, S. 45 f.]. Ein White-Noise-Prozess ist „eine Folge von unabhängigen identisch verteilten Zufallsvariablen“ [84, S. 4]. Aus der identischen Verteilung ergibt sich der Erwartungswert $E(\varepsilon_t) = 0$ für die Zufallsvariable ε_t. Aus der Unabhängigkeit folgt, dass die Kovarianz, das Maß für Abhängigkeit, zweier Punkte $Cov[\varepsilon_t, \varepsilon_s] = 0$ ist für $s \neq t$ [84, S. 97]. Die Innovationen stellen in dieser Gleichung also den Rest dar, für den keine Abhängigkeit existiert.

Um den Parameter p zu finden, kann die Autokorrelationsfunktion (ACF) r_τ verwendet werden, die die Kovarianz der Zeitreihe zu verschiedenen Zeitabständen ins Verhältnis setzt zur Varianz der Zeitreihe [85, S. 7]. Auch die partielle Autokorrelationsfunktion (PACF) kann dafür nützlich sein. Sie beschreibt den Zusammenhang zwischen zwei Messzeitpunkten $t - \tau$ und t, wobei der Einfluss der Werte zwischen diesen Zeitpunkten entfernt wird. Dafür wird deren Erwartungswert, der dank der Stationarität konstant ist, von den zu vergleichenden Werten abgezogen [85, S. 194]. Dies ist nötig, da die Korrelation zweier Variablen allein daher rühren könnte, dass sie beide zu einer gemeinsamen, dritten Variable korrelieren.

ACF und PACF der Zeitreihe cpuusage_ps aus dem vorliegenden Datensatz befinden

sich in den Abbildungen 2.4 und 2.5.

2.4.2 MA(q)-Modelle

Ein $MA(q)$-Modell ist ein Moving-Average-Modell q-ter Ordnung, das die Langzeitwirkung von Störeinflüssen berücksichtigt [60, S. 101]. Das MA(q)-Modell erklärt die Zeitreihe durch einen gewichteten White-Noise-Prozess.

Ein $MA(q)$-Prozess wird durch folgende Gleichung bestimmt:

$$Y_t = \varepsilon_t - \beta_1 \varepsilon_{t-1} - ... - \beta_q \varepsilon_{t-q}, \tag{2.9}$$

wobei ε_t erneut ein White-Noise-Prozess ist [84, S. 65]. Daher sind die ε_{t-i} mit $i = 0..q$ unabhängige Zufallsvariablen, die mit β_i gewichtet werden. Der Parameter q kann ebenfalls anhand der Autokorrelationsfunktion ermittelt werden und sollte so gewählt sein, dass ab dem Lag q kein Wert der Autokorrelationsfunktion mehr die sogenannten „Barlett-Grenzen“ überschreitet [84, S. 60 f.]. Man untersucht also, für welchen Lag kein relevanter Zusammenhang mehr zwischen den Daten besteht. Die Relevanz wird dabei über eine obere und untere Grenze definiert. Diese sind jeweils die doppelte geschätztes Standardabweichung mit positivem Vorzeichen für die obere, und negativem Vorzeichen für die untere Grenze [84, S. 61]. Abbildung 2.4 zeigt den Plot der ACF, das sogenannte Korrelogramm, für die Zeitreihe cpuusage_ps aus dem Datensatz.

2.5 ARMA(p,q)-Modelle

Unter autoregressiven Moving-Average-Prozessen ($ARMA(p,q)$-Prozessen) versteht man nach [84, S. 62] die Kombination eines AR(p)- und eines MA(q)-Prozesses in folgender Form:

$$Y_t = \alpha_1 Y_{t-1} + ... + \alpha_p Y_{t-p} + \varepsilon_t - \beta_1 \varepsilon_{t-1} - ... - \beta_q \varepsilon_{t-q}. \tag{2.10}$$

Ein ARMA(p,q)-Prozess nutzt also gleichzeitig einen $AR(p)$- und einen $MA(q)$-Prozess.

Bringt man den AR-Anteil auf die linke Seite lässt sich das ARMA(p,q)-Modell auch in der nach [84, S. 62] üblichen Form angeben als

$$Y_t - \alpha_1 Y_{t-1} - ... - \alpha_p Y_{t-p} = \varepsilon_t - \beta_1 \varepsilon_{t-1} - ... - \beta_q \varepsilon_{t-q}. \tag{2.11}$$

Somit lässt sich festhalten, dass ein $AR(p)$-Prozess ein $ARMA(p,0)$-Prozess, respektive ein $MA(q)$-Prozess ein $ARMA(0,q)$-Prozess ist [98, S. 101]. ARMA-Prozesse ermöglichen es, ein „komplexes Verhalten einer Zeitreihe [...] mit einer geringen Zahl von Parametern [zu] beschreiben“ [84, S. 63]. Die gleichzeitige Verwendung erlaubt es also, kleinere Ordnungen zu wählen. Ein großer Vorteil von ARMA-Prozessen ist nach [84, S. 63], dass die Summe zweier ARMA-Prozesse wiederum einen ARMA-Prozess ergibt:

$$ARMA(p,q) + ARMA(p',q') = ARMA(p+q', max\{q+p', p+q'\}). \tag{2.12}$$

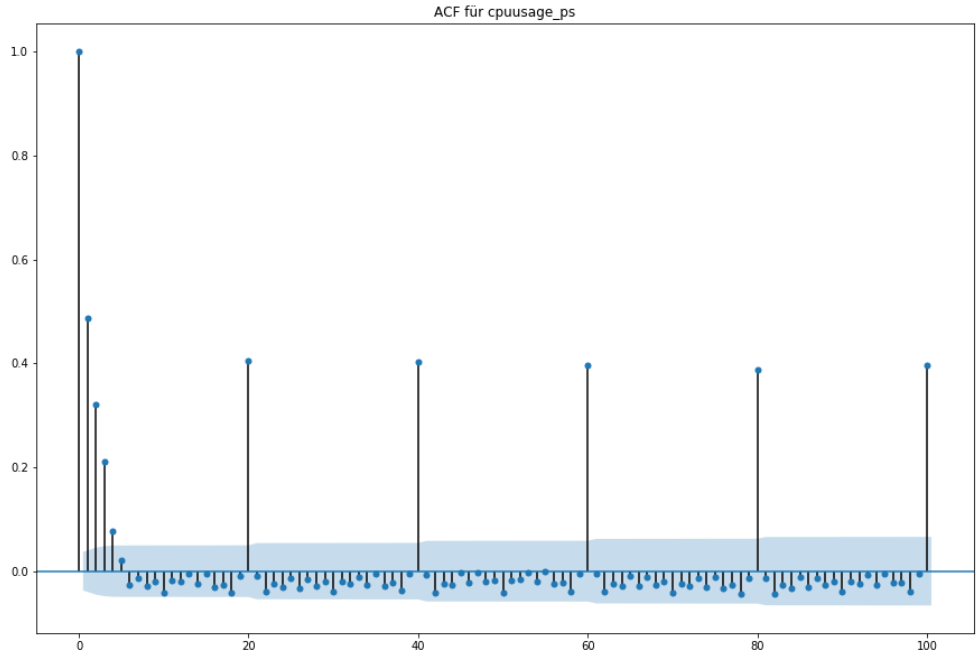

Abbildung 2.4: Korrelogramm für die Zeitreihe cpuusage_ps. Bis zum vierten Lag sind Abhängigkeiten sichtbar. Außerdem nimmt die Funktion beim 20. Lag einen relativ gesehen hohen Wert an, sodass auch hier eine Abhängigkeit besteht. Dieser wiederholt sich für jeden weiteren 20. Lag.

So ist es möglich, zwei Zeitreihenmodelle zu aggregieren [84, S. 63], indem die p und q Ordnung aus den Ordnungen der Ausgangsprozesse berechnet werden.

2.6 ARIMA(p,d,q)-Modelle

Auf den vorher angesprochenen Modellen beruht eine weitere Möglichkeit zur statistischen Zeitreihenvorhersage: die ARIMA-Modelle. ARIMA steht für „autoregressive integrated moving average“ [98, S. 122]. Der Name ergründet sich darin, dass sich ein ARIMA-Modell Integrieren eines ARMA-Modells ergibt. Damit ist allerdings nicht die Integralrechnung aus der Analysis gemeint. Beispiele für die Verwendung von ARIMA-Modellen liefern unter anderem [67], [19] und [26].

ARIMA-Prozesse können für nicht-stationäre Zeitreihen verwendet werden, da diese sich durch Differenzenbildung als ein stationärer ARMA-Prozess modellieren lassen [84, S. 69]. Das Kriterium der Stationarität für einen ARMA-Prozess wird also erfüllt, indem der Trend mittels Differenzenbildung entfernt wird.

Die Differenzenbildung erfolgt nach [98, S. 123] in folgender Form:

$$Y'_t = Y_t - Y_{t-1} = Y_t - BY_t = (1 - B)Y_t. \tag{2.13}$$

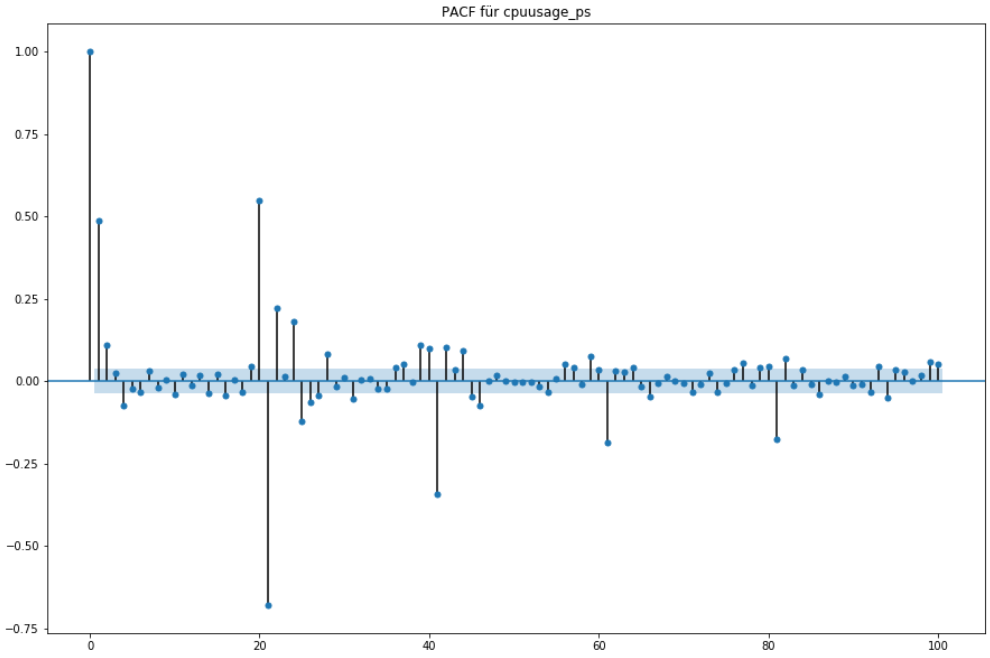

Abbildung 2.5: Partielle Autokorrelationsfunktion für die Zeitreihe cpuusage_ps. Interessante Lags sind demnach 1, 2, 4 sowie 20, 21, 22, 24, 25, 26 und 28. Außerdem sind die Lags 40, 60 und 80 auffällig.

Dabei ist B der sogenannte „Backshift-Operator“, der die Verschiebung der Zeitreihe von Zeitpunkt t hin zu Zeitpunkt $t-1$ bewirkt:

$$BY_t := Y_{t-1} \tag{2.14}$$

Man bildet also die Differenz aus dem stochastischen Prozess Y_t und dem Prozess zum vorherigen Zeitpunkt Y_{t-1}. Dies wird zum Teil auch als „differenzieren“ bezeichnet, sollte aber nicht mit der Differenzialrechnung der Analysis verwechselt werden. Hat beispielsweise Y_t einen linearen Trend a, wie er in Abbildung 2.1 erkennbar ist, lässt sich der Prozess somit schreiben als $Y_t = a + bt + \varepsilon_t$, wobei ε_t ein zentrierter stationärer Prozess ist. Dann ergibt sich unter Anwendung von Gleichung (2.13):

$$Y'_t = a + bt + \varepsilon_t - (a + b(t-1) + \varepsilon_{t-1}) = b + \varepsilon_t - \varepsilon_{t-1} \tag{2.15}$$

[98, S. 124]. Der Trend a kommt in Y'_t nicht mehr vor.

Wenn man nun eine Prognose für Y'_t erstellt hat, kann man durch Integrieren, also das Addieren des vorher abgezogenen Terms, die Prognose für Y_t umrechnen: $Y_t = Y'_t + Y_{t-1}$.

Angewendet auf Gleichung (2.15) bedeutet dies:

$$Y_t = Y'_t + Y_{t-1} = b + \varepsilon_t - \varepsilon_{t-1} + a + b(t-1) + \varepsilon_{t-1} = a + bt + \varepsilon_t. \tag{2.16}$$

Das zeigt, dass auf diese Weise der Trend a in der Prognose wieder berücksichtigt wird.

Der Parameter d eines $ARIMA(p,d,q)$-Prozesses gibt an, wie oft die Differenz gebildet wird. Die Bedeutung von p und q ist Analog zu einem $ARMA(p,q)$- bzw. $AR(q)$- und $MA(p)$-Prozess. Wird ein $ARIMA(p,d,q)$-Prozess d-mal differenziert, entsteht ein $ARMA(p,q)$-Prozess. Durch das Differenzieren kann die Zeitreihe auch von nichtlinearen Trends bereinigt werden [98, S. 123]. Beinhaltet die Reihe beispielsweise einen quadratischen Trend, kann durch 2-faches Differenzenbilden, also mit $d = 2$, Stationarität hergestellt werden:

$$Y''_t := Yt' - Yt - 1' \tag{2.17}$$

[98, S. 123]. So lässt sich unter Verwendung von Gleichungen (2.13) und (2.14) für das zweifache Differenzieren schreiben:

$$Y''_t = Y'_t - Y'_{t-1} = (1-B)^2 Y_t \tag{2.18}$$

[98, S. 123].

2.6.1 Vorgehensweise zur Zeitreihenvorhersage mit ARIMA-Modellen

Um mit einem ARIMA-Modell Vorhersagen für eine Zeitreihe zu erzeugen, kann man laut [19] grundlegend nach folgenden Schritten vorgehen.

1. Auswählen einer Modell-Klasse, beispielsweise ARMA oder ARIMA.
2. Aufstellen eines Modells und Schätzen der Modellparameter.
3. Überprüfen, ob die Residuen, also die ε_t ein White-Noise-Prozess sind. Falls dem so ist, kann mit dem nächsten Schritt fortgefahren werden, ansonsten muss zum vorherigen Schritt zurückgegangen werden.
4. Das Modell ist bereit für die Prognosenerstellung.

Da jetzt einige Grundlagen zur Zeitreihenanalyse in der Statistik erklärt sind, soll im Folgenden die Entscheidung für Python als Programmiersprache für datenwissenschaftliche Projekte begründet werden.

Kapitel 3

Python für Machine-Learning und Data Science

Die Programmiersprache Python hat sich in den letzten Jahren wachsender Beliebtheit erfreut. Obwohl sie im TIOBE Index 2018 das erste Mal unter die Top drei gewählt wurde, erhielt sie von dem Unternehmen den Titel als die Sprache mit dem größten prozentualen Zuwachs den Titel „Programmiersprache des Jahres 2018“ [68]. Im März 2020 belegt Python hinter Java und C den dritten Platz im TIOBE Ranking noch vor C++ und C# [96]. Dies zeigt, dass Python aktuell eine der bedeutendsten Programmiersprachen ist. Besonders in den Bereichen des maschinellen Lernens und der damit eng verbundenen Disziplin Data Science ist Python zudem weit verbreitet. Eine Untersuchung der Stellenausschreibungen mit Bezug zu Data Science am deutschen Arbeitsmarkt ergab, dass Python mit Erwähnungen in rund 30% der Anzeigen die meistgesuchte Programmiersprache in diesem Kontext ist [72]. Einen weiteren Beleg für die Relevanz von Python für diesen Anwendungsbereicht liefert GitHub. Github ist ein Portal, das die Verwendung des Versionsverwaltungssystem Git anbietet. Github veröffentlicht jährlich den „State of the Octoverse Report 2018“. Dabei stellten die Autoren fest, dass sehr viele Projekte maschinelles Lernen beinhalten [27]. Genauere Betrachtungen der Autoren ergaben dann, dass auf GitHub Python die beliebteste Programmiersprache für Machine Learning ist. Auch die Python-Entwickler selbst geben laut dem Python Developers Survey 2018 an, die Sprache zu großen Teilen für datenwissenschaftliches Arbeiten - damit sind sowohl Data Science als auch Machine Learning gemeint - zu verwenden [72].

Die somit belegte Verbreitung Pythons im datenwissenschaftlichen Kontext geht einher mit der Existenz zahlreicher Publikationen, themenverwandten wissenschaftlichen Arbeiten und Programmierbibliotheken. Einige dieser Bibliotheken und ihre Verwendungszwecke werden im Folgenden beleuchtet.

3.1 Relevante Python-Bibliotheken

Im Folgenden werden kurz ausgewählte Python Bibliotheken vorgestellt, die in den Bereichen Data Science und Machine Learning weit verbreitet und für diese Arbeit relevant sind.

3.1.1 Numpy

Im Paket *Numpy* sind Datenstrukturen wie N-Dimensionale Arrays und Matrizen sowie Funktionen der Linearen Algebra implementiert. Durch die hardwarenahe Umsetzung in der Programmiersprache C wird eine sehr gute Performanz erreicht. „Numpy“ steht für „Numeric Python“ und ist Teil vieler Anwendungen aus dem Bereich „big data“ [56, S. 43]. Numpy kam in dieser Arbeit nicht nur über andere Bibliotheken, die Numpy verwenden, indirekt zum Einsatz, sondern wurde auch direkt bei der Datenverarbeitung und -aufbereitung genutzt.

3.1.2 Pandas

Pandas ermöglicht eine effiziente Analyse und Verarbeitung großer Datenmengen [66]. Pandas basiert auf Numpy und kann sowohl lesend als auch schreibend DSV- und Excel-Dateien verarbeiten [56, S. 253]. Pandas wurde in der vorliegenden Arbeit verwendet, um die Daten einzulesen und mit ihnen in einem Tabellenartigen Format zu arbeiten.

3.1.3 statsmodels

In *statsmodels* werden verschiedene Modelle, Tests sowie Funktionen zur Datenexploration implementiert [87]. statsmodels wurde zum Beispiel für die Komponentenzerlegung verwendet.

3.1.4 Matplotlib

Matplotlib ist ein Open-Source-Modul, das zur Datenvisualisierung verwendet wird. Es ermöglicht die Erstellung von Diagrammen verschiedener Typen sowie die optische Anpassung dieser Plots [56, S. 167]. Um Daten außerhalb der Webanwendung zu visualisieren wurde im Rahmen dieser Arbeit Matplotlib verwendet.

3.1.5 Tensorflow

Tensorflow ist ein low-level Open-Source-Modul für Machine Learning, das von Google entwickelt wurde [69, S. 11]. Der Name setzt sich aus „Tensor“ und „flow“ zusammen. Tensoren sind Strukturen die Daten enthalten und der Begriff „flow“ rührt daher, dass diese Tensoren in einem Flussdiagraphen verarbeitet werden [99, S. 97 f.]. Tensoren können dabei n Dimensionen haben und dadurch Skalare, Vektoren, Matrizen sowie mehrdimensionale Felder darstellen [99, S. 97 f.].

3.1.6 Keras

Keras ist ein high-level Machine Learning Framework, das als API-Schnittstelle für mehrere low-level Frameworks dient. Das am häufigsten verwendete Backend für Keras ist Tensorflow [69, S. 11 f.]. Das bedeutet, dass die Keras-Funktionen intern Tensorflow-Code verwenden. So ist ein Arbeiten auf einer höheren Abstraktionsebene möglich [69, S. 11 f.] . Die tiefen Neuronalen Netze, die in dieser Arbeit verwendet werden, wurden mittels Keras implementiert.

3.1.7 Scikit-learn

Scikit-learn bietet verschiedene aktuelle Machine Learning Algorithmen auf einer hohen Abstraktionsebene und mit einfachen Schnittstellen an. Die Funktionen sind effizient implementiert und bieten eine gute Performance [74]. Zu den Nutzern von Scikit-learn gehören unter anderem J.P.Morgan, Spotify und Booking.com [102]. Scikit-learn wurde für diese Arbeit an mehreren Stellen verwendet. Sowohl die Hauptkomponentenanalyse und Standardisierung (siehe Abschnitt 6.2) als auch die Rastersuche und Kreuzvalidierung (siehe Abschnitte 6.4.1 und 6.4.2) wurden mit Scikit-learn umgesetzt.

Das folgende Kapitel stellt einen Prozess vor, der die Herangehensweise an ein Data Science Projekt beschreibt.

Kapitel 4

Der Data Science Prozess

Die Herangehensweise an eine Problemstellung wie sie dieser Arbeit zugrunde liegt, kann mittels in der Abbildung 4.1 gezeigten Data Science Prozess beschrieben werden.

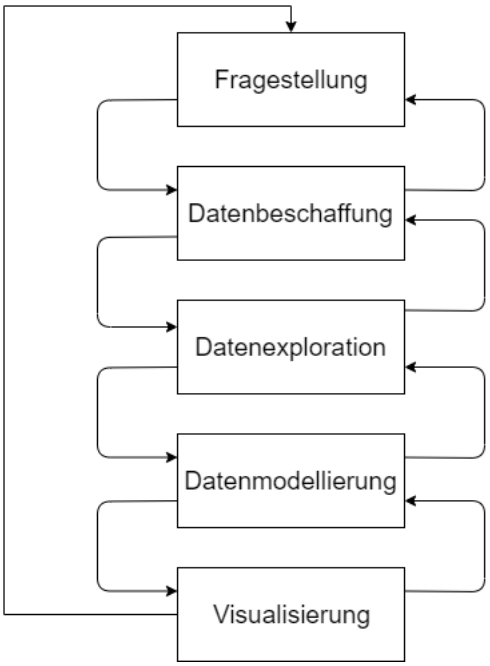

Abbildung 4.1: Der Data Science Prozess nach [5].

Dieser formuliert die Schritte Fragestellung, Datenbeschaffung, Datenexploration, Datenmodellierung und Datenvisualisierung. Dabei handelt es sich allerdings nicht um einen linearen Prozess und die Stufen werden nicht zwingend nacheinander durchlaufen. Vielmehr sind Iterationen notwendig, in denen vorhergegangene Schritte erneut betrachtet werden. In einem linearen Durchlauf würde zunächst also eine Fragestellung beziehungsweise ein Ziel definiert werden. Für dieses Ziel müssen dann die Daten beschafft werden. Anschließend können diese exploriert und analysiert werden, sodass im nächsten Schritt ein Modell aufgestellt werden kann. Die Prognosen dieses Modells werden im letzten Schritt visualisiert. Iterative Schritte zu einer vorherigen

Stufe könnten beispielsweise dadurch notwendig werden, dass die Ergebnisse eines Schrittes nicht zufriedenstellend sind oder wenn sich die Anforderungen während des Prozesses ändern.

4.1 Fragestellung

Die Frage „Wie stark wird die CPU-Auslastung eines Servers zu einem in der Zukunft liegenden Zeitpunkt t sein?“ beschreibt die der Arbeit zugrunde liegende Fragestellung und kann den ersten Schritt im Data Science Prozess einnehmen. Die in Kapitel 1 dargelegten potenziellen Vorteile einer Prognose verleihen der Beantwortung dieser Frage große Relevanz. Die möglichen finanziellen Einsparungen und Sicherstellung der Verfügbarkeit sind dabei für Unternehmen von herausragender Bedeutung.

4.2 Datenbeschaffung (1)

Zu Beginn dieser Arbeit lag bereits ein Datensatz in Form einer 14-Megabyte großen CSV-Datei vor. Dieser diente dem Unternehmen als Beispieldatensatz für die Entwicklung des Monitoring-Dashboards. Es handelt sich bei den Einträgen um reale, anonymisierte Messwerte aus den Rechenzentren der Bundesagentur für Arbeit, für die die Webapp entwickelt wird.

4.3 Datenexploration (1)

Der Datensatz umfasst etwa siebzigtausend Zeilen. Die Messwerte haben eine 15-minütige Abtastrate. Dabei wurden die Auslastung verschiedener Server in mehreren Clustern gemessen wurde. Die Last wird als Zugriffsanzahl im Messzeitraum festgestellt. Dies ist jedoch das einzige quantitative Merkmal im Datensatz, das für die Analyse beziehungsweise die Prognose verwendet werden kann. Das Diagramm in Abbildung 4.2 erlaubt einen ersten Blick auf diese Daten.

Nach Konsolidierung mit einem Datenbankadministrator der Bundesagentur für Arbeit wurde klar, dass ein völlig neuer Datensatz benötigt wurde. Der vorliegende Datensatz war bereits vorverarbeitet und es konnte nicht rekonstruiert werden, welche Schritte dabei stattgefunden haben. Da in dieser Arbeit Machine Learning angewendet werden sollte, war es zudem gewünscht, einen Datensatz mit mehreren quantitativen Merkmalen zu untersuchen. Daher musste eine neue Iteration im Prozess beginnen, an deren Anfang die Datenbeschaffung stand.

4.4 Datenbeschaffung (2)

Es war möglich, neue Messdaten und deren Freigabe zu organisieren. Allerdings ist es mit der verwendeten Datenbank nur möglich, die Rohdaten von sieben Tage zu

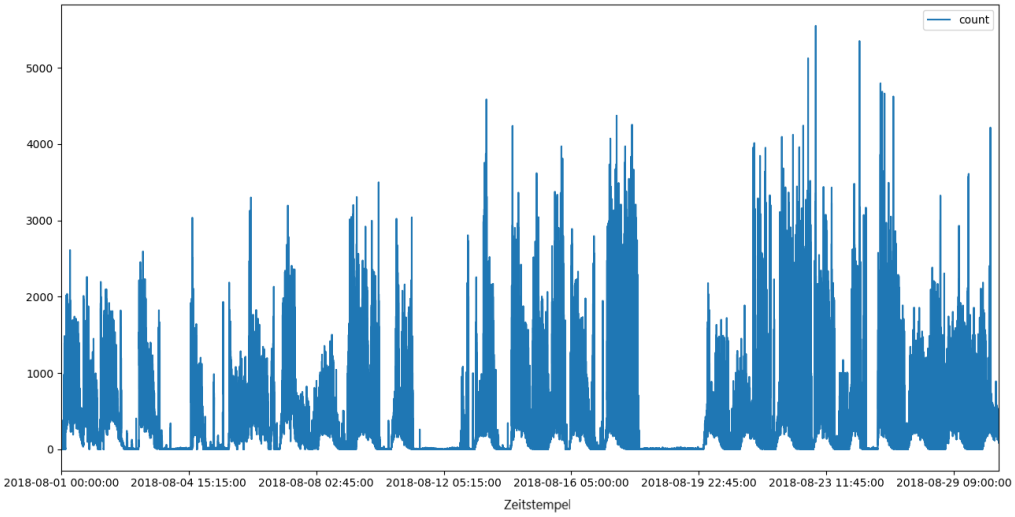

Abbildung 4.2: Plot zu dem ersten Datensatz. Der „count“ ist dabei der Wert einer Metrik, die die Zugriffszahlen auf die Datenbank beschreibt. Es war jedoch nicht mehr rekonstruierbar, welche Metrik genau verwendet wurde.

extrahieren. Etwas später kamen erneut die Rohdaten einer weiteren Woche hinzu, allerdings gab es zwischen den beiden Wochen eine Lücke.

4.5 Datenexploration (2)

Bereits auf den ersten Blick wurde klar, dass es sich um eine deutlich größere Datenmenge handelt. Die Daten einer Woche lagen nun in einer knapp 33 Gigabyte großen DSV-Datei vor. Somit war die Datei zu umfangreich für viele Programme, um geöffnet zu werden. Der für die Abbildung 4.2 verwendete Quellcode war nicht in der Lage, die Daten zu verarbeiten. Die Datenmenge war so groß, dass die verwendete Bibliothek „Pandas“ einen Fehler erzeugte. Die Lösung bestand darin, die Daten in Blöcken einzulesen und zu verarbeiten. Dies zeigte, dass dieser Datensatz mehr als 130 Millionen Einträge enthält. Außerdem wurde klar, dass bei diesen Daten wie gewünscht mehrere quantitative Merkmale vorlagen. Allerdings waren einige Spalten in allen Reihen leer, redundant oder irrelevant für den Zweck der Zeitreihenvorhersage. Die quantitativen Merkmale bestanden dabei aus einer Reihe verschiedener Metriken an den Servern. Außerdem lagen die Metriken nicht als einzelne Spalte vor, sondern es gab eine Spalte, in der der Name der Metrik stand, und eine, in der der Wert der Metrik stand. Dies verbraucht aufgrund der redundanten Speicherung der meisten Spalten nicht nur unnötig viel Speicher, sondern ist auch unpraktikabel für das weitere Vorgehen. In Abbildung 4.3 wird der Vergleich zwischen dem vorliegenden und dem gewünschten Datensatzformat skizziert.

Da für den nächsten Schritt im Data-Science Prozess, die Datenmodellierung, allerdings ein viel tieferes Verständnis für die zu modellierenden Daten nötig ist, musste der

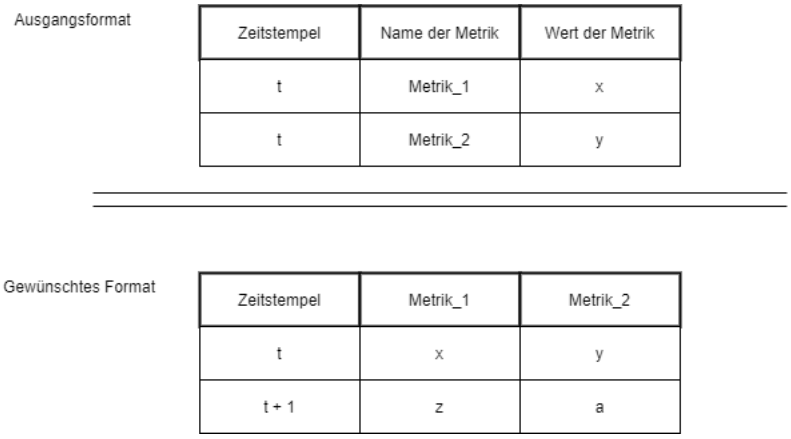

Abbildung 4.3: Vergleich der Datensatzformate. *Oben:* In diesem Format lagen die Datenbankexporte vor. *Unten:* Dieses Format sollte für die weitere Verarbeitung erzeugt werden.

Datensatz weiter analysiert werden. Dafür wurden als Nächstes die quantitativen Merkmale genauer untersucht um ein Feature Engineering durchzuführen. In [23, S. 84] bezeichnet der Autor das Feature Engineering als den Faktor, der darüber entscheidet, ob ein Machine-Learning-Projekt scheitert oder erfolgreich ist. Ein „Feature“ ist dabei ein quantitatives Merkmal aus dem Datensatz, das die Grundlage für Prognosen bildet und „Feature Engineering“ bezeichnet das "Herausarbeiten von relevanten Merkmalen als Eingabe"[103, S. 78]. Neben [23] weißt auch [103, S. 78] auf die hohe Bedeutung des Feature Engineering hin.

Zunächst wurden alle vorhandenen Merkmale mit ihrer Abkürzung und dem vollen Namen in eine CSV-Datei gespeichert. Dafür wurde das im Python-Standard enthaltene Paket `csv` und die Methode `writerow()` eines `writer`-Objekts [20] verwendet. Beim blockweisen Einlesen des Datensatzes wurden die Datentypen der Spalten angegeben, um die Performance zu verbessern. Auf diese Weise müssen sie nicht mehr von `read\_csv`-Funktion bestimmt werden. Außerdem wurde das Python-Modul `pickle` verwendet um die Blöcke zu serialisieren und als Binärdateien zu speichern, sodass der Datensatz zukünftig nicht komplett neu gelesen werden muss. Es wurde untersucht, ob die Daten vollständig sind. Durch die Pandas-Funktion `isnull()` wurde geprüft, ob im Datensatz für jede Zeile alle Spalten ausgefüllt sind. Es zeigte sich, dass der Datensatz vollständig ist. Es war demnach nicht nötig, das Fehlen von Werten zu beheben.

Eine Untersuchung auf Duplikate im Datensatz mittels der Pandas-Funktion `duplicated()` ist in diesem speziellen Fall nicht relevant. Duplikate bedeuten lediglich, dass die gleichen Messwerte zu mehreren Zeitpunkten vorlagen. Würden mehrfach vorkommende Werte entfernt, würde der Datensatz verfälscht und somit die Prognose negativ beeinflusst.

Da jedoch das tiefere Verständnis für die Features fehlte und deren Bezeichnungen unklar waren, war es nötig, Kontakt mit der Bundesagentur für Arbeit aufzunehmen, um in einem Interview Klarheit zu schaffen.

4.5.1 Rücksprache mit Datenbankadministrator

Im Gespräch mit einem der Zuständigen für Datenbankadministration und -support bei der Bundesagentur für Arbeit sollten folgende offene Fragen zum Datensatz geklärt werden:

- Gibt es andere Exportmöglichkeiten der Datenbank?
- Lässt sich der Messzeitpunkt vereinheitlichen?
- Welche Bedeutung haben die Spalten?
- Welche Metriken bzw. Merkmale sind interessant?
- Welchen Features entsprechen diese?
- Welche Metrik soll prognostiziert werden? Beziehungsweise: Was ist die Zielvariable?
- Welche Metriken sollen dafür als Eingabe verwendet werden?

Im Interview erläuterte der Kontakt, dass die Bundesagentur für Arbeit den Oracle Enterprise Manager zur Verwaltung der Datenbanken verwendet. Die Rohdaten werden für sieben Tage gespeichert, danach liegen sie nur noch aggregiert vor. Daher ist es nicht möglich, die Daten in anderer Form zu exportieren. Die Messzeitpunkte lassen sich nicht vereinheitlichen, da die Werte aus dem Produktivsystem der Bundesagentur für Arbeit kommen und die nötige Neukonfiguration daher nicht durchgeführt werden kann. Die Bedeutung der Spalten ergibt sich zum Teil aus dem Interview sowie aus der Dokumentation des Enterprise Managers [28] und wird in Tabelle B.1 (siehe Anhang) dargestellt.

Der Oracle Enterprise Manager erfasst dabei die Features selbstständig. Aus Sicht des Datenbankadministrators sind die Metriken zu den Sessions, dem PGA-Speicher und zur CPU-Auslastung interessant, da er diese überprüft, um den Zustand des Systems zu beurteilen. Allerdings konnte nicht geklärt werden, welche Metriken sinnvoll für die Eingabe sind. PGA ist die Abkürzung für „Program global area“, einem Speicherbereich, in dem jeder Server Prozess Daten und Kontrollinformationen ablegen kann [22]. Die Summe der einzelnen PGA-Speicher wird in der Metrik „pga_total“ erfasst. Die Spalte „TARGET_NAME“ enthält den anonymisierten Namen der Datenbankinstanz. Im Datensatz sind mehrere Datenbankinstanzen aufgeführt, für die jeweils einzeln die Metriken erhoben werden. Daher war es für folgende Schritte nötig, zunächst eine Datenbankinstanz auszuwählen.

Zusätzlich war es nötig, die Daten weiter zu transformieren, um das gewünschte Format des Datensatzes zu erreichen.

4.6 Datenbeschaffung (3)

Da die Daten einer Woche nicht zwingend aussagekräftig sind, sollte eine größere Datenbasis für die Vorhersage geschaffen werden. Wie oben beschrieben ist ein Export

immer nur über die letzten sieben Tage möglich. Daher wurde ein Zeitraum von vier Wochen festgelegt. Das stückweise Abziehen der Datenbank nach begann nach Weihnachten. Gegen Ende Januar standen vier Datensätze mit je einer Woche Messwerten zur Verfügung.

4.7 Datenexploration (3)

Die vier einzelnen Datensätze mussten als erstes zusammengefügt werden. Dafür wurde eine Datenbankinstanz ausgewählt und für diese aus allen Datensätzen die Vorkommnisse extrahiert. Dafür wurden die Dateien erneut blockweise eingelesen. Anschließend wurde die Datenbankinstanz ausgewählt und schließlich die Datensätze durch die Pandas-Funktion `concat()` zusammengeführt. Dabei fiel auf, dass es bei den Ausgangsdaten Überschneidungen an den jeweiligen Anfangs- und Endtagen gab. Dies wurde bei der Vereinheitlichung der Zeitstempel bereinigt. Denn um den Datensatz praktikabel verwenden zu können, sollte er die folgenden Charakteristika erfüllen:

- Einheitliche Zeitstempel in gleichem Abstand
- Die Merkmale sind Spalten
- Für jede Reihe sind alle Merkmalswerte eingetragen

4.7.1 Vereinheitlichung der Timestamps

Die Problematik bestand hier hauptsächlich darin, dass manche Metriken in zehnminütigen und andere in 15-minütigen Intervallen erfasst wurden. Außerdem gab es einige wenige Metriken, die in einer noch geringeren Frequenz gemessen wurden.

Zunächst musste also für jedes Merkmal festgehalten werden, in welchen Abständen die Zeitstempel vorliegen. Anschließend wurden die Minuten der Timestamps auf den nächstmöglichen Wert gerundet. Dabei gab es für 15-minütige Features vier mögliche Werte: 0, 15, 30, 45 und 60. Für Zehn-Minuten-Intervalle gab es sechs: 0, 10, 20, 30, 40, 50, 60. Wurde eine Minute auf 60 gerundet, wurde auf den Zeitstempel eine Stunde addiert und dann die Minute auf 0 gesetzt. Die Sekunden wurden für jede Reihe auf 0 gesetzt. Allerdings musste eine weitere Anpassung vorgenommen werden, um die Zeitstempel für alle Merkmale, unabhängig von ihrer Abtastfrequenz, zu vereinheitlichen. Da die 15-minütigen Messungen seltener stattfinden sind sie ungenauer. Daher wurde für den multivariaten Datensatz das Abtastintervall auf 15-Minuten festgelegt. Dafür wurde das Prinzip aus Tabelle 4.1 auf Seite 24 verwendet. Hat das Merkmal einen 15-minütigen Abstand, konnten die Werte einfach übernommen werden. Bei einer zehnminütigen Periode konnten hingegen nur die Werte zu den Minuten hh:00 und hh:30 direkt verwendet werden. Der Wert zum Zeitpunkt hh:15 wurde als arithmetisches Mittel zwischen den Werten zu den Zeitpunkten hh:10 und hh:20 gebildet. Für hh:45 wurde mit den Werten zu den Zeiten hh:40 und hh:50 analog vorgegangen. Diese Transformation wurde mit dem Code aus Listing C.6 (siehe Anhang) erreicht. Anschließend wurden die nicht mehr benötigte Zeilen, also solche deren Zeitstempel nicht in das 15-Minuten-Intervall passt und die durch einen

Neuer Zeitstempel	Verwende Wert vom Zeitstempel (falls 15-min Intervall)	Verwende Wert vom Zeitstempel (falls 10-min Intervall)
YYYY-MM-DD hh:00:00	f(YYYY-MM-DD hh:00:ss)	f(YYYY-MM-DD hh:00:ss)
YYYY-MM-DD hh:15:00	f(YYYY-MM-DD hh:15:ss)	0.5 * (f(YYYY-MM-DD hh:10:ss) + f(YYYY-MM-DD hh:20:ss))
YYYY-MM-DD hh:30:00	f(YYYY-MM-DD hh:30:ss)	f(YYYY-MM-DD hh:30:ss)
YYYY-MM-DD hh:45:00	f(YYYY-MM-DD hh:45:ss)	0.5 * (f(YYYY-MM-DD hh:40:ss) + f(YYYY-MM-DD hh:50:ss))

Tabelle 4.1: Prinzip zur Vereinheitlichung der Zeitstempel. $f(ts)$ gibt den Wert vom Zeitpunkt ts zurück.

vereinheitlichten Zeitstempel ersetzt wurden, mittels dem Code in Listing C.7 (siehe Anhang) aus dem Datensatz entfernt.

4.7.2 Merkmale als Spalten

Um die Merkmale aus den Zeilen in die Spalten zu transformieren, wurde dem Datensatz zunächst für jedes vorhandene Merkmal eine Spalte hinzugefügt, die mit `NaN` initialisiert wurden. Dann wird für jede Reihe der Wert aus der Spalte „VALUE“ in die entsprechende Metrik-Spalte, die sich aus „METRIC_COLUMN“ ergibt, eingetragen. Listing C.8 (siehe Anhang) zeigt, wie dies implementiert wurde. Im Anschluss an diese Transformation wurden noch textuelle Spalten sowie leere Platzhalterspalten (siehe Tabelle B.1) entfernt. Diese beinhalten auch keine Informationen, die für das Modell von Bedeutung wären, sondern lediglich beispielsweise den Benutzernamen der Datenbankverbindung oder Dateinamen.

4.7.3 Keine fehlenden Merkmalswerte

Da bei den vierwöchigen Daten manche Merkmalswerte mit einer geringeren Frequenz als 15 Minuten abgetastet wurden, gab es im Datensatz zunächst noch `NaN`-Werte, also leere Felder ohne jeglichen Wert. Um diese zu füllen, wurden zunächst alle fehlenden Werte x_t zum Zeitpunkt t mit dem nächst früheren, gültigen Wert x_{t-n} mit $n > 0$ aufgefüllt. Nach diesem sogenannten „forward fill“ wurde noch ein „backward fill“ angewandt, bei dem ein `NaN` bei x_t durch den Wert von x_{t+n} mit $n > 0$ ersetzt wird. So wurden `NaN`-Werte für die erste Zeile, beziehungsweise den ersten Zeitstempel, aufgefüllt.

Aus dem so entstandenen Datensatz wurden die Metriken, die der Datenbankadministrator verwendet (siehe Abschnitt 4.5.1), geplottet. Die Ergebnisse finden sich in den Abbildungen 4.4, A.2 und A.3. Für alles weitere ist hauptsächlich die Metrik

Kennzahl	Wert
Arithm. Mittel	1.11
Standardabweichung	3.44
Minimum	0.22
Maximum	67.84
25%-Quantil	0.29
50%-Quantil / Median	0.41
75%-Quantil	0.49

Tabelle 4.2: Kennzahlen zur Zeitreihe cpuusage_ps. Die Werte wurden auf zwei Nachkommastellen gerundet. Ein p-Quantil ist dabei ein Wert, für den p Prozent der Zeitreihenwerte kleiner oder gleich sind. 25% der Messungen sind also kleiner oder gleich 0.22.

„cpuusage_ps“ aus der Abbildung 4.4 interessant, da diese in der Web-Anwendung angezeigt und vorhergesagt werden soll. Aus diesem Plot wird auch klar, dass die Auslastung nie kritische Werte erreicht. Es gibt lediglich einen Zeitpunkt, zu dem sie mehr als 50% beträgt. Dennoch können darin Optimierungsmöglichkeiten bestehen. Denn wie in Abschnitt 1.1 geschildert, setzen sich die Betriebskosten eines Rechenzentrums hauptsächlich aus den Kosten für Hardware und Strom zusammen. Wenn man die Dienste neu auf eine geringere Serveranzahl verteilen würde, würde zwar die Auslastung steigen, überflüssige Server könnten aber abgeschaltet werden und die kosten so gesenkt. Allerdings muss berücksichtigt werden, dass die Bundesagentur für Arbeit als Körperschaft des öffentlichen Rechts kein gewinnorientiertes Unternehmen ist. Die IT-Infrastruktur ist so aufgebaut, dass hohe Auslastungen oder gar Ausfälle sehr unwahrscheinlich sind.

Zu der Zeitreihe der CPU-Auslastung wurden dann noch einige Kennzahlen berechnet, die in Tabelle 4.2 aufgeführt werden. Auffällig sind das arithmetische Mittel, das man aufgrund des Plots vermutlich höher erwartet hätte. Außerdem zeigt das 75%-Quantil, dass 75% der Werte kleiner sind als die Hälfte des arithmetischen Mittels. Die Auslastung ist also meistens sehr gering und schon die Werte von etwa 10%, die in Abbildung 4.4 auffallen, stellen in diesem Kontext eine verhältnismäßig hohe Auslastung dar.

4.8 Datenmodellierung

Die Datenmodellierung wird in Kapitel 6 beschrieben.

4.9 Visualisierung

Die Visualisierung wird in Kapitel 8, speziell ab 8.4 geschildert.

In diesem Kapitel wurden die Daten einer Vorverarbeitung unterzogen, sodass nun

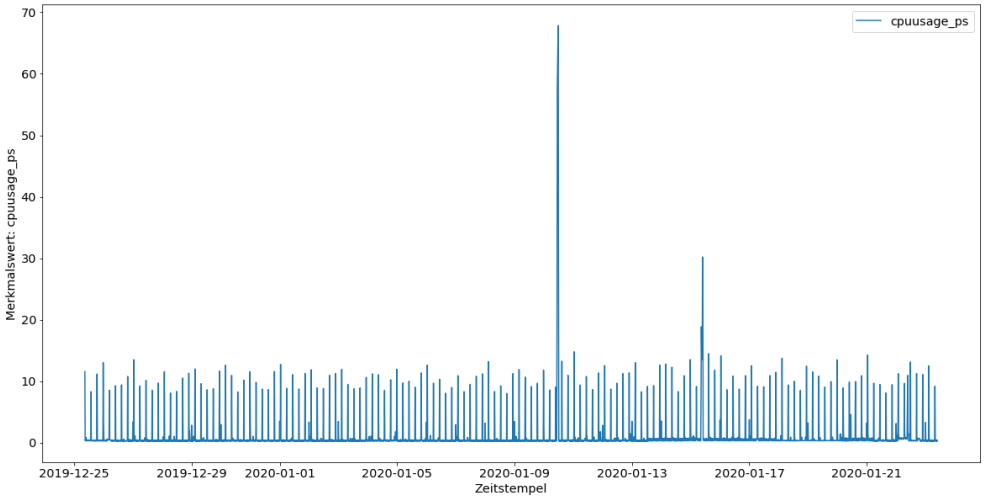

Abbildung 4.4: Plot der CPU-Auslastung aus dem vierwöchigen Datensatz. Verwendete Metrik: cpuusage_ps.

Prognosen erstellt werden können. Die Grundlagen dafür und Entscheidungen dazu werden im Folgenden Kapitel erläutert.

Kapitel 5

Grundlagen für die Prognosenerstellung mit Machine-Learning

Nachdem in Kapitel 2 bereits die Zeitreihenanalyse aus der Sicht der Statistik betrachtet wurde, wird in diesem Kapitel die Zeitreihenprognose mit Machine-Learning vorgestellt. Zunächst wird eine allgemeine Definition für Machine-Learning formuliert. Anschließend wird die Entscheidung für die Anwendung von Machine-Learning begründet. Daraufhin werden die verschiedenen Machine-Learning-Klassen vorgestellt und im Anschluss einige Grundlagen zu Künstlichen Neuronalen Netzen und deren Funktionsweise dargelegt.

5.1 Was ist Machine-Learning?

Machine-Learning könnte nach [82, S. 10] allgemein definiert werden als „ein Algorithmus, der durch gesammelte Erfahrung über die Zeit seine Leistungsfähigkeit, eine bestimmte Aufgabe zu erledigen, verbessert." Die Aufgabe, die der Algorithmus, also das Künstliche Neuronale Netz, in diesem Fall erledigen soll, ist die Vorhersage einer Zeitreihe. Die Erfahrung ergibt sich aus den gesammelten Daten, die für das Training verwendet werden. Und die Leistungsfähigkeit wird anhand der Genauigkeit der Vorhersagen gemessen.

5.2 Statistische Methoden oder Machine-Learning?

Der erhoffte Vorteil des maschinellen Lernens gegenüber klassischer statistischer Methoden zur Zeitreihenvorhersage ist, dass die Zusammenhänge zwischen mehreren Features gelernt werden und somit bessere Vorhersagen getroffen werden können. Sowohl [63] als auch [61] kommen zu dem Schluss, dass Machine-Learning-Methoden für Zeitreihenvorhersagen bei Datensets mit nur einem untersuchten Merkmal mit weniger

Aufwand zu besseren Prognosen führen. Sie stellen allerdings in Aussicht, dass die Ergebnisse bei anderen Datensätzen anders ausfallen hätten können. Die Autoren von [63] sehen bei der Verwendung von Machine-Learning zudem als Problem, dass keine Ungenauigkeiten beziehungsweise Konfidenzintervalle für die Vorhersagen angegeben werden. Es ist nicht ersichtlich, wie weit die tatsächlichen Werte um die Vorhersage schwanken könnte. In dieser Arbeit sollte speziell Machine-Learning angewendet werden, um die Prognosen zu treffen. Allerdings zeigen die oben genannten Artikel, dass es sinnvoll wäre, einen Vergleich zu klassischen statistischen Methoden bei dem vorliegenden Datensatz anzustellen. Da in dem vorliegenden Datensatz nicht nur die Metrik, die vorhergesagt werden soll, enthalten ist, sondern ein multivariater Datensatz existiert, könnte ein vektorisiertes ARMA-Modell (VARMA) verwendet werden. Diese lassen multivariate Eingangsdaten in Form von Vektoren zu [52].

5.3 Machine-Learning-Klassen

Machine-Learning-Algorithmen lassen sich prinzipiell in drei verschiedene Klassen kategorisieren: Supervised Learning, Unsupervised Learning und Reinforcement Learning. Welche Klasse angewandt wird, hängt von der Problemstellung beziehungsweise dem Ziel eines Projektes ab. Im Folgenden wird ein kurzer Überblick über die Begriffe gegeben und schließlich die für dieses Vorhaben passende Klasse gewählt.

5.3.1 Supervised Learning

Das Supervised Learning basiert auf dem Prinzip, mit den vorhandenen Daten ein Vorhersagemodell zu trainieren, mit dem anschließend neue Daten bewertet und so Prognosen erstellt werden können [76, S.26 f.]. Entscheidend dafür ist nach [76, S.26 f.], dass die Ausgangsdaten bereits die zu vorhersagenden Werte enthalten. Innerhalb der Supervised Learning Klasse unterscheidet man zusätzlich zwischen Klassifizierung und Regression [76, S.26 f.]. Bei der Klassifizierung sollen die neuen Daten in eine der Klassen bzw. Gruppen eingeteilt werden, auf die das Modell mit den Trainingsdaten trainiert wurde [76, S.26 f.]. Dafür muss in den Trainingsdaten bereits die Klasse enthalten sein, zu der ein Eingabebeispiel gehört. Anwendungsbeispiele liegen in den Bereichen der Bilderkennung oder Optical-Character-Recognition (OCR). Soll beispielsweise ein Modell trainiert werden, das erkennt, ob auf einem Bild ein Hund vorhanden ist, ist es nötig, die Trainingsbilder mit dieser Information zu markieren. Das Ziel der Regression dagegen ist es, anhand von Regressor-Variablen und einer Ziel-Variable, ein Modell zu trainieren, das die Zusammenhänge zwischen diesen Variablen erkennt, und so für neue Werte Prognosen treffen kann [76, S.28 f.].

5.3.2 Unsupervised Learning

Anders als beim Supervised Learning sind beim Unsupervised Learning keine Bezeichner in den Ausgangsdaten vorhanden. Stattdessen wird durch das Trainieren des Modells die Struktur der Daten analysiert und daraus Informationen gewonnen [76, S. 31]. Ein Beispiel für Unsupervised Learning sind nach [99, S. 31] sogenannte „Deep

Autoencoder“, die zum Beispiel zur Suche nach ähnlichen Bildern verwendet werden können. Das Prinzip eines Deep Autoencoders besteht dabei darin, den Input zunächst zu kodieren und so eine komprimierte Version zu erstellen [99, S. 31]. Das Lernen geschieht dann, indem der Autoencoder versucht, aus der komprimierten Version das Original zu dekodieren [99, S. 31].

5.3.3 Reinforcement Learning

Beim Reinforcement Learning soll die Interaktion eines sogenannten Agenten mit seiner Umgebung optimiert werden, indem dessen Aktionen beurteilt werden [76, S.29 f.]. Die Bewertung wird durch eine Belohnungsfunktion realisiert [76, S.29 f.]. Handlungen, die zu größeren Belohnungen führen, werden dabei zukünftig favorisiert. Ein Anwendungsbeispiel sind Schachcomputer [76, S.29 f.]. Hier lassen sich zudem Parallelen zu „natürlicher“ Intelligenz ziehen. Der Psychologe Edward Lee Thorndike formulierte nach seinen Tierstudien das „Gesetz der Wirkung“ [17, S. 62 ff.]. Dieses besagt, dass Tiere durch Reiz-Reaktions-Ketten, die zu neuen Neuronenverknüpfungen im Gehirn führen, lernen [17, S. 62 ff.]. Wirkt sich das Ergebnis einer Handlung für das Individuum positiv aus, wird das Verhalten in Zukunft mit höherer Wahrscheinlichkeit wiederholt. Ist das Gegenteil der Fall, wird die neuronale Verknüpfung geschwächt [17, S. 62 ff.]. Das Resultat einer Aktion wirkt sich demnach in Zukunft auf die Aktion selbst aus.

5.3.4 Entscheidung für eine Klasse

Das hier entwickelte System soll Zeitreihenprognosen treffen. Dabei soll kein Feedback vom Nutzer eingeholt werden. Die Vorhersagegenauigkeit bis zum Zeitpunkt t kann bei erreichen dieses Zeitpunktes mit den tatsächlichen Messungen verglichen werden und so automatisiert ein Feedback erzeugt werden. Die Klasse Reinforcement Learning wird daher ausgeschlossen. Zeitreihenvorhersagen können als ein Supervised-Machine-Learning Problem betrachtet werden und der Regression zugeordnet werden. Allerdings bedarf es bei der Wahl eines Algorithmus besonderer Sorgfalt. Denn bei Zeitreihen hängt der Wert x_{t+1} zum Zeitpunkt $t+1$ oft vom Wert x_t ab. Zudem ist daher die Verarbeitungsreihenfolge von Bedeutung und muss parallel zur Zeitreihe eingehalten werden [76, S. 529 ff.]. Für diesen speziellen Fall eignen sich besonders sogenannte „Recurrent Neural Networks“ [76, S.532 ff.], die in Abschnitt 6.1.3 erläutert werden.

5.4 Künstliche Neuronale Netze

Ein Künstliches Neuronales Netz (KNN) ist der Versuch, die Neuronennetze im menschlichen Gehirn nachzubilden. KNNs sind eine Teilmenge des Machine-Learning. Ein künstliches Neuron modelliert die Funktionsweise eines biologischen Neurons [9, S. 40]. Die Synapsen werden durch Verbindungen zwischen den Neuronen modelliert und erhalten Aktivierungen sowie Gewichtungen [9, S. 40]. Diese werden verwendet um die biologische Funktionsweise technisch umzusetzen. Die Summe der gewichteten

Aktivierungen bildet die Ausgabe des Neurons und ist das Pendant zum biologischen Axon [9, S. 40].

5.4.1 Das Perzeptron nach Rosenblatt

Künstliche Neuronale Netze gehen zurück auf die Idee des Perzeptrons, die 1958 in [79] veröffentlicht wurde und im Folgenden erklärt werden soll. Sie basiert auf den Erkenntnissen der damaligen Zeit, dass Neuronen und Computer funktional ähnlich sind, und den daraus entstandenen Gehirnmodellen, die aus algorithmischen Reaktionen auf Reizfolgen bestehen [79]. Bereits in den 1940-er Jahren wurden Neuronen mit mathematischen Modellen beschrieben [99, S. 11].
Ein Perzeptron besteht zunächst aus Sensoreinheiten („S-points“), die Impulse an die Assoziationszellen („A-units“) senden. Diese A-units bilden die Projektionsfläche A_I. Jede A-unit in A_I erhält eine bestimmte Anzahl an unidirektionalen Verbindungen von den S-points, über die deren Impulse weitergereicht werden. Übersteigt die Summe der Impulse den Schwellenwert θ der A-unit ist sie „aktiv“ und feuert wiederum einen Impuls an die Assoziationsfläche A_{II}. Die Verbindungen von A_I zu A_{II} sind im Gegensatz zu den Verbindungen der S-units zu A_I zufällig. Ansonsten sind die A-units in A_I und A_{II} identisch. Schließlich gibt es eine bidirektionale Verbindung zwischen A_{II} und den Ergebniszellen R_i. Diese funktionieren wiederum ähnlich zu den A-units, sind allerdings wechselseitig ausschließend - es kann also immer nur eine Ergebniszelle aktiv sein [79]. Abbildung 5.1 zeigt den Aufbau eines Perzeptrons.

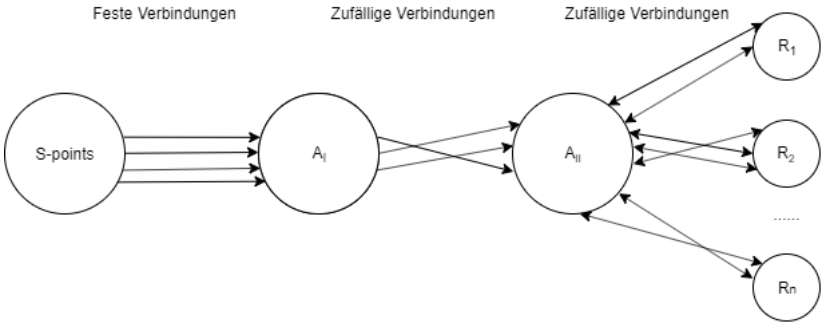

Abbildung 5.1: Das Perzeptron nach Rosenblatt [79]. Es ist aus den S-Points, A-units und R-Zellen sowie deren Verbindungen untereinander aufgebaut.

In [9, S. 112 f.] wird ein Beispiel angeführt: Wenn dieses Perzeptron nun eine Klassifikation lösen soll, konkret ein Bild in eine von zwei Klassen zuteilen, kann man ein Lernprinzip formulieren. Man verwendet das Bild als Eingabe für die S-points. Die Ausgabe der A_{II} sei x und die Verbindung zwischen A_{II} und den Ergebniszellen R_i habe die Gewichtung w, die zu Beginn zufällig initialisiert werden. Man prüft dann, ob die Ausgabe des Neurons das Bild richtig klassifiziert. Dann passt man die Gewichtung so lange an, bis alle Bilder korrekt klassifiziert werden. So kann die Perzeptron-Lernregel nach [9, S. 113] wie folgt definiert werden:

$$w_t = w_{t-1} + \gamma x. \tag{5.1}$$

Dabei wird durch γ die Anpassung der Gewichtung w_t zum Verarbeitungsschritt t

erreicht. Mit Index t ist nicht der Zeitstempeln einer Zeitreihe gemeint, sondern die aufeinanderfolgenden Lernschritte des Perzeptrons.

5.4.2 Aufbau von KNNs

Ein Künstliches Neuronales Netz (KNN) besteht aus mindestens drei Schichten: Der Eingabe-, Ausgabe und den dazwischen liegenden 1..n verdeckten Schichten. Besitzt ein KNN mehr als eine verdeckte Schicht, bezeichnet man es als ein tiefes Neuronales Netz. Man spricht dann auch von Deep Learning, das somit eine Untermenge des Machine-Learning darstellt. Hier lässt sich die Parallele zur Perzeptron-Idee aus dem vorherigen Abschnitt erkennen. Die Eingabe-Schicht entspricht den S-points, die verdeckten Schichten den Assoziationsflächen A_I und $A_I I$ und die Ausgabeschicht den R-Zellen. Ein Neuron in einer verdeckten Schicht ist demnach das Pendant zu einer A-unit. Abbildung 5.2 zeigt einen beispielhaften Aufbau.

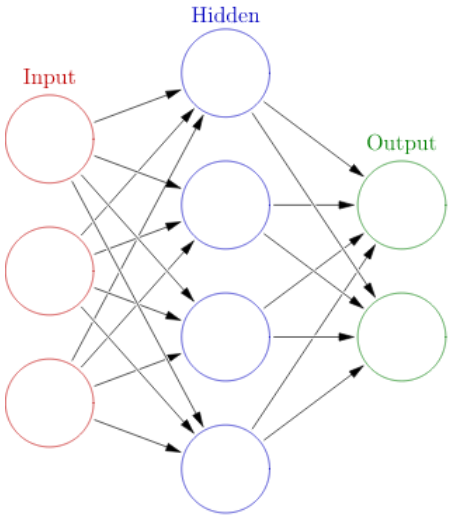

Abbildung 5.2: Schematischer Aufbau eines Künstlichen Neuronalen Netzes aus [82, S. 31]. Es gibt eine Eingabeschicht *Input-Layer*, eine verdeckte Schicht *Hidden-Layer* und eine Ausgabeschicht *Output-Layer*. Die Schichten sind vollständig verknüpft. Jedes Neuron der Eingabeschicht hat eine Verknüpfung zu jedem Neuron der verdeckten Schicht. Und jedes dieser verdeckten Neuronen eine Verbindung zu jedem Neuron der Ausgabeschicht.

Die Berechnungsfunktion eines Neurons besteht aus einem linearen Teil und einem nicht-linearen Teil [8, S. 7 f.]. Die Ausgabe s_i des Neurons i kann geschrieben werden als:

$$s_i = f_i(\sum_j w_{ij} * s_j - \theta_i) \tag{5.2}$$

[8, S. 7].

Man wendet zur Berechnung also die Funktion f_i auf den linearen Teil $net_i := \sum_j w_{ij} * s_j - \theta_i$ an. Dabei ist w_{ij} die sogenannte Gewichtung der Verbindung zwischen den

Neuronen j und i und θ_i der Schwellenwert des Neurons i [8, S. 7 f.], der auch Bias genannt wird.

Um den linearen Teil des Neurons i zu berechnen, wird also von allen Neuronen j, die eine Verbindung zu i haben, die Gewichtung w_{ij} dieser Verbindung mit dem Ausgabewert s_j des Neurons j multipliziert. Davon wird der Schwellenwert θ_i abgezogen. Dieser Schwellenwert, der entscheidet, ob das Neuron aktiv ist, findet sich ebenfalls bereits bei der Perzeptron-Idee. Die Ausgabe eines Neurons hängt offensichtlich von den Ausgaben der vorherigen Neuronen ab. Für den nicht-linearen Teil f_i, oft auch Aktivierungsfunktion genannt, können verschiedene Funktionen verwendet werden. Beispielsweise eine „Sigma“ oder „Stufen“-Funktion, die abhängig von einem Schwellenwert x entweder 1 oder 0 liefert und beim in Abschnitt 5.4.1 behandelten Perzeptron verwendet wird:

$$\sigma(x) := \begin{cases} 1 & \text{für } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{5.3}$$

Oder die logistische Funktion, die auch „Sigmoid-Funktion“ genannt wird:

$$f_{log}(x) := \frac{1}{1 + e^{-x}} \tag{5.4}$$

Dabei ist e die Eulersche Zahl. Der Wertebereich dieser Funktion liegt zwischen 0 und 1 [8, S. 8]. Abbildung 5.3 zeigt die S-Form der Funktionskurve.

Eine ähnliche, sigmoide Funktion ist der Tangens hyperbolicus *tanh* (Abbildung 5.4), der sich durch lineare Transformation aus f_{log} auf folgende Weise ergibt:

$$tanh(x) := 2f_{log}(2x) - 1 = 2(\frac{1}{1 + e^{-2x}}) - 1 \tag{5.5}$$

[8, S. 19].

Der größte Unterschied zwischen dem $tanh(x)$ und der $f_{log}(x)$ ist ihr Wert für $x = 0$: $tanh(0) = 0$ aber $f_{log}(0) = 0.5$.

Eine weitere Möglichkeit für die Aktivierungsfunktion besteht in der „Rectified Linear Unit“ (ReLU)-Funktion, die in Abbildung 5.5 dargestellt wird. Diese verhält sich für Eingaben $x \geq 0$ wie die Identitätsfunktion und gibt ansonsten 0 zurück:

$$ReLU(x) := \begin{cases} x & \text{für } x \geq 0 \\ 0 & \text{sonst} \end{cases} \tag{5.6}$$

[99, S. 13].

Um mit einem KNN Zeitreihenvorhersagen zu erzeugen, muss das Netz lernen, welche Eingaben zu welchen Ausgaben führen sollen. Wenn man dies abstrahiert betrachtet, versuchen die Netze, ähnlich zu den klassischen statistischen Ansätzen aus Kapitel

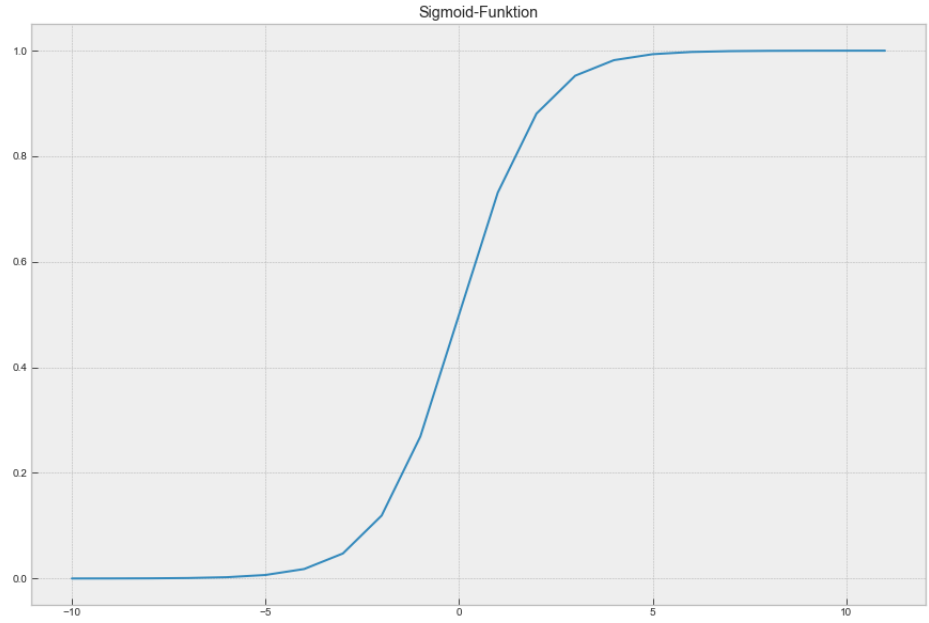

Abbildung 5.3: Die Sigmoid-Funktion.

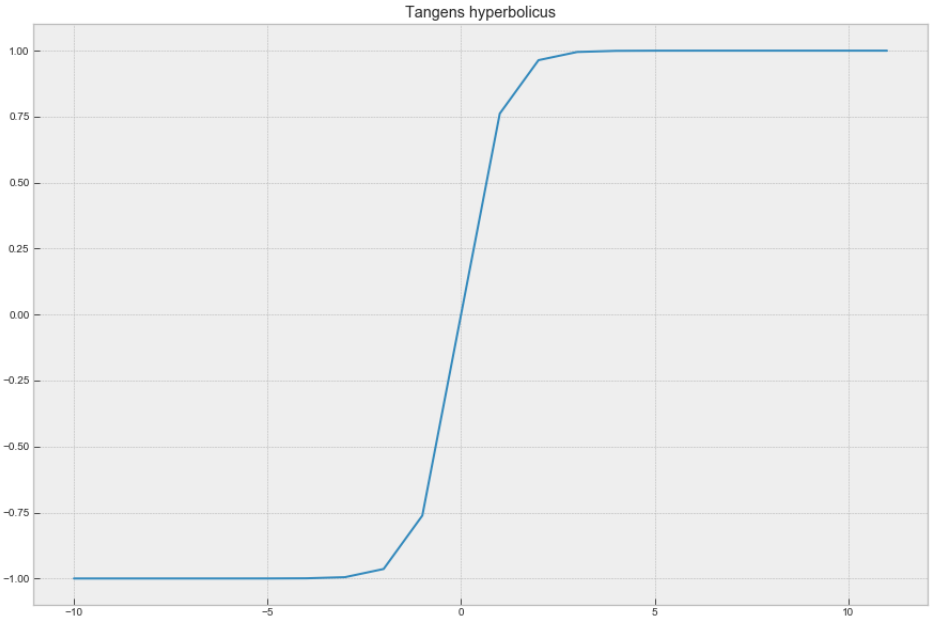

Abbildung 5.4: Der Tangens hyperbolicus.

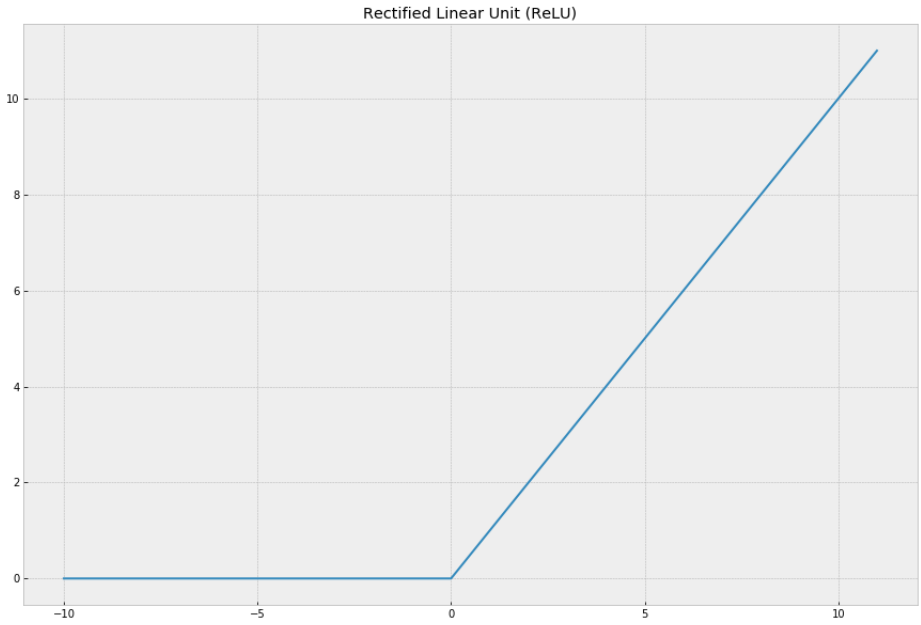

Abbildung 5.5: Die ReLU-Funktion, die entweder null oder die Eingabe zurückgibt.

2, die „Regeln“ zu erlernen, die zu der Zeitreihe geführt haben. Dabei soll diese Bildungsvorschrift als eine Reihe mathematischer Operationen erklärt werden. Allerdings werden bei den Neuronalen Netzen wie eben erläutert nicht-lineare Aktivierungsfunktionen verwendet. Hat das Neuronale Netz die Regeln gelernt, ist es möglich, Vorhersagen zu erzeugen, indem die Bildungsvorschrift auf eine Eingabe mit unbekanntem Resultat angewendet wird. Je besser das KNN die Regeln erkannt hat, desto genauer ist die Vorhersage.

5.4.3 Lernen mittels Gradientenabstiegsverfahren

Nach der Definition in Abschnitt 5.1 sollte ein Künstliches Neuronales Netz durch Erfahrung „lernen“ können. Die Wahrscheinlichkeit, dass die Ausgabe fehlerhaft ist, also reduziert werden. Dies wird erreicht, indem eine sogenannte Zielfunktion optimiert wird. Im Bereich des Deep Learning möchte man meist einen Fehler minimieren, und spricht daher auch von der Kosten- oder Verlustfunktion [34, S. 90]. Prinzipiell ermöglicht das Nullsetzen der Ableitung einer Funktion das Finden von sogenannten stationären Punkten dieser Funktion. Diese können dann entweder ein lokales Minimum, lokales Maximum oder ein Sattelpunkt sein. Den absolut niedrigsten Wert der Funktion nennt man globales Minimum. Beim Minimieren der Kostenfunktion sucht man also eigentlich ein globales Minimum. Da das Finden eines globalen Minimums allerdings schwierig ist, reicht es für Deep Learning Algorithmen, ein möglichst gutes lokales Minimum zu finden [34, S. 91 f.]. Da die Funktionen im Deep Learning Kontext normalerweise mehrere Variablen haben, müssen die partiellen Ableitungen $\frac{\partial}{\partial x_i} f(x)$

verwendet werden [34, S. 92], wobei $f(x)$ nach x_i abgeleitet wird. Vereint man die Menge aller partiellen Ableitungen in einem Vektor, nennt man diesen den Gradienten $\nabla_x f(x)$ von $f(x)$ [34, S. 92]. Dabei ist das Element an der Stelle i die Ableitung $\frac{\partial}{\partial x_i} f(x)$ [34, S. 92]. Ein solcher Gradient zeigt immer in die „Richtung der größten Funktionszunahme“ [81, S. 555]. Da $f(x)$ aber minimiert werden soll, ist es nötig, die Funktion in die entgegengesetzte Richtung, also entlang des negativen Gradienten, anzupassen. Dieses Vorgehen wird als Gradientenabstiegsverfahren oder Gradientenverfahren bezeichnet. So lässt sich der Punkt x', der näher am lokalen Minimum liegt als x und somit „besser“ ist, nach [34, S. 92 f.] berechnen als:

$$x' = x - \epsilon \nabla_x f(x), \tag{5.7}$$

wobei ϵ die sogenannte Lernrate ist, die bestimmt, wie groß der Schritt entlang des negativen Gradienten ist [34, S. 92 f.]. ϵ sollte so gewählt werden, dass $\epsilon > 0$ ist, jedoch einen „geringen“ Betrag hat [34, S. 92 f.]. Über die Gleichung (5.7) werden die Gewichtungen basierend auf dem Gradienten aktualisiert. Diese Berechnungen werden in mehreren Iterationen durchgeführt, mit dem Ziel, dass sie bei dem globalen Minimum konvergieren. Allerdings wird die Konvergenz bei fester, geringwertiger Lernrate ϵ nur langsam erreicht, da die Verbesserungen auch nur in kleinen Schritten geschehen. Dies gilt besonders für Neuronale Netzen mit mehr als einer verdeckten Schicht [106]. Das liegt laut den Autoren daran, dass der Betrag des Gradienten sehr klein werden kann. Eine große Lernrate hingegen könnte zur Divergenz führen [106]. Dies zeigt, dass bei diesem iterativen Verfahren der Wert der Lernrate großen Einfluss auf das Neuronale Netz hat und er somit einen wichtigen Hyperparameter darstellt [82, S. 263]. Ein Hyperparameter ist eine Konfigurationsmöglichkeit für ein Neuronales Netz, die nicht bei den verwendeten Daten besteht. Dazu gehören beispielsweise der Betrag der Lernrate, die Aktivierungsfunktion oder die Netztiefe. [34, S. 477] empfiehlt sogar, die Lernrate zu optimieren, falls nur ein einziger Hyperparameter angepasst werden kann. Die Begründung dafür ist, dass die Performanz des KNNs dann am besten ist, wenn die Lernrate für den konkreten Anwendungsfall richtig ist - unabhängig davon, ob ihr Betrag groß oder klein ist. Eine Anpassungsmöglichkeiten besteht nach [9, S. 71] in der dynamischen Änderung der Lernrate, bei der diese mit der Annäherung an das Minimum verkleinert wird. Alternativ weißt der Autor auf darauf hin, dass eine Matrix aufgestellt werden kann, die in der Hauptdiagonale für jede Gewichtung eigene Lernraten und ansonsten nur Nullen enthält. Eine weitere Optimierungsmöglichkeit besteht im Verwenden der zweiten Ableitung. In [106] werden dazu mehrere Möglichkeiten vorgestellt. Außerdem präsentieren die Autoren ein Verfahren um die zweite Ableitung zu erhalten, ohne die Rechenkomplexität ausschlaggebend zu verändern.

Optimierungsalgorithmen zweiter Ordnung

Algorithmen, die lediglich auf Basis des Gradienten funktionieren heißen auch „Optimierungsalgorithmen erster Ordnung“ [34, S. 99]. Optimierungsalgorithmen der zweiten Ordnung verwenden dagegen zusätzlich die Hesse-Matrix $H(f)(x)$, die alle

zweiten Ableitungen enthält: $H(f)(x)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(x)$ [34, S. 99 f.]. Ein Beispiel dafür ist das sogenannte Newton-Verfahren [34, S. 98 f.].

Stochastisches Gradientenabstiegsverfahren

Die Rechenkomplexität für das oben beschriebene Gradientenabstiegsverfahren beträgt $O(m)$ [34, S. 168], wobei m die Anzahl der Trainingsdaten ist. Denn für die Berechnung des Gradienten muss über m Terme summiert werden:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(x_i, y_i, \theta) \tag{5.8}$$

[34, S. 168].

Dabei ist x_i die Eingabe, y_i die Ausgabe, θ die Parametrisierung und $J(\theta)$ die Verlustfunktion:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x_i, y_i, \theta) \tag{5.9}$$

[34, S. 168].

$L(x, y, \theta)$ ist der Fehler oder „Velust“, der zum Beispiel als negative Log-Likelihood oder Mean Squared Error (MSE, siehe Gleichung (6.15)) berechnet werden kann [34, S. 168]. Die Berechnung des Gradienten unter Verwendung aller Trainingsdaten wird auch als „Batch-“Gradientenabstiegsverfahren bezeichnet [34, S. 310].

Das Stochastische Gradientenabstiegsverfahren (Stochastic Gradient descent, SGD) basiert auf der Tatsache, dass der Gradient den Erwartungswert der Trainingsdaten darstellt [34, S. 168]. Dieser Erwartungswert, also der Gradient, wird beim SGD aufgrund von Stichproben, den sogenannten „Mini-Batches“, geschätzt [34, S. 168]. Diese Mini-Batches enthalten m' Trainingsdaten [34, S. 168]. m' wird dabei fest auf einen Wert gesetzt, der kleiner ist als die Anzahl der Trainingsdaten m, zum Beispiel 256, und ändert sich auch bei wachsender Trainingsdatenmenge nicht [34, S. 168]. Der geschätzte Gradient g lässt sich dann analog zu Gleichung (5.8) berechnen als:

$$g = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L(x_i, y_i, \theta) \tag{5.10}$$

[34, S. 168].

Da der Gradient nur geschätzt wird, kommt es zu einer Abweichung vom tatsächlichen Gradienten. Der Standardfehler, beziehungsweise die Standardabweichung, eines aufgrund von Stichproben geschätzten Erwartungswertes lässt sich allgemein berechnen als:

$$SE(\hat{\mu}_m) = \sqrt{Var\left[\frac{1}{m} \sum_{i=1}^{m} x_i\right]} = \frac{\sigma}{\sqrt{m}}, \tag{5.11}$$

wobei m die Anzahl der Stichproben und x_i die i-te Stichprobe ist [34, S. 141]. Es lässt sich erkennen, dass die Stichprobengröße m nur als $\sqrt{m}$ eingeht. Eine Erhöhung

der Stichprobenanzahl um den Faktor 100 führt nach Gleichung (5.8) also zu einem 100-fachen Rechenaufwand, der Standardfehler der Schätzung reduziert sich allerdings nur um den Faktor $\sqrt{100} = 10$.
Bei der Wahl der Größe der Mini-Batches müssen nach [34, S. 311] mehrere Aspekte berücksichtigt werden: Größere Batches führen zu einer genaueren Schätzung, erhöhen allerdings auch die Laufzeit. Eine parallele Verarbeitung der Trainingsdaten in einem Batch führt mit wachsender Größe allerdings auch zu wachsenden Anforderungen an den Arbeitsspeicher. Wird die Batch-Größe zu gering gewählt, werden die möglichen Rechenressourcen nicht optimal ausgenutzt. Werden GPUs verwendet, sollte die Anzahl der Trainingsdaten pro Batch zudem einer 2er-Potenz entsprechen, um die Berechnungsdauer zu reduzieren.

Zum Teil versteht man unter dem Begriff Stochastisches Gradientenabstiegsverfahren auch nur den Sonderfall mit der Mini-Batch-Größe $m = 1$ und bezeichnet das Verfahren für andere m als „Mini-Batch-Gradienten(abstiegs)-Verfahren“. In [89, S. 8 f.] werden die Begriffe beispielsweise auf diese Art verwendet.

5.4.4 Backpropagation

Das Gradientenabstiegsverfahren beziehungsweise SGD werden angewandt, um dem KNN das Lernen zu ermöglichen. Dafür ist es, wie oben dargelegt, nötig, Gradienten zu berechnen. Der Backpropagation-Algorithmus dient zur Berechnung dieser Gradienten. Während man unter der Forwardpropagation versteht, dass die Eingabedaten von der Eingabeschicht über die verdeckten Schichten bis hin zur Ausgabeschicht nach vorne propagiert werden, geht der Datenstrom bei der Backpropagation rückwärts von der Ausgabeschicht zur Eingabeschicht [34, S. 225].

Der Algorithmus beruht auf der Kettenregel der Analysis, mit der die Ableitung einer Funktion $z = f(g(x)) = f(y)$ über die Ableitung der Funktion $y = g(x)$ bestimmt werden kann:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{5.12}$$

[34, S. 228].

Die Gleichung (5.12) lässt sich auch für Funktionen in mehreren Variablen anpassen. Sei $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, $g\colon \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ und $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ die Jacobi-Matrix, die alle partiellen ersten Ableitungen enthält [34, S. 78], dann gilt nach [34, S. 228]

$$\nabla_x z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_y z. \tag{5.13}$$

Die Kettenregel wird dann rekursiv angewandt, um die Backpropagation umzusetzen. Das Künstliche Neuronale Netz wird nun als Berechnungsgraph dargestellt. Der Berechnungsgraph wird gebildet, indem Variablen mit beliebigem Inhalt als Knoten und Operationen als Kanten zwischen den Variablen, auf denen sie ausgeführt werden, dargestellt werden [34, S. 226 ff.]. Da bei der rekursiven Anwendung Teilausdrücke

mehrmals benötigt werden, gibt es durch Speichern dieser Ausdrücke Möglichkeiten zur Laufzeitoptimierung [34, S. 229 f.]. Eine erneute Berechnung kann so zu Gunsten der Laufzeit umgangen werden. Alternativ kann speicherschonend vorgegangen werden, indem die mehrmalige Auswertung in Kauf genommen wird. Dieses Vorgehen hängt von der Implementierung ab, wobei für komplexe Netze das Zwischenspeichern nach [34, S. 229] unerlässlich ist.

Backpropagation Through Time

Da in dieser Arbeit auch Rekurrente Neuronale Netze (RNNs, siehe Abschnitt 6.1.3) behandelt werden, ist es nötig, den Backpropagation Trough Time (BPTT) Algorithmus zu erklären. Die rekurrenten Kanten in diesen Netzen führen zu Schleifen im Berechnungsgraphen [34, S. 420 ff.]. Diese Schleifen werden aufgelöst, indem der Graph aufgefaltet wird [34, S. 421]. Da der Graph nach dieser Auffaltung keine Rekurrenz mehr enthält, kann auch bei RNNs der oben beschriebene Backpropagation-Alogrithmus angewendet werden. Man nenn ihn dann Backpropagation Through Time [34, S. 424]. Die Auffaltung der rekurrenten Funktion $s_t = f(s_{t-1}; \theta$ geschieht nach folgendem Prinzip, das auch in Abbildung 6.2 auf Seite 46 für ein RNN dargestellt wird:

$$s_3 = f(s_2; \theta) = f(f(s_1; \theta); \theta) \tag{5.14}$$

[34, S. 417].

5.5 Über- und Unteranpassung eines Künstlichen Neuronalen Netzes

Von einer Überanpassung (engl. „Overfitting“) spricht man, wenn das KNN bei den Trainingsdaten einen geringen Fehler erzeugt, das Anwenden auf die Testdaten allerdings zu großen Fehlern führt [76, S. 93 f.]. Überanpassung spricht für ein Modell, dass aufgrund zu vieler Parameter zu genau an die Trainingsdaten angepasst wurde und daher keine Generalisierung zulässt [76, S. 93 f.]. Das Modell ist zu komplex. Analog spricht man von einer Unteranpassung (engl. „Underfitting“), wenn das Modell nicht komplex genug ist. So gelingt es dem KNN nicht, die Trainingsdaten genügend zu modellieren, was auch bei den Testdaten zu großen Fehlern führt [76, S. 93 f.].

Der Fehler eines KNN lässt sich in Bias und Varianz unterteilen [**geman_neural_2008**]. Ein möglichst gutes Modell hat also einen Fehler mit gleichzeitig möglichst geringem Bias und möglichst geringer Varianz.

Beim Overfitting liegt eine große Varianz vor, während beim Underfitting ein großer Bias ursächlich ist [76, S. 93 f.]. Der Bias bezeichnet dabei einen systematischen Fehler des Modells, während die Varianz angibt, wie das Modell mit Zufälligkeit in den Trainingsdaten umgeht [76, S. 93 f.]. Mit anderen Worten ist der Bias der konstant vorhandene Fehler und die Varianz ein zufälliger Fehler, der unabhängig von der erwarteten Ausgabe ist [23].

Bias und Varianz stehen allerdings in einem Spannungsverhältnis. Die Varianz kann durch ein „Glätten“ der an die Daten angepassten Kurve verringert werden, wodurch allerdings der Bias steigt, da die Abweichung zu Ausschlägen im Kurvenverlauf dadurch größer wird. Dieses Verhalten ist als „bias/variance dilemma“ bekannt [**geman_neural_2008**] und wird in Abbildung 5.6 dargestellt.

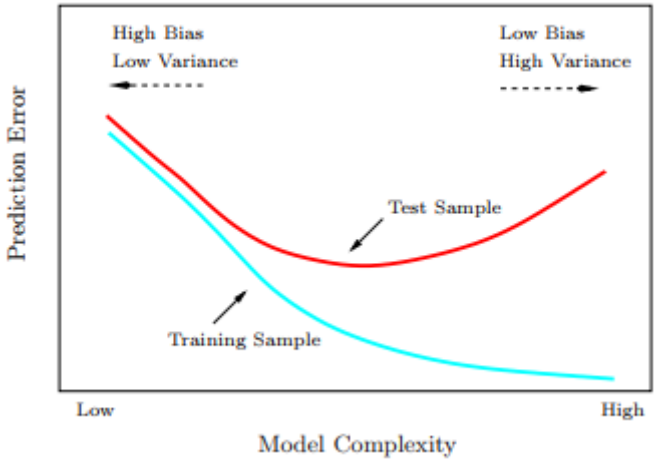

Abbildung 5.6: Das Bias/Variance dilemma aus [82, S. 288], das die Schwierigkeit beschreibt, gleichzeitig einen geringen Bias und eine geringe Varianz zu erreichen. Mit steigender Modell-Komplexität (x-Achse), sinkt der Trainingsfehler. Der Testfehler hat allerdings eine „U-Form“. Er ist bei geringer Modell-Komplexität hoch und besteht aus einem großen Bias (Underfitting). Er sinkt dann mit wachsender Komplexität. Bei zu großer Modell-Komplexität wächst der Testfehler allerdings wieder. Hier ist dann eine hohe Varianz ursächlich (Overfitting).

Die Komplexität eines KNNs lässt sich nach [50, S. 50] beispielsweise durch folgende Maßnahmen erhöhen:

- Anzahl der verdeckten Schichten erhöhen.
- Anzahl der Neuronen in den verdeckten Schichten erhöhen.
- Komplexe Aktivierungsfunktionen verwenden.
- Anzahl der Trainingsepochen (= Trainingsiterationen) erhöhen.

Eine Möglichkeit, Overfitting aufgrund der Modell-Komplexität zu verhindern, besteht in der Verwendung von Regularisierung [76, S. 94]. Diese werden im nächsten Abschnitt behandelt.

5.6 Regularisierung

Die Idee der Regularisierung ist es, extreme Gewichtungen durch einen Bias zu „bestrafen“ [76, S. 94 f.]. So kann verhindert werden, dass es zu einem Overfitting kommt. Ein Beispiel dafür ist die „L2-Regularisierung“, die sich wie folgt definieren

lässt:

$$\frac{\lambda}{2}||w||^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2 \tag{5.15}$$

Dabei ist λ der sogenannte „Regularisierungsparameter“ und w der Gewichtungsvektor mit m Elementen. Je größer λ ist, desto stärker ist die Regularisierung, also desto kleiner die Gewichtungen. Der Regularisierungsterm wird zur Kostenfunktion addiert [76, S. 94 f.].

Diese Regularisierung verwendet das Quadrat der Länge $||w||$ beziehungsweise des Betrags des Gewichtungsvektors w:

$$||w|| = \sqrt{\sum_{j=1}^{m} w_j^2} \tag{5.16}$$

[76, S. 97].

Eine Alternative ist der L1-Regularisierer, der die Gewichte nicht quadriert summiert, sondern deren Absolutbeträge als Summanden nimmt [76, S. 141].

5.6.1 Dropout-Regularisierung

Besonders für tiefe, also mehrschichtige, Künstliche Neuronale Netze bietet sich nach [76, S. 504 f.] die sogenannte Dropout-Regularisierung an. Dabei wird während des Trainings ein zufällig gewählter Teil der Neuronen deaktiviert. Wie groß dieser Anteil ist, wird durch die Dropwahrscheinlichkeit p_{drop} festgelegt. Wird mit einem mit Dropout-Schicht trainierten Modell schließlich eine Vorhersage getroffen, wird allerdings kein Dropout vorgenommen [76, S. 504 f.]. Die Aktivierungen der aktiven Neuronen müssen daher bei diesem Verfahren entsprechend der Wahrscheinlichkeit p_{drop} angepasst werden, um die Größenordnung zu erhalten [76, S. 504 f.]. Damit diese Modifikation der Aktivierungen nicht jedes mal beim Treffen einer Vorhersage durchgeführt werden muss, erledigen die gängigen Bibliotheken wie Tensorflow dies bereits während des Trainings [76, S. 504 f.]. Eine Dropout-Schicht wird meistens in den höheren verdeckten Schichten eingesetzt und ist ein gutes Mittel, um Overfitting zu verhindern. Für die Dropwahrscheinlichkeit wird ein Wert von $p_{drop} = 0.5$ empfohlen [76, S. 504 f.].

In Keras lässt sich eine solche Dropout-Schicht mit $p_{drop} = 0.5$ sehr einfach zu einem Modell hinzufügen, wie in Listing C.1 (siehe Anhang) dargestellt.

5.6.2 Regularisierung durch Early Stopping

Das *Early Stopping* (im Deutschen auch früher Abbruch) stellt eine einfache und effektive Regularisierung dar [34, S. 241 ff.] und wird im Folgenden beschrieben. Es basiert darauf, dass der Validierungsfehler mit voranschreitendem Training oft wieder zunimmt, während der Trainingsfehler weiter reduziert wird [34, S. 241 ff.]. Dies führt

letztendlich zu einer Überanpassung (siehe Abschnitt 5.5). Da ein geringerer Validierungsfehler für einen möglichst kleinen Testfehler wichtiger ist als der Trainingsfehler, wird während dem Training das Modell mit dem jeweils besten Validierungsfehler zwischengespeichert. Das gespeicherte Modell wird nur aktualisiert, wenn der Validierungsfehler in einer weiteren Trainingsiteration ein neues Minimum erreicht hat. Wenn nach einer vorher definierten Anzahl an Epochen keine Verbesserung aufgetreten ist, wird das Training vor dem Vollenden der eigentlich festgelegten Anzahl der Trainingsiterationen beendet. Daher ergibt sich der Name „Early Stopping“. Das aus dem Training resultierende Modell ist das mit dem geringsten Validierungsfehler. Somit ergibt sich durch das Anwenden des Early-Stoppings automatisch das richtige Maß an Regularisierung [34, S. 241 ff.].

In Keras lässt sich das Early Stopping durch ein `EarlyStopping`-Objekt implementieren. Durch ein `ModelCheckpoint`-Objekt kann das Modell während dem Training gespeichert werden. Die beiden Objekte können dann als Callback-Parameter beim Trainieren übergeben werden und so die Early Stopping Regularisierung umgesetzt werden. Für diese Arbeit wurde das Early Stopping so konfiguriert, dass das Training abbricht, wenn über 25 der 200 Epochen keine Modellverbesserung stattgefunden hat.

5.7 Standardisierung

Für die Regularisierung ist es wichtig, dass die Merkmale eine ähnliche Größenordnung haben [76, S. 94]. Auch das SGD lässt sich durch eine Standardisierung der Daten verbessern, da so die Verlangsamung des Verfahrens durch enge Täler im Gradienten umgangen wird [53, S. 124]. Standardisierung bedeutet, ein Merkmal in Normalverteilung zu übersetzen. Das Merkmal x mit j Einträgen kann mit folgender Formel standardisiert werden:

$$x'_j = \frac{x_j - \mu}{\sigma} \tag{5.17}$$

[53, S. 124]. Dabei ist x'_j das standardisierte Merkmal, μ das arithmetische Mittel und σ die Standardabweichung. Somit wird erreicht, dass das Merkmal einen Mittelwert von 0 und eine Standardabweichung von 1 hat. Dadurch können die Gewichtungen, die meist mit geringen Beträgen initialisiert werden, effizienter angepasst werden [76, S. 139]. Die Standardabweichung ist die Quadratwurzel der Varianz, die in Gleichung (2.4) auf Seite 5 definiert wird.

Neben der Standardisierung lässt sich auch eine Normierung auf den Wertebereich $[0, 1]$ durchführen. Dies kann durch eine sogenannte Min-Max-Skalierung in folgender Form erreicht werden:

$$x_{i_{norm}} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{5.18}$$

[76, S. 138 f.]. Dabei ist x_i der i-te Wert eines Merkmals und x_{max} beziehungsweise x_{min} sind das Maximum und Minimum der Merkmalswerte.

Aufgrund der Optimierungsmöglichkeiten beim SGD empfiehlt sowohl [76] als auch [53] die Verwendung der Standardisierung.

5.8 Univariate und multivariate Vorhersagen

Ein grundlegender Unterschied in der Vorhersagemethodik besteht darin, ob ein oder mehrere Merkmale als Eingabe verwendet werden. Wird nur ein Feature verwendet, nennt man dies „univariat“, ansonsten „multivariat“. Für eine univariate Prognose wird also nur der bisherige Verlauf der Zeitreihe als Eingabe verwendet. Für eine multivariate Vorhersage können zusätzlich weitere Zeitreihen oder anders generierte Features genutzt werden. Bei univariaten Vorhersagen gibt es also keine Beziehungen zu anderen Parametern, die das KNN erlernen könnte. Bei multivariaten Daten können Relationen zwischen den Parametern erkannt und zur Präzisierung der Vorhersage verwendet werden [82, S. 418 ff.]. Wie in Kapitel 5.2 dargelegt, soll die Anwendung von Neuronalen Netze für multivariate Daten Vorteile bringen.

Falls bei einem multivariaten Datensatz die Features unterschiedliche Größenordnungen haben, ist eine Standardisierung besonders wichtig. Ansonsten kann es passieren, dass die qualitativen Merkmale mit einem sehr großen Betrag das Modelltraining verzerren und so die Vorhersagen ungenauer werden [82, S. 239].

Nach der Datenvorverarbeitung, die in Abschnitt 4.7 beschrieben wurde, liegt der Datensatz multivariat vor und kann durch auswählen eines Features, respektive einer Spalte, leicht in einen univariaten Input verändert werden.

Da jetzt einige Grundlagen für die Verwendung von Neuronalen Netzen zur Zeitreihenprognose erläutert wurden, wird im nächsten Kapitel die passende Architektur ausgewählt und die Implementierung vorgenommen.

Kapitel 6

Zeitreihenprognose mit Neuronalen Netzen

Dieses Kapitel handelt von der Zeitreihenprognose mit Neuronalen Netzen. Zunächst werden verschiedene Architekturen Neuronaler Netze theoretisch evaluiert. Anschließend die Datenvorbereitung für die multivariate Vorhersage beschrieben. Daraufhin wird die Implementierung und Optimierung der Modelle erläutert. Schließlich wird die Vorhersagegenauigkeit der Modelle verglichen und basierend darauf die Entscheidung für eine Architektur getroffen.

6.1 Evaluation verschiedener Architekturen Neuronaler Netze

6.1.1 Multilayer Perceptron (MLP)

Ein Multilayer Perceptron (MLP) ist ein „gerichtetes, azyklisches“[8, S. 16] KNN. Azyklisch bedeutet in diesem Zusammenhang, dass der Datenfluss nur in einer Richtung, von der Eingabeschicht über die verborgene Schicht hin zur Ausgabeschicht, läuft. Die Struktur eines MLP lässt sich ähnlich zu Abbildung 5.2 auf Seite 31 abstrahieren. Ein MLP kann mehrere verdeckte Schichten haben. In einem solchen KNN sind die Neuronen vollständig verknüpft [8, S. 9]. Jedes Neuron n_i^j der Schicht j ist also mit jedem Neuron n_k^l der darauffolgenden Schicht l verbunden. Die Anzahl der Neuronen ist dabei nur bei der ersten auf die Eingabeschicht folgende verborgene Schicht und der letzten verborgenen Schicht festgelegt. Die Schichten können prinzipiell eine beliebige Anzahl an Neuronen enthalten, was für die Modelloptimierung von Bedeutung ist (siehe Kapitel 6.4). MLPs besitzen keine Eigenschaft, die sie für Sequenzen besonders geeignet machen. Sie können daher als Vergleichswert zu den spezialisierteren KNNs, die im Folgenden vorgestellt werden, verwendet werden.

6.1.2 Convolutional Neural Network (CNN)

Ein Convolutional Neural Network (CNN) ist ein KNN „das zur Verarbeitung von Daten mit einer bekannten rasterähnlichen Topologie“ [34, S. 369] eingesetzt wird. Der Name kommt von der Faltungs-(engl. *convolution*) Schicht. Meistens werden diese Netze für Bilddaten, die ein 2-dimensionales Raster aufweisen, verwendet. Jedoch ist es auch möglich, Zeitreihen mit regelmäßigem Messintervall als 1-D Raster zu verarbeiten [34, S. 369]. Damit werden sie für die der Arbeit zugrunde liegende Situation relevant.

Faltung

Jedes Neuronale Netz, das „in mindestens einer Schicht Faltung anstelle der allgemeinen Matrixmultiplikation“ [34, S. 369] einsetzt, ist ein CNN. Diese Faltung wird auf der Eingabefunktion $x(t)$ mit dem sogenannten Kernel $w(a)$ durchgeführt. Die nach der Zeit diskretisierte Faltung, die einen nach dem Alter a des Messwertes gewichteten Schätzwert liefert, lässt sich nach [34, S. 370 f.] schreiben als

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{6.1}$$

Die Anzahl der Verbindungen zwischen den Eingaben und Ausgaben der Faltung hängt von der Größe des Kernels ab. Da der Kernel normalerweise kleiner ist als die Eingabe, verringert sich durch die Anwendung der Faltung im Vergleich zur sonst typischen Matrizenmultiplikation die Anzahl der Verbindungen [34, S. 374]. Dadurch sinkt wiederum die Laufzeit. Man nennt dies „spärliche Konnektivität“ [34, S. 374]. Aufgrund des sogenannten „Parameter Sharing“ wird außerdem der Speicherbedarf reduziert [34, S. 374 f.]. Denn die Parameter werden in Form des Kernels für die gesamte Eingabe verwendet, weswegen sich der Speicherbedarf für die Parameter auf den Kernel beschränkt [34, S. 374 f.]. Eine weitere wichtige Eigenschaft, die sich daraus ergibt, ist die Äquivarianz gegenüber Verschiebungen [34, S. 377 f.]. Eine Verschiebung der Eingabe wirkt sich durch die Faltung also in gleicher Form auf die Ausgabe aus.

Da der Kernel nur eine gewisse Umgebung, entsprechend der Kernel-Größe, einbezieht, werden im Zusammenhang der Zeitreihen auch nur die Werte in dieser Umgebung um x_t berücksichtigt. Ein CNN lernt also eher kurzfristige Zusammenhänge zwischen den Zeitreihenwerten. Dieses Prinzip wird in Abbildung 6.1 veranschaulicht.

Es gibt einige Variationen dieser Faltung [34, S. 386 ff.]. Zwei der in [34, S. 386 ff.] aufgeführten Varianten werden im Folgenden erklärt.. Beispielsweise kann ein sogenannter „stride“ s verwendet werden. Beim Verschieben des Kernels werden dann s Werte übersprungen. Bei einem stride von beispielsweise $s = 2$ wird der Kernel nicht um einen Schritt, sondern um drei Schritte verschoben. Eine weitere Variation ist das Auffüllen des Inputs mit Nullen, das sogenannte „zero padding“. Dieses wird verwendet um zu verhindern, dass die Dimension des Inputs abnimmt. Denn wenn der Input zu klein ist, ist die Faltung nicht mehr möglich und das CNN würde so in seiner maximalen Tiefe beschränkt. Durch das Auffüllen mit Nullen am Anfang und am Ende des Inputs wird erreicht, dass die Dimension gleich bleibt und das Netz so

beliebig tief werden kann [34, S. 386 ff.]. Abbildung A.4 (siehe Anhang) visualisiert den Effekt des zero paddings.

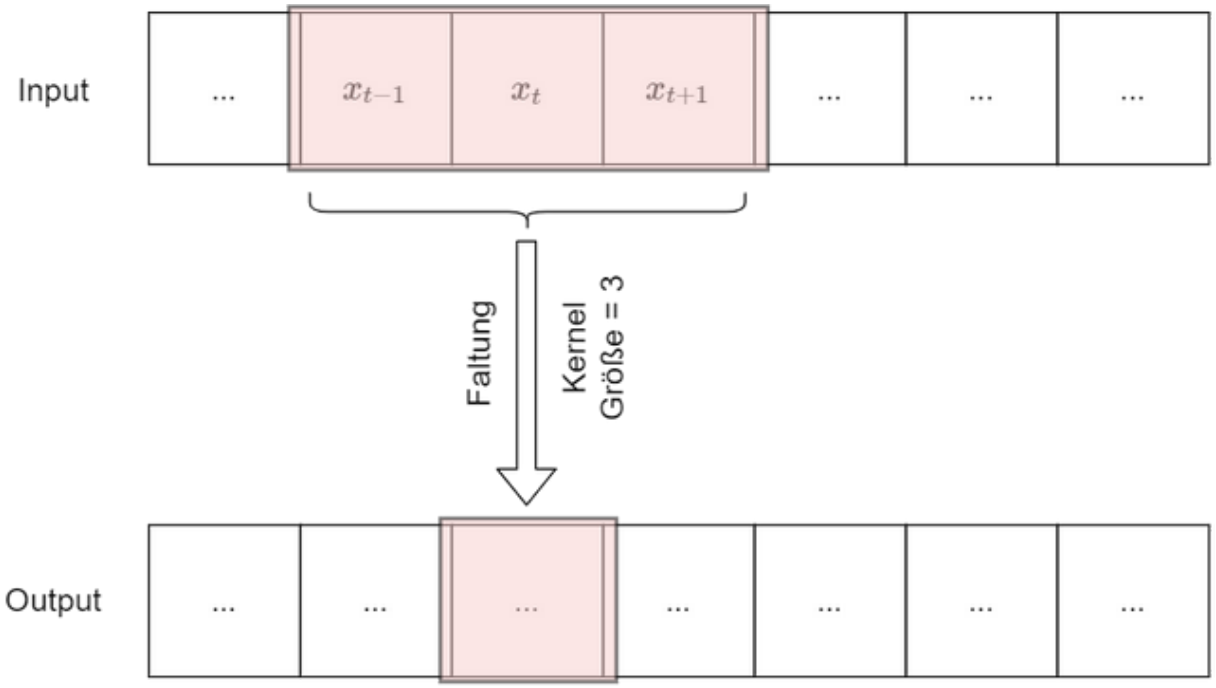

Abbildung 6.1: Prinzip der eindimensionalen Faltung für einen Kernel der Größe 3.

Pooling

Neben der Faltung wird bei CNNs im Allgemeinen auch das sogenannte „Pooling“ eingesetzt. Ziel des Poolings ist es, Invarianz gegenüber Verschiebungen in der Eingabe zu erreichen. Eine geringfügige Verschiebung im Input soll nur zu wenig Änderungen in den sogenannten „pooled outputs“ führen. Dafür können zum Beispiel der Maximalwert, der Durchschnittswert oder die L2-Norm einer gewissen Umgebung verwendet werden [34, S. 379 ff.]. Dabei wird ähnlich wie bei der Faltung die ausgewählte Funktion, beispielsweise der Maximalwert, auf eine gewisse Umgebung angewendet. Anschließend wird das „Fenster“ verschoben. Dabei entspricht die Verschiebung der Umgebungsbreite [53, S. 70 f.], sodass jede Eingabe nur genau einmal im Pooling verwendet wird.

Beispiele für die Anwendung von CNNs auf Zeitreihen

Auch wenn CNNs meistens im Bereich Computer Vision verwendet werden, gibt es auch Beispiele für die Anwendung auf Zeitreihen. Ein Beispiel für die Verwendung von CNNs zur Zeitreihenvorhersage liefert [88]. Die Autoren verglichen die Vorhersagegenauigkeit eines RNN, LSTM-Netzes, CNN sowie ARIMA für verschiedene Aktienpreise. In diesem Fall lieferte das CNN die beste Performanz. Die Autoren begründen dies mit den plötzlichen Veränderungen der Aktienpreise, die nicht zwingend einem Muster folgen. Hier haben aufgrund der hohen Volatilität kurzfristige Zusammenhänge eine größere Bedeutung, weswegen die Faltungsoperation gute Ergebnisse erzeugt. Auch

die Autoren von [38] verwendeten ein CNN zur Vorhersage von Preisen. Sie bildeten die Preisentwicklung von Strom-Verträgen als ein Klassifizierungsproblem mit je einer Klasse für - innerhalb eines Schwellwerts - gleichbleibende, steigende und fallende Preise ab. In [45] wurde ein CNN mit anderen Methoden verglichen, um Lokalisierung im Raum anhand von received signal strength-(RSS) Werten des WiFi-Signals zu ermöglichen. Das CNN erzielte in diesem Vergleich die besten Ergebnisse.

6.1.3 Recurrent Neural Network(RNN)

Ein Recurrent Neural Network (RNN) ist eine besondere Machine-Learning-Methode. Es unterscheidet sich durch die sogenannte „rekurrente Kante“ von anderen Neuronalen Netzen [76, S. 532 ff.] und wird im Folgenden vorgestellt. Ein Neuron erhält in einem RNN im Zeitschritt t seinen Input sowohl von der Eingabeschicht, als auch von der eigenen verdeckten Schicht des vorhergegangenen Zeitschritts $t-1$ [76, S. 532 ff.]. Somit beeinflussen die Daten der Eingabeschicht aus dem Zeitschritt $t-1$ auch die Neuronenausgabe zum Zeitpunkt t [76, S. 532 ff.]. Auf diese Weise wird die zeitliche Abhängigkeit der Daten beim Trainieren des Modells berücksichtigt. Dafür sind drei zeitunabhängige Matrizen notwendig: Eine für die Verbindung zwischen der Eingabeschicht und den verdeckten Schichten, eine weitere für die rekurrente Kante und die dritte für die Verbindung zwischen der letzten verdeckten Schicht mit der Ausgabeschicht [76, S. 532 ff.]. Abbildung 6.2 zeigt den beispielhaften Aufbau eines RNNs. dabei sind U, V und W die Gewichtungen, H_t der Zustand und x_t beziehungsweise y_t Ein- und Ausgabe zum Zeitpunkt t.

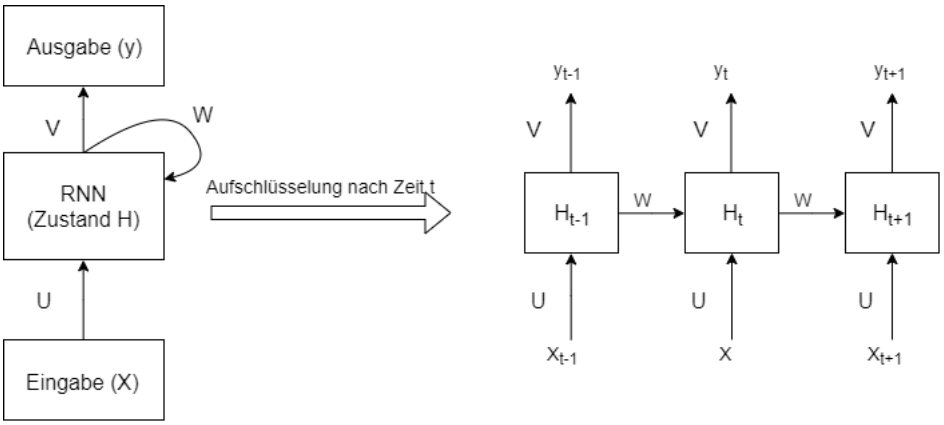

Abbildung 6.2: Aufbau eines Recurrent Neural Networks nach [50, S. 124]

Die Probleme bei Recurrent Neural Networks

Die Autoren von [4] beschreiben, dass Recurrent Neural Networks für Zeitreihen zwar bessere Ergebnisse liefern können als Neuronale Netze ohne rekurrente Kante. Bei langen Zeitspannen sei das Trainieren des Modells allerdings schwieriger. Dies wird durch verschwindende bzw. explodierende Gradienten begründet [73]. Diese Begriffe beschreiben, dass der Gradient der verdeckten Schicht entweder exponentiell wächst („explodiert“) oder gegen null geht („verschwindet“) [73]. Dies hat zur Folge, dass

das Recurrent Neural Network nicht mehr optimal trainiert werden kann. Bei einem verschwindenden Gradienten spiegeln die Gewichtungen den Einfluss von Zeitpunkten, die weiter in der Vergangenheit liegen, weniger genau wieder [4]. Während das Modell in diesem Fall die Beziehungen zwischen zeitlich weit auseinander liegenden Werten nicht lernen kann, tritt bei einem explodierenden Gradienten das Gegenteil ein [73].

Das Explodieren des Gradienten kann mittels Gradienten-Clipping vermieden werden [76, S.562 f.]. Dabei wird anstelle eines großen Schrittes, der sich eigentlich aus dem Gradienten ergeben würde, nur ein kleinerer Schritt in Richtung des Gradienten genommen [34, S. 285 f.]. Mit Long Short Term Memory(LSTM) Einheiten stellen die Autoren in [41] eine Lösung für das Problem des verschwindenden Gradienten vor.

6.1.4 Long Short Term Memory (LSTM)

Ein Long Short Term Memory (LSTM) ist eine RNN-Architektur, die entworfen wurde, um das Problem des verschwindenden bzw. explodierenden Gradienten bei RNNs zu lösen. Sie wurde zuerst in [41] vorgestellt und im Folgenden erläutert. Zur besseren Lesbarkeit wird der Begriff LSTM in dieser Arbeit nicht nur für eine einzelne Zelle verwendet, sondern auch für ein Neuronales Netz, dass LSTM-Schichten verwendet. Bei einem LSTM fließt der Fehler konstant über den Zellzustand in das Netz, statt wie bei dem in Abschnitt 5.4.4 beschriebenen BPTT-Algorithmus iterativ rückwärts durch das Netz zu fließen [41]. In einer Netzschicht können beliebig viele LSTM-Zellen anstelle der üblichen Neuronen eingesetzt werden. Um dieses Verhalten zu erreichen, wird das von den Autoren in [41] als „constant error carrousel“ (CEC) bezeichnete Prinzip eingesetzt, bei dem die rekurrente Kante eine festgesetzte Gewichtung von $w = 1$ hat und eine lineare Aktivierungsfunktion verwendet wird. Das CEC stellt einen zentralen Aspekt in der LSTM-Architektur dar, da es so weder zu einem verschwindenden Gradienten ($|w| < 1$) noch zu einem explodierenden Gradienten($|w| > 1$) kommen kann. Das CEC speichert den Zellzustand. Neben dem CEC besitzt eine LSTM-Speicherzelle zudem ein Eingabe-Gate, auch Input-Gate, und ein Ausgabe-Gate, auch Output-Gate genannt. Diese sind dazu da, Gewichtungskonflikte, die das Lernen erschweren, zu verhindern. Das Eingabe-Gate wird verwendet, um zu entscheiden, wann die gespeicherten Informationen, also der Zustand, überschrieben werden sollen und verhindert so den „input weight conflict“. Damit ist gemeint, dass eine Einheit den Input entweder speichern oder den vorherigen Zustand behalten möchte. Die Entscheidung hängt aber vom Input ab. Das Ausgabe-Gate hingegen entscheidet, wann der Zustand einer Zelle abgefragt wird und wann dies verhindert wird, um andere Zellen nicht zu beeinträchtigen. Ein Beispiel soll dies verdeutlichen. Sei die Einheit j über die Outputgewichtung w_{kj} mit der Einheit k verbunden. Dann entsteht der Konflikt, dass w_{kj} sowohl zum Abruf der Informationen in j verwendet wird, als auch um Störungen an k durch j zu verhindern. Dies bezeichnen die Autoren als „output weight conflict“ [41]. Da die Gates multiplikativ sind, gelten sie als geschlossen, wenn ihr Wert nahe bei null ist.

Nach der initialen Vorstellung der LSTM in [41] wurde die Architektur in [33] um das Lösch-Gate erweitert, das auch als Forget-Gate bezeichnet wird. Motivation dafür war, dass der gespeicherte Zustand einer Zelle unbegrenzt wachsen kann. Das führt dazu, dass die Funktion $h(x) = \frac{2}{1+e^{-x}} - 1$ mit Wertebereich $[-1, 1]$, die am Output-Gate auf den Zellzustand angewendet wird, gesättigt wird. In diesem Fall wäre die Zellausgabe gleich der Aktivierung am Output-Gate, wodurch die Zelle ihre Speicher-Fähigkeit verliert. Die Gewichtung der rekurrenten Kante ist in diesem aktualisierten LSTM-Entwurf nun nicht mehr fest 1, sondern wird als die Ausgabe des Lösch-Gates bestimmt. Allerdings werden die Gates so initialisiert, dass zu Beginn des Trainings die Ausgabe des Lösch-Gates und somit die Gewichtung der rekurrenten Kante nahezu 1 ist [33].

In [32] wird eine weitere Variante der LSTM-Einheiten vorgestellt, in denen diese um sogenannte „peephole connections“ erweitert werden. Da die Ausgabe der Zelle über das Ausgabe-Gate gesteuert wird, werden bei geschlossenem Ausgabe-Gate keine Informationen weitergegeben. Die gewichteten peephole connections verbinden die drei Gates mit dem Zellzustand. Bei der Berechnung der Aktivierung der Gates wird dann über die peephole connections das CEC, also der Zustand, mit einbezogen. Auf diese weise nimmt der Zustand Einfluss auf die Gate-Aktivierungen. Da so modifizierte LSTM-Einheiten die Länge von Zeitintervallen selbstständig lernen können, sehen die Autoren sie besonders für die Anwendungsbereiche Musik und Rhythmus geeignet [32].

In Keras sind LSTMs zwar laut [77] in der Variante nach [41] implementiert. Allerdings besitzen sie dennoch ein Lösch-Gate und [33] wird als Referenz aufgeführt [77]. Da die Implementation mit Keras erfolgt, wird in dieser Arbeit also eine LSTM-Variante verwendet, die der von [33] entspricht.

Neben den bereits aufgeführten Varianten gibt es noch weitere Ausprägungen von LSTM-Zellen. In [37] wurden acht Varianten verglichen. Die Autoren kamen zu dem Ergebnis, dass die entscheidendsten Faktoren für eine gute Performance das Vorhandensein eines Input-, Output- und Forget-Gates sowie die Verwendung einer Aktivierungsfunktion für den Input und Output sind. Andere Eigenschaften, die in manchen Varianten verwendet werden, wie beispielsweise die oben erwähnten peephole connections, wirken sich dagegen im Allgemeinen nicht signifikant auf die Ergebnisse aus [37]. Damit kann die in Keras implementierte Variante ohne Anpassungen verwendet werden.

Abbildung 6.3 zeigt eine solche Variante einer LSTM-Einheit ohne peephole connections.

Funktionsweise eines LSTM

Sei x_t der Input-Vektor zum Zeitpunkt t, N die Zahl der LSTM-Blocks und M die Anzahl der Eingaben. Dann lassen sich unter Anwendung der Notationen aus Abbildung 6.3 nach [37] folgende Gewichtungsmatrizen definieren:

- Die Input-Gewichtungen: W_z, W_i, W_f, $W_o \in \mathbb{R}^{N \times M}$

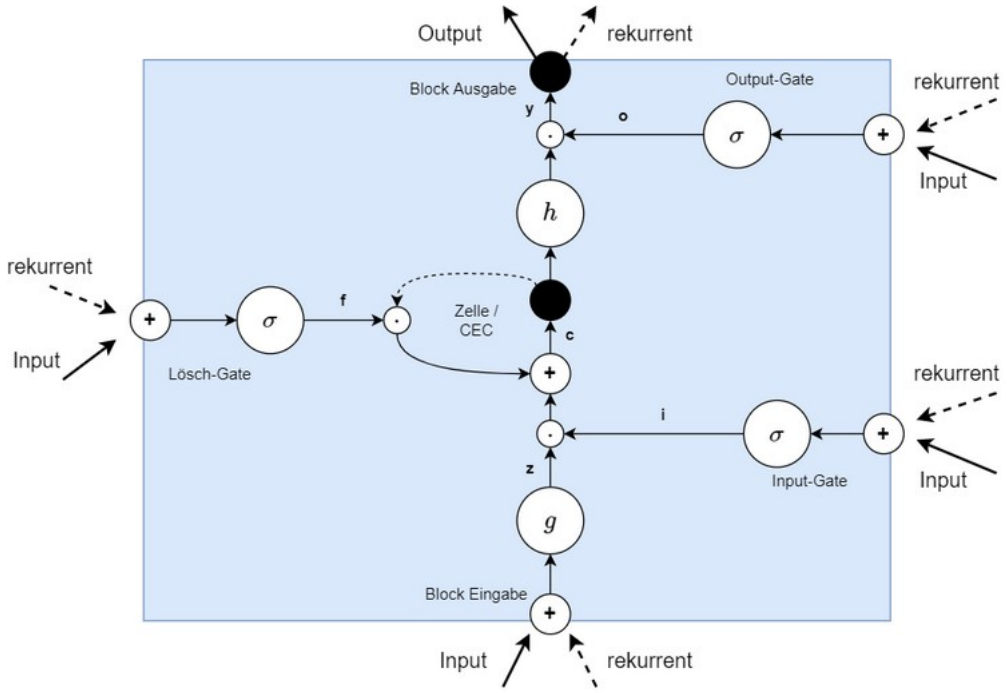

Abbildung 6.3: Aufbau eines LSTM-Blocks nach [37]. Dabei sind g und h die Aktivierungsfunktionen für den Input beziehungsweise Output (meistens *tanh*) und σ ist die Aktivierungsfunktion für die Gates (immer eine Sigmoid Funktion). Das $+$ bedeutet, dass die Summe über alle Eingaben gebildet wird und $\odot$ stellt die elementweise Multiplikation (Hadamardprodukt) dar. Die dünnen Verbindungen innerhalb des Blocks sind nicht gewichtet, die breiteren Verbindungen außerhalb besitzen dagegen eine Gewichtung. Gestrichelte Linien deuten eine Zeitverschiebung an.

- Die rekurrenten Gewichtungen: R_z, R_i, R_f, $R_o \in \mathbb{R}^{N \times M}$
- Die Bias Gewichtungen: b_z ,b_i, b_f, $b_o \in \mathbb{R}^N$

Der Index $[.]_i$ signalisiert die Zugehörigkeit zum Input-Gate, $[.]_f$ zum Forget-Gate, $[.]_o$ zum Output-Gate und $[.]_z$ zur Block Eingabe. Mit diesen kann der sogenannte „forward pass“ durchgeführt werden, mit dem die Werte für die Gates, den Zustand und die Ausgabe berechnet werden. Dafür werden nach [37], mit Ausnahme der peephole connections, die folgenden Berechnungen durchgeführt. Für die Block-Eingabe:

$$z_t = g\left(W_z x_t + R_z y_{t-1} + b_z\right). \tag{6.2}$$

Für das Input-Gate:

$$i_t = \sigma\left(W_i x_t + R_i y_{t-1} + b_i\right). \tag{6.3}$$

Für das Forget-Gate:

$$f_t = \sigma\left(W_f x_t + R_f y_{t-1} + b_f\right). \tag{6.4}$$

Für den Zustand:

$$c_t = z_t \odot i_t + c_{t-1} \odot f_t. \tag{6.5}$$

Für das Output-Gate:

$$o_t = \sigma\left(W_o x_t + R_o y_{t-1} + b_o\right). \tag{6.6}$$

Daraus ergibt sich die Block-Ausgabe als

$$y_t = h(c_t) \odot o_t. \tag{6.7}$$

Ein LSTM hat nach [41] folgende Vorteile:

1. Verhinderung des explodierenden bzw. verschwindenden Gradienten.
2. Durch die Gates und die Zustandsspeicherung ist es möglich, sowohl langfristige als auch kurzfristige Zusammenhänge zu erlernen.
3. Es sind anders als zum Beispiel bei sogenannten „hidden Markov models“ keine a priori Informationen notwendig.
4. Die Performance hängt nur geringfügig von Hyperparametern wie der Lernrate ab, weswegen diese nicht stark optimiert werden müssen.
5. Die Gewichtsaktualisierung hat eine Komplexität von $O(1)$.

Beispiele für die Anwendung von LSTM-Einheiten auf Zeitreihen

Da die LSTM-Architektur speziell für die Verarbeitung von Sequenzen geschaffen wurde, gibt es mehrere Beispiele für die Anwendung auf Zeitreihen. In [12] nutzten die Autoren ein tiefes LSTM-Netz zur Anomalieerkennung in EKG-Signalen. Einen weiteren Einsatz für medizinische Zwecke stellen die Autoren in [62] vor, bei dem sie aus Sensordaten Krankheitsdiagnosen als Klassifikation erzeugten. In [64] wurde ein tiefes LSTM-Netz zur Vorhersage und anschließenden Anomalieerkennung auf vier verschiedenen Zeitreihen erprobt. Die Autoren kamen zu der Schlussfolgerung, dass diese Architektur robuster sein könnte als ein normales RNN.

6.1.5 Gated Recurrent Unit (GRU)

Der Abschnitt 6.1.4 hat gezeigt, dass LSTM-Zellen komplex sind. Ein ähnlicher aber weniger komplexer Ansatz, der ebenfalls eine Spezialform eines RNNs darstellt, sind die sogenannten „Gated Recurrent Units“. Sie wurden in [14] vorgestellt und werden im Folgenden erläutert. Die Gated Recurrent Units haben zwar auch einen internen Zustand (*hidden state*), im Gegensatz zu einem LSTM aber nur zwei Gates: das Reset- und Update-Gate. Eine solche GRU wird in Abbildung 6.4 dargestellt.

Funktionsweise einer GRU

Der Zustand und somit die Aktivierung der j-ten GRU lässt sich über folgende Gleichungen berechnen. Dabei ist σ die logistische Sigmoid-Funktion, x der Eingabevektor, h_{t-1} der vorherige Zustand, $[.]_j$ das j-te Element eines Vektors, r_j das Reset-Gate und z_j das Update-Gate der j-ten Einheit. Seien W_r und U_r gelernte Gewichtungsmatrizen des Reset-Gates, dann ist

$$r_j = \sigma\left(\left[W_r x_r\right]_j + \left[U_r h_{\langle t-1\rangle}\right]_j\right) \tag{6.8}$$

[14]. Analog gilt für das Update-Gate

$$z_j = \sigma\left(\left[W_z x_z\right]_j + \left[U_z h_{\langle t-1\rangle}\right]_j\right) \tag{6.9}$$

[14].

Der neue Zustand $\tilde{h}_{\langle t\rangle}$, der sich durch den Input ergibt, lässt sich dann unter Berücksichtigung des Reset-Gates und Verwendung des Hadamard-Produkts $\odot$ sowie einer Aktivierungsfunktion ϕ berechnen als

$$\tilde{h}_{\langle t\rangle} = \phi\left(\left[Wx\right]_j + \left[U\left(r \odot h_{\langle t-1\rangle}\right)\right]_j\right) \tag{6.10}$$

[14].

Damit lässt sich die Aktivierung der Zelle $h_{\langle t\rangle}$ in folgender Weise berechnen:

$$h_{\langle t\rangle} = z_j h_{\langle t-1\rangle} + (1 - z_j)\tilde{h}_{\langle t\rangle} \tag{6.11}$$

[14].

Das Update-Gate z bestimmt, ob der bestehende beziehungsweise vorherige Zustand $h_{\langle t-1\rangle}$ durch den neuen Zustand $\tilde{h}_{\langle t\rangle}$ aktualisiert werden soll, was offensichtlich aus Gleichung (6.11) hervorgeht. Je weiter das Update-Gate geöffnet ist, desto stärker geht der vorherige Zustand in den neuen Zustand mit ein. Bei geschlossenem Reset-Gate r, also wenn es nahe bei null ist, wird $h_{\langle t-1\rangle}$ verworfen, da r elementweise multiplikativ einfließt, siehe Gleichung (6.10). Der aktuelle Input wirkt sich dann nicht auf den Zustand aus. Da jede GRU ein eigenes Update- und Reset-Gate hat, werden einige Zellen durch häufig aktive Reset-Gates verstärkt kurzfristige Abhängigkeiten lernen, während Zellen mit oftmals aktivem Update-Gate eher langfristige Zusammenhänge lernen [14].

In [100] stellten die Autoren einen Vergleich zwischen LSTM und GRU an und kamen zu dem Ergebnis, dass die LSTM-Architektur theoretisch strikt leistungsfähiger ist als die GRU-Architektur. Ein empirischer Vergleich von LSTMs und GRUs kam jedoch zu dem Ergebnis, dass die beiden Ansätze sehr ähnliche Resultate liefern. Daher konnten die Autoren keine der beiden Varianten als die Überlegende festhalten, sondern heben die Abhängigkeit vom Datensatz und der Problemstellung hervor [15].

Beispiele für die Anwendung von GRUs auf Zeitreihen

Neben [15] liegt auch in [31] ein Vergleich zwischen der Performance von LSTM und GRU für Zeitreihen vor. Die Autoren nutzten die Neuronalen Netze zur Vorhersage des Verkehrsflusses, wobei das GRU-Netz bessere Ergebnisse lieferte. Auch die Autoren von [59] erzielten bei ihrem Test zur Vorhersage der Stromlast mit einem GRU-Netz leicht bessere Ergebnisse als mit einem LSTM.

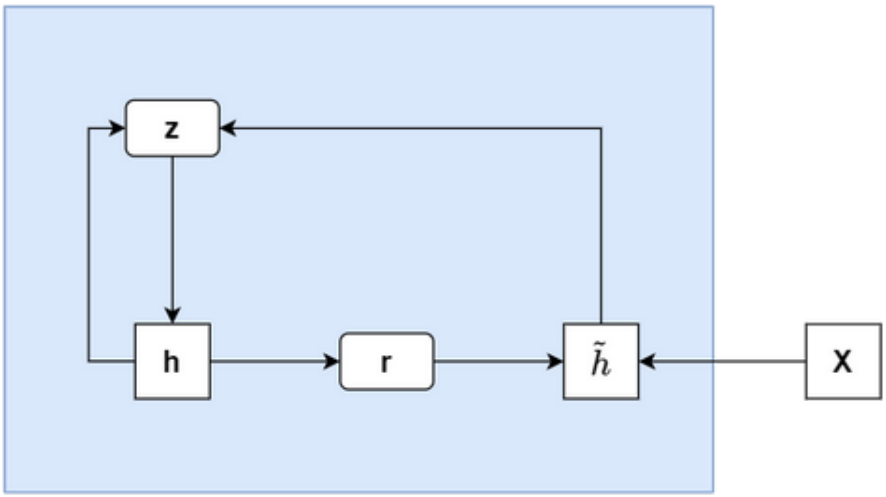

Abbildung 6.4: Eine Gated Recurrent Unit nach [14]. Dabei ist X der Input, h der aktuelle Zustand, $\tilde{h}$ der neue Zustand, z das Update-Gate und r das Reset-Gate.

6.2 Datenvorbereitung für multivariate KNNs

Der zuvor verarbeitete Datensatz wurde für die multivariaten Vorhersagen mit den KNNs erneut transformiert.

6.2.1 Featuregenerierung aus den Zeitstempeln

In der bisherigen Form des Datensatzes würde der Zeitstempel nur indirekt vom KNN für die Vorhersage berücksichtigt werden. Denn nur die Reihenfolge der Messwerte spiegelt deren zeitlichen Zusammenhang wieder. Daher wurden aus dem Zeitstempel Informationen extrahiert, die als Feature verwendet wurden. Aus dem Timestamp wurden für die einzelnen Messwerte die Stunde, die Minute und der Wochentag als Feature generiert. Bei einer Datengrundlage von mehreren Jahren könnte auch die Kalenderwoche oder der Monat relevant werden. Bei vier Wochen sind diese allerdings nicht wirklich aussagekräftig.

Der Datensatz enthielt nun etwa 190 Features. Aufgrund der Tatsache, dass die meisten dieser Merkmale den Systemzustand widerspiegeln, waren starke Korrelation zu vermuten. Daher wurde eine Hauptkomponentenanalyse (engl. „Principal components analysis“, PCA) angewendet, um eine Reduktion der Dimensionen zu erreichen.

6.2.2 Hauptkomponentenanalyse (PCA)

Nach [97] ist eine solche Dimensionsreduktion mittels PCA bei Zeitreihenvorhersagen üblicherweise möglich und führt zu besseren Vorhersageergebnissen.

Ein Beispiel für die Korrelation der Daten zeigt der Plot der Metrik „cpuusage_ps“ in Abbilung 4.4 auf Seite 26 mit dem der Metrik „dbCpuPs“ in Abbildung A.5 (siehe Anhang). Bei Betrachtung der Plots ließe sich vermuten, dass die beiden Metriken die gleichen Werte enthalten und sich lediglich um den Faktor 100 unterscheiden, weil

die Metrik cpuusage_ps in der Einheit Prozent vorliegt. Ein Vergleich der Werte hat dies allerdings widerlegt. Die Metriken ähnlich sich lediglich so stark, dass der Verlauf nahezu identisch aussieht.

Mittels PCA kann ein Vektor $x \in \mathbb{R}^l$ der Länge l verlustbehaftet in einen Vektor $c = f(x)$, $c \in \mathbb{R}^n$ der Länge n kodiert werden, wobei $n < l$ gilt [34, S. 51 ff.]. Um den Verlust möglichst gering zu halten, soll der Abstand zwischen x und dem aus c dekodierten Vektor $x' = g(c)$ minimiert werden [34, S. 51 ff.]. Der Abstand kann mit der L2-Norm als euklidischer Abstand gemessen werden [34, S. 51 ff.]. Wenn die Dekodierung durch Multiplikation von x mit der Matrix $D \in \mathbb{R}^{n \times l}$ als

$$g(c) = Dc \tag{6.12}$$

erfolgt, wird die Kodierung durch

$$f(x) = D^T c \tag{6.13}$$

erreicht, wobei für D die Eigenvektoren mit den größten Eigenwerten verwendet werden [34, S. 51 ff.]. Die PCA wurde mit der Implementierung der Bibliothek „sklearn“ umgesetzt und so eingestellt, dass mindestens 95% der Varianz erhalten bleiben. Damit konnte die Anzahl der Features um zwei Drittel auf etwa 60 reduziert werden.

6.2.3 Ablauf der Datenvorbereitung

Vor dem Anwenden der PCA auf den vorliegenden Datensatz wurden die Daten standardisiert um die Anforderung, dass der Erwartungswert null beträgt [34, S. 164], zu erfüllen. Außerdem wurde die Zielvariable zuvor entfernt und anschließend als letzte Spalte angefügt, um die Eingabesequenzierung, die in Abbildung 6.6 auf Seite 58 dargestellt wird, vorzubereiten. Dieser Transformationsablauf wird in Abbildung 6.5 dargestellt.

Für die Implementierung der Standardisierung wurde ebenfalls die Bibliothek *sklearn* verwendet. Tabelle 6.1 gibt einen Überblick über die Auswirkung der PCA. Die Effekte auf das Training waren allerdings etwas überraschend. Nur bei dem MLP konnten deutliche Zeitvorteile festgestellt werden. Die Laufzeit einer Epoche verringerte sich von geschätzt mehreren Stunden auf knapp nicht einmal zwei Minuten. Der Versuch, das MLP ohne PCA zu trainieren führte zu einem Systemabsturz. Die Laufzeitangabe basiert auf der Schätzung, die Keras angibt. Bei den KNNs mit LSTM- und GRU-Zellen und dem CNN ließ sich die Laufzeit durch die Hauptkomponentenanalyse allerdings nicht verbessern. Auch wenn die absoluten Zeiten hardwareabhängig sind, lässt sich aufgrund der gleichen Testbedingungen von einer Vergleichbarkeit ausgehen. Beim GRU-Netz ist der Validierungsfehler mit PCA besser als ohne. Beim LSTM-Netz und dem CNN führte die PCA allerdings zu Verschlechterungen beim Validierungsfehler. Besonders beim LSTM-Netz ist ein relevanter Unterschied festzustellen. Die fürs Training verwendete Hardware besteht aus einem Intel Core i5-8250U und acht Gigabyte Arbeitsspeicher. Es wurde keine Grafikkarte verwendet.

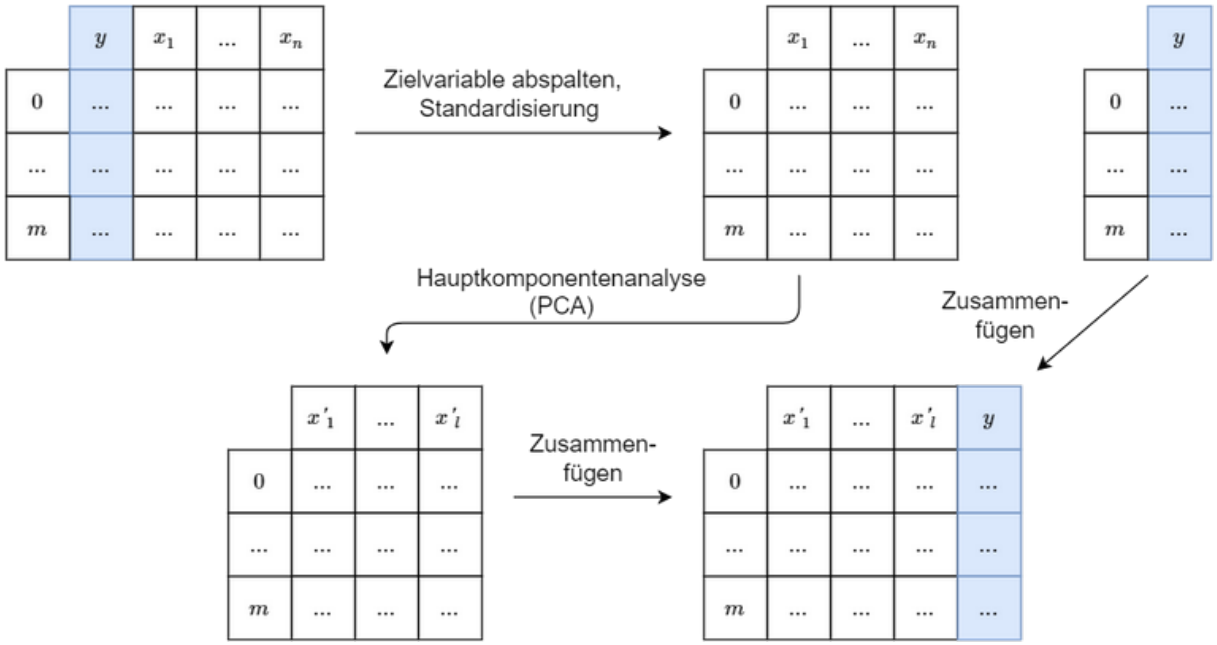

Abbildung 6.5: Vorbereitung des Datensatzes für eine multivariate Vorhersage mit KNNs. Die Featuregenerierung aus den Timestamps ist nicht enthalten. Es wird eine Standardisierung und Hauptkomponentenanalyse durchgeführt.

Typ	Validierungsfehler mit PCA	Validierungsfehler ohne PCA	Zeit pro Epoche mit PCA	Zeit pro Epoche ohne PCA
MLP	0.673	– (*)	107s	5h38min (*)
CNN	0.048	0.045	10s	7s
LSTM	0.160	0.104	22s	21s
GRU	0.063	0.084	24s	24s

Tabelle 6.1: Auswirkung der Hauptkomponentenanalyse auf die Trainingszeit pro Epoche und den Validierungsfehler (MSE). (*): Das MLP konnte nicht ohne PCA trainiert werden, da dies zum Systemabsturz führte. Die Laufzeit ist eine Schätzung.

6.3 Implementierung der KNNs

Im Folgenden wird die Implementierung der Neuronalen Netze beschrieben. Die Bibliothek Keras ermöglicht durch die vorhandenen Klassen, ein Modell mit nur wenigen Zeilen Code zu erzeugen, zu trainieren und schließlich Vorhersagen zu erstellen. Die Implementierungen der verschiedenen KNNs weisen einige Gemeinsamkeiten auf. Zunächst wird ein Modell als Objekt `model` der Klasse `Sequential` erzeugt. Dann werden mit der Funktion `model.add()` dem Modell Schichten hinzugefügt. Dabei kann die Art der Schicht in Form eines Objekts als Parameter angegeben werden. Diese haben wiederum Parameter für beispielsweise die Neuronenanzahl, die Aktivierungsfunktion oder Schichtenspezifische Eigenschaften. Bei der Eingabeschicht wird die Dimension der Eingaben angegeben. Wird für eine Schicht keine Aktivierungsfunktion angegeben, wird die Identitäsfunktion $f(x) = x$ verwendet. Wenn alle Schichten hinzugefügt

wurden, wird über `model.compile()` noch der Optimisierer und die Kostenfunktion angegeben. In dieser Arbeit wurden der Adam-Optimisierer (siehe Abschnitt 6.3.1) und der Mean Squared Error (siehe Abschnitt 6.6) als Kostenfunktion verwendet. Nach dem `model.compile()` Aufruf kann das Modell über `model.fit()` trainiert werden. Dafür werden als Parameter die Trainingseingaben und die zugehörigen erwarteten Trainingsausgaben angegeben. Außerdem werden die Validierungsdaten, die Epochenanzahl und die Callbacks, das sind die Objekte für das Early Stopping, angegeben. Unter anderem kann auch die Batch-Größe angegeben werden, die standardmäßig 32 beträgt. Die Batch-Größe wurde in dieser Arbeit bei dem Standardwert belassen. Ist das Modell fertig trainiert, kann mittels `model.predict()` aus einer Eingabe eine Vorhersage erzeugt werden. Die Ausgabeschicht wurde bei den KNNs als Keras `keras.layers.Dense`-Layer so instanziiert, dass die Anzahl der Neuronen gleich der Anzahl der Zeitschritte im Vorhersagehorizont ist. Die Ausgabe enthält also genau so viele Zeitschritte, wie nötig sind, um den Vorhersagehorizont zu erreichen. Sollte beispielsweise nur eine Stunde vorhergesagt werden und ein Zeitschritt wie hier 15 Minuten entsprechen, würden vier Neuronen verwendet werden.

Um die Reproduktionsfähigkeit zwischen mehreren Trainingsläufen zu verbessern ist es ratsam, für die Zufallszahlengeneratoren von Numpy und Tensorflow einen Seed zu verwenden [69, S. 14]. Dies kann über `numpy.random.seed()` und `tensorflow.random.set_seed()` erreicht werden.

6.3.1 Grundlegende Designentscheidungen

Im Folgenden werden einige grundlegende Entscheidungen dieser Arbeit erläutert, die auf alle KNNs angewendet werden.

Optimierungsalgorithmus

Die Wahl des Optimierungsalgorithmus ist eine wichtige Entscheidung, da die Lernrate die Performance des KNN stark beeinflusst. Allerdings lässt sich diese nicht universell gültig treffen [34, S. 347]. [83] kam bei dem Vergleich von Optimierungsalgorithmen, die auf dem SGD beruhen, zu dem Ergebnis, dass solche mit adaptiver Lernrate robuster sind. Das bedeutet, dass bei ihnen die Optimierung der Hyperparameter keine so große Bedeutung hat [83, S. 7]. Ein Beispiel für einen solchen Algorithmus ist *Adam*. Adam ist ein effizienter Algorithmus des stochastischen Gradientenabstiegverfahrens mit adaptiver Lernrate, der die Vorteile der Algorithmen *RMSProp* und *AdaGrad*, die ebenfalls auf dem SGD basieren, verbindet [54]. Sowohl Adam als auch RMSProp und AdaGrad sind ebenfalls in Keras implementiert und können als Optimierungsalgorithmus ausgewählt werden. Laut [69, S. 37] ist Adam der beliebteste und am weitesten verbreitete Optimisierer und kann in den meisten Fällen mit den in Keras hinterlegten Standardparametern verwendet werden, ohne Alternativen evaluieren zu müssen. Auch [34] bezeichnet ihn als „robust gegenüber der Auswahl der Lernparameter“ [34, S. 347]. Aus diesem Grund wird für alle KNNs der Adam-Algorithmus verwendet.

Single-Step-Ahead und Mutli-Step-Ahead Vorhersagen

Bei der Vorhersage muss zwischen der sogenannten Single-Step-Ahead- und der Multi-Step-Ahead-Methode unterschieden werden. Bei Single-Step-Ahead (SSA) wird immer nur ein einzelner zukünftiger Wert vorhergesagt. Wird hingegen ein Multi-Step-Ahead (MSA)-Ansatz gewählt, werden mit jeder Vorhersage mehrere Zeitschritte prognostiziert. Da nach der Anpassung der Zeitstempel aus Abschnitt 4.7.1 ein Zeitschritt 15 Minuten entspricht, würde eine SSA-Vorhersage lediglich 15 Minuten nach dem letzten Messwert liegen. Dieses Zeitfenster ist jedoch offensichtlich zu kurz, um eine sinnvolle Unterstützung bei der Erreichung der in Kapitel 1.2 definierten Ziele darzustellen. Natürlich wäre es möglich, erneut eine SSA-Vorhersage zu treffen, sobald ein neuer Messwert vorhanden ist, also eine neue Eingabe für die Vorhersage existiert. Wie viele Schritte in die Zukunft eine MSA Vorhersage geht, ist eine wichtige Entscheidung für die Erstellung des Modells.

Vorhersagehorizont

Aus Sicht des Anwenders ist natürlich ein möglichst großer Vorhersagehorizont wünschenswert. Allerdings nur, wenn die Vorhersage dabei genau genug bleibt. Die vorhandenen Daten stellen jedoch einen limitierenden Faktor dar. Für diese Arbeit gab es einen vierwöchigen Datensatz. Daher konnten allein aus dem Grund, dass ja Testdaten zurückgehalten werden müssen um das Modell zu beurteilen, maximal knapp vier Wochen vorhergesagt werden. Im Extremfall würde man aus einem Messwert den Rest der Daten, also knapp vier Wochen, vorhersagen. Es wird klar, dass hier eine gute Balance getroffen werden muss. Je größer der Vorhersagehorizont, desto kleiner wird die Trainingsdatenmenge und desto ungenauer, oder zumindest unfundierter, werden die Vorhersagen.

Außerdem muss berücksichtigt werden, dass der Vorhersagehorizont auch beeinflusst, wie viele Zeitschritte als Eingabe verwendet werden sollen. Auch hier soll dies mit einem Extrembeispiel verdeutlicht werden: Möchte man einen Tag vorhersagen, könnte als Eingabe auch nur ein einziger Zeitschritt verwendet werden. Hier wäre die Vorhersage selbstverständlich sehr schlecht. Wie viele Zeitschritte für einen gewissen Vohersagehorizont nötig sind, hängt natürlich stark vom verwendeten Datensatz ab. Je unregelmäßiger der Datensatz ist, desto länger sollte die Eingabe gewählt werden. Die Größe des Inputs wirkt sich jedoch auch stark auf das Training aus. Längere Eingabesequenzen führen zu längeren Epochenzeiten und reduzieren die Anzahl der Trainingseingaben.

Daher wurde mit dem Datenbankadministrator Rücksprache gehalten, wie weit die Vorhersagen in die Zukunft gehen müssten, damit diese einen Mehrwert bieten. Er unterteilte den Nutzen der Vorhersage dabei in zwei Bereiche. Ressourcenprobleme können einerseits dergestalt sein, dass sie durch eine Änderung der Konfiguration vermieden werden können und andererseits so, dass sie nur durch Beschaffung neuer Kapazitäten verhindert werden können. Da diese Neuanschaffung und anschließende Inbetriebnahme einige Wochen dauern kann, wären im letzteren Fall nur Vorhersagen über mehrere Wochen oder gar Monate hilfreich. Dies lässt sich aus einem vierwöchigen

Datensatz natürlich prinzipiell nicht mit relevanter Vorhersagegenauigkeit umsetzen. Zudem wie oben beschrieben bereits das Abspalten der Testdaten problematisch wäre. Eine Änderung der Konfiguration, um zu hohe Auslastungen zu verhindern, könnte beispielsweise das Verschieben eines geplanten Dienste auf eine andere Uhrzeit sein. Dafür wäre aus der Sicht des Datenbankadministrators bereits ein Vorhersagehorizont von einigen Stunden nützlich und ab drei Tagen ideal. Daher wurde die Entscheidung getroffen, als Vorhersagehorizont vier Tage festzusetzen. Das bedeutet, dass das Modell mit einer Vorhersage die nächsten vier Tage vorhersagt. Auch beim Training und dem Messen der Modellgenauigkeit wird dieser Vorhersagehorizont verwendet. Die besten Ergebnisse ergaben sich für die Eingabelänge von einem Tag. Die besten Prognosen für die nächsten vier Tage wurden also Betrachtung der letzten 24 Stunden erzeugt. Dies wirkt zunächst kontraintuitiv. Ein Möglicher Grund dafür wäre, dass hauptsächlich kurzfristige Abhängigkeiten vorliegen und ältere Messungen zu Fehlschlüssen führen. Außerdem ließ sich an der Komponentenzerlegung der Zeitreihe erkennen, dass eine tägliche Saisonalität vorliegt (siehe Abbildung 2.3).

Um die Daten für die MSA Vorhersage zu sequenzieren wurde das Sliding Window Verfahren verwendet.

Sliding Window

Für die Sequenzierung wird der Sliding Window Ansatz verwendet. Dafür wird über den Datensatz ein „Fenster“, das sogenannte Sliding Window, geschoben, das aus dem Datensatz die Sequenzen zur Eingabe in das Künstliche Neuronale Netz erzeugt. Abbildung 6.6 zeigt dieses Verfahren beispielhaft. In der Abbildung werden aus dem Datensatz mit jeder Verschiebung des Sliding Windows je zwei Zeitschritte der Merkmale und drei Zeitschritte der Zielvariable extrahiert. Prinzipiell könnte damit auch eine SSA Vorhersage erzeugt werden, hier wird aber wie oben erklärt ein MSA Ansatz realisiert.

6.3.2 Implementierung des MLP

Da die Struktur eines MLPs weniger komplex ist als die der anderen KNN-Typen und keine besonderen Mechanismen besitzt, um Sequenzen zu verarbeiten, erschien es sinnvoll, diese Option zu evaluieren. So sollte festgestellt werden, wie ein einfaches, nicht spezialisiertes KNN bei einer Zeitreihenprognose performt. Für die Implementierung eines MLPs wird in Keras die Layer-Klasse `keras.layers.Dense` verwendet. Als Aktivierungsfunktion wurde der *tanh* genutzt. Der Codeausschnitt in Listing C.2 (siehe Anhang) zeigt die Implementierung, die in dieser Arbeit verwendet wurde.

6.3.3 Implementierung des CNN

Auch für die Implementierung eines CNNs sind in Keras Schnittstellen vorhanden. Sowohl für die Faltung als auch das Pooling sind Klassen implementiert. Dabei muss für die Verarbeitung der Zeitreihe jeweils die eindimensionalen „1d“-Implementierung

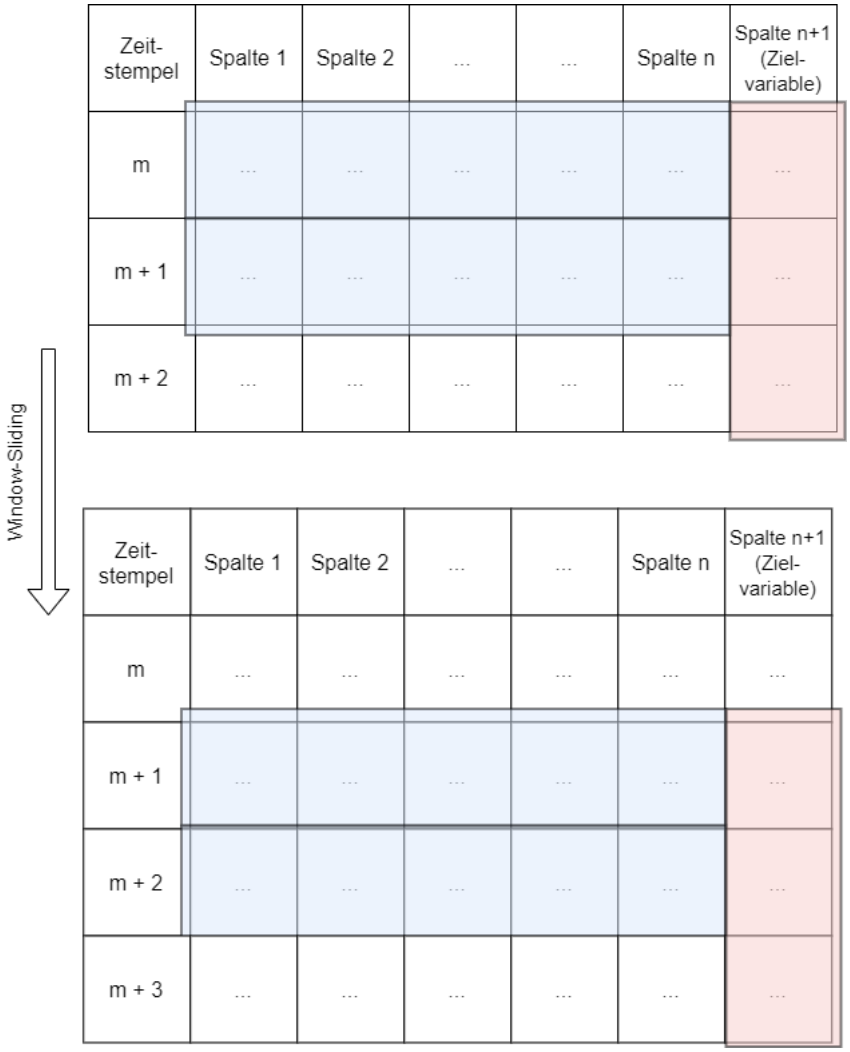

Abbildung 6.6: Das Sliding Window Prinzip. Es werden als Input zwei Zeitschritte der Merkmale gewählt und als Output drei Zeitschritte der Zielvariable. Dann wird das Fenster um einen Schritt verschoben und erneut eine Sequenz erzeugt. Das Neuronale Netz würde also aus den letzten zwei Zeitschritten der Merkmale (= 30 Minuten) die nächsten drei Zeitschritte (= 45 Minuten) der Zielvariable vorhersagen.

gewählt werden (siehe Abschnitt 6.1.2). Für die Faltung stehen unter anderem `keras.layers.Conv1D` und `keras.layers.ZeroPadding1D` zur Verfügung. Der stride kann bei `keras.layers.Conv1D` als Parameter übergeben werden. Für das Pooling sind beispielsweise das Max-Pooling (`keras.layers.MaxPooling1D`) und das Average-Pooling (`keras.layers.AveragePooling1D`) implementiert. In dieser Arbeit wurde erprobt, im CNN LSTM-Schichten zu verwenden. Da sich dies positiv auf die Vorhersage auswirkte, enthält die Implementierung des CNNs in Listing C.3 (siehe Anhang) auch eine LSTM-Schicht. Als Ausgabeschicht wird ein `keras.layers.Dense`-Layer verwendet, dessen Neuronenanzahl dem Vorhersagehorizont entspricht. Als Aktivierungsfunktion wurde sowohl für die Faltungsschicht als auch für das LSTM-Layer der *tanh* verwendet. Für die Faltung wurde der Parameter `padding = "same"` angegeben. Dieser sorgt dafür, dass ein zero padding eingesetzt wird, sodass die Dimension durch die Faltung nicht verändert

wird.

6.3.4 Implementierung des LSTM- und GRU-Netzes

Mit Keras lassen sich LSTM- und GRU-Netze in gleicher Weise implementieren. Diese wurden in dieser Arbeit in einer Architektur umgesetzt, die als „stacked“ bezeichnet wird [64]. Dabei werden mehrere rekurrente Schichten nacheinander als verborgene Schichten verwendet. In Keras muss für jede rekurrente Schicht, auf die eine weitere rekurrente Schicht folgt, der Parameter `return_sequences = True` verwendet werden. Nur dann wird die gesamte Ausgabesequenz und nicht nur das letzte Element dieser Sequenz an die nächste Schicht weitergereicht. Dies ist nötig, da als Eingabe für eine LSTM-Schicht alle Zeitschritte verwendet werden sollen. Als Ausgabeschicht wurde wie bei den anderen KNNs ein `keras.layers.Dense`-Layer verwendet. Für das LSTM wurde der *tanh* als Aktivierungsfunktion und für das GRU die ReLU-Funktion verwendet.

6.4 Modelloptimierung

Die Performance eines Modells, also wie gut die Vorhersagen sind, hängt neben den verwendeten Daten von den Hyperparametern ab [82, S. 282 ff.], mit denen es erstellt wurde. Das sind die Konfigurationsmöglichkeiten der KNNs, die unabhängig der Daten sind. Sie können als Stellschrauben bei der Optimierung eines Modells dienen, bei gleichbleibender Datenmenge. Einige solcher Hyperparameter sind beispielsweise die Anzahl der verborgenen Schichten, der Neuronen innerhalb einer Schicht und der Epochen. Aber auch die verwendete Aktivierungsfunktion, die Dropoutwahrscheinlichkeit und die Lernrate sind Hyperparameter.

Um die Effektivität eines Models bei unbekannten Daten zu bewerten, teilt man die vorhandenen Daten üblicherweise in einen Trainings- und einen Testdatensatz auf [82, S. 288 f.]. Das Modell wird mit dem Trainingsdatensatz trainiert, wohingegen ihm die Testdaten während dem Training vorenthalten werden. Um die Verallgemeinerungsfähigkeit des Modells zu beurteilen, werden mit dem Modell die Testdaten vorhergesagt. So kann ein Vergleich zwischen den Soll-Werten und der Prognose des Modells angestellt werden. Die Testdaten kennt das Modell davor nicht, um die Qualitätsabschätzung nicht zu verfälschen.

Da man für gewöhnlich daran interessiert ist, das bestmögliche Modell zu erreichen, ist es sinnvoll, die Hyperparameter zu optimieren. Würde man oben beschriebenes Prinzip mehrfach anwenden, um die Resultate verschiedener Hyperparameterkonfigurationen bei gleichen Trainings- und Testdaten zu vergleichen, würde man allerdings das Ergebnis verzerren. Die Testdaten würden auf diese Weise indirekt in die Erstellung des Modells einfließen, da basierend auf den Testdaten die Hyperparameter optimiert würden. Somit wären die Testdaten Teil der Trainingsdaten und die Verallgemeinerungsfähigkeit des Modells könnte nicht mehr eingeschätzt werden [82, S. 288 f.]. Die Verallgemeinerungsfähigkeit ist eine der wichtigsten Eigenschaften eines Modells. Das Modell soll nicht nur für die Trainingsdaten, sondern auch für unbekannte Daten gut

funktionieren, sich also gut „verallgemeinern lassen“. Um die Hyperparameter zu optimieren, ohne die Testdaten zu verwenden, kann die sogenannte Kreuzvalidierung (engl. „Crossvalidation“) angewendet werden.

6.4.1 Kreuzvalidierung

Bei der Kreuzvalidierung werden die Trainingsdaten zusätzlich in Trainings- und Validierungsdaten aufgeteilt [82, S. 289 f.]. Die Validierungsdaten werden genutzt, um verschiedene Hyperparametersets zu vergleichen. So kann die optimale Modellkonfiguration gefunden werden. Die Verallgemeinerungsfähigkeit dieses optimierten Modells kann anschließend mit den Testdaten evaluiert werden. Das Modell sieht die Testdaten also erst im allerletzten Schritt. So können nicht nur Modelle gleichen Typs, sondern auch unterschiedliche Modellansätze verglichen werden, beispielsweise ein MLP und ein LSTM-Netz. Die Aufteilung zwischen Trainings- und Validierungsdatensatz und anschließendem Training wird bei der Kreuzvalidierung iterativ wiederholt [82, S. 290]. Somit wird eine bessere Einschätzung des Modells erreicht, da für eine Hyperparameterkonfiguration die Modellperformance mit verschiedenen Validierungsdaten geprüft wird. Aus den Fehlern der einzelnen Iterationen wird der Mittelwert gebildet, um den durchschnittlichen Fehler des Modells zu erhalten [82, S. 290]. Dies ist notwendig, da die konkrete Aufteilung in Trainings- und Validierungsdatensatz, also welche Datenpunkte sich in den Validierungsdaten befinden, einen starken Einfluss auf den Fehler des Modells haben kann [76, S. 206]. Außerdem ist es auf diese Weise möglich, den gesamten Trainingsdatensatz zum Trainieren zu verwenden, da die Validierungsdaten für jede Iteration aus einem anderen Bereich der Daten genommen werden können [82, S. 288 ff.]. Die Testmenge bleibt hierbei aus dem oben angeführten Grund unberührt. Dieses Verfahren wird in Abbildung 6.7 auf Seite 60 dargestellt.

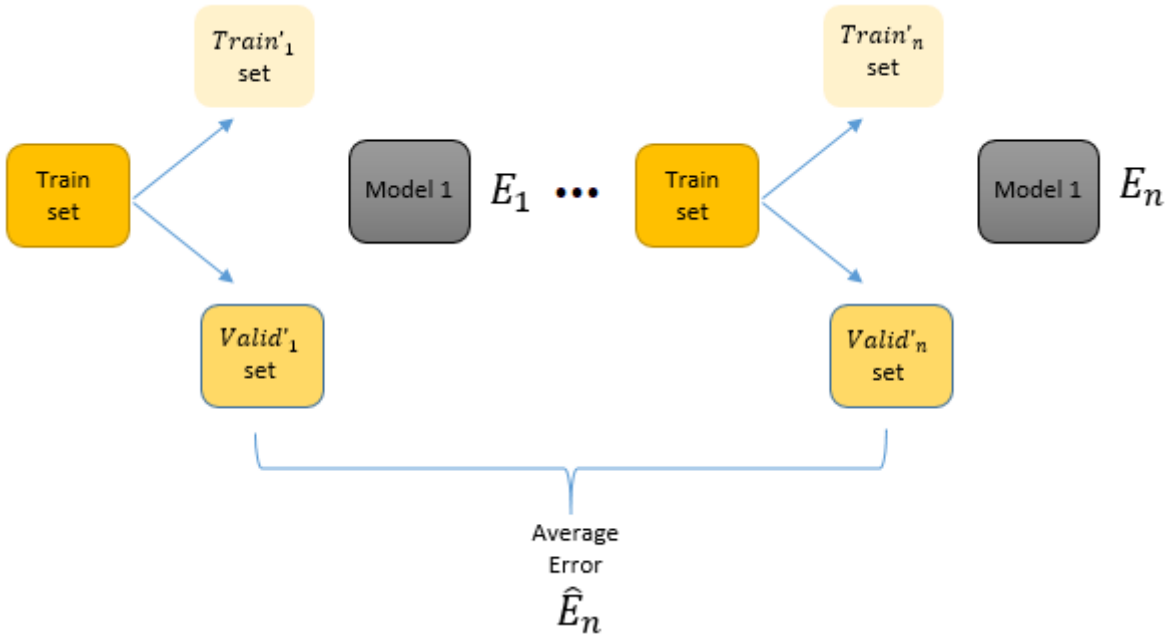

Abbildung 6.7: Das Prinzip der Kreuzvalidierung aus [82, S. 290].

Für die Aufteilung in Trainings- und Validierungsdaten gibt es mehrere Ansätze. Im Folgenden werden zwei davon kurz vorgestellt.

Leave One Out Kreuzvalidierung

Bei der „Leave One Out" Kreuzvalidierung wird aus dem für das Training vorhergesehenen Datensatz ein zufälliger Eintrag gewählt, der aus den n Trainingsdaten entfernt und als Validierungsdatenpunkt verwendet wird [82, S. 291]. Mit dem Rest wird das Modell dann trainiert. Dieser Vorgang wird n-mal durchgeführt, weswegen diese Methodik hauptsächlich für kleine Datensätze sinnvoll ist [82, S. 291].

k-fache Kreuzvalidierung

Im Folgenden wird die k-fache Kreuzvalidierung nach [76, S. 206 ff.] beschrieben. Um eine „k-fache Kreuzvalidierung" (engl. „k-fold Crossvalidation") durchzuführen, teilt man die Trainingsdaten zunächst in k Teilmengen auf, wobei jeder Datenpunkt nur in genau einer Teilmenge vorkommt. Die Teilmengen werden also aus dem Datensatz entfernt, ohne sie zu ersetzen. Davon wird nun eine Teilmenge als Validierungsdatensatz und $k-1$ Teilmengen für das Training genutzt. Dieser Mechanismus wird k-mal durchgeführt. Danach ergibt sich der Fehler des Modells als arithmetisches Mittel aus den k Fehlern. Welcher Wert für k gewählt werden sollte, ist abhängig von der Größe des vorliegenden Datensatzes. Bei vielen Daten sollte laut [76] ein kleineres k, zum Beispiel $k = 5$ gewählt werden. So kann die Rechenzeit verkürzt werden, ohne die Modelleinschätzung zu stark zu verfälschen. Ist der vorhandene Datensatz allerdings klein, sollte für k ein größerer Wert, beispielsweise $k = 10$ gewählt werden. So sind die k Datensatzfragmente kleiner, wodurch der Validierungsdatensatz, der ja eines der k Datenfragmente ist, einen geringeren Anteil der Trainingsdaten einnimmt und das Modell mit mehr Daten trainiert werden kann. Allerdings wird dadurch die Laufzeit verlängert und die Abschätzungen weisen aufgrund der größeren Ähnlichkeit der Trainingsdatensätze eine höhere Varianz auf [76, S. 206 ff.]. Für k empfiehlt [76, S. 207 f.] einen Standardwert von $k = 10$, da bei diesem eine gute Balance zwischen Bias und Varianz herrsche. Auch [82, S. 291] empfiehlt für k einen Wert mit $5 < k < 10$.

6.4.2 Rastersuche

Wie oben beschrieben, können mittels Kreuzvalidierung verschiedene Hyperparametersets verglichen werden. Dabei ist nicht festgelegt, wie die zu vergleichenden Konfigurationen zustande kommen. Dies kann entweder manuell oder automatisiert durchgeführt werden. Eine Möglichkeit zum automatisierten Vergleichen ist die sogenannte Rastersuche (engl. „Grid Search"), die im Folgenden nach [34, S. 481] vorgestellt wird. Für eine Rastersuche werden zunächst Werte definiert, die für die zu optimierenden Hyperparameter getestet werden sollen. Anschließend wird das Training für alle möglichen Kombinationen automatisiert durchgeführt. Anhand des Validierungsfehlers kann anschließend das beste Hyperparameterset ausgewählt werden. Da die anfangs definierten Hyperparameter nicht zwingend den bestmöglichen Wert enthalten, ist es ratsam, die Rastersuche mehrfach durchzuführen. So kann man sich an

die optimale Hyperparameterkonfiguration annähern. Da alle möglichen Kombinationen verwendet werden, beträgt die Komplexität für m Hyperparameter mit je n zu testenden Werten $O(m^n)$ [34, S. 481].

6.4.3 Ergebnisse der Hyperparameteroptimierung

Um die Hyperparameter für die einzelnen KNN-Typen automatisiert zu optimieren wurde in dieser Arbeit jeweils eine mehrfache Rastersuche in Kombination mit einer 5-fachen Kreuzvalidierung durchgeführt. Nicht alle Hyperparameter wurden automatisiert trainiert. Für beispielsweise die Lernrate wurde der Vergleich händisch durchgeführt. Die so optimierten Hyperparameter werden im Folgenden für die einzelnen KNNs aufgeführt.

In der Tabelle 6.2 werden einige optimierte Hyperparameter aufgeführt. Beim Test der Lernrate bestätigte sich die in 6.3.1 vorgestellt Ansicht, dass Adam mit den in Keras hinterlegten Standardparametern am besten funktioniert. Die Verwendung einer Dropout-Regulierung wurde ebenfalls erprobt, erwies sich jedoch nicht als förderlich. Für das CNN ist außerdem anzumerken, dass hier die Anzahl der Convolutional- und Pooling-Layer, die Anzahl der Filter beziehungsweise Kernel, die Kernel- und Pooling-Größe sowie die Anzahl der verwendeten LSTM-Layer und deren Zellenanzahl und Aktivierungsfunktion getestet wurden. Für diese Hyperparameter des CNN ergab die Grid Search:

- Anzahl Convolutional-Layer: 3
- Aktivierungsfunktion für Convolutional-Layer: *tanh*
- Anzahl Pooling-Layer: 0
- Amzahl Filter / Kernel: 150 für die erste, 80 für die zweite und 60 für die dritte Schicht
- Kernel-Größe: 2
- Pooling-Größe: n.a.
- Anzahl LSTM-Layer: 1
- Aktivierungsfunktion für LSTM: *tanh*
- Anzahl LSTM-Zellen: 75

Die Modelle wurden mit diesen Hyperparametern trainiert, um sie zu vergleichen und die Entscheidung für ein Modell zu treffen.

6.5 Trainings-, Validierungs- und Testdaten

Um die Modelle zu trainieren, musste der Datensatz in Trainingsdaten, Validierungsdaten und Testdaten aufgeteilt werden. Die Trainingsdaten werden zum Trainieren des Modells, also zum Lernen, verwendet. Mit den Validierungsdaten wird während des

Typ	Aktivierungs-funktion	Anzahl hidden layer	Anzahl Zellen / Neuronen	Lernrate
MLP	*tanh*	4	50	0.001
CNN	*tanh*	siehe Abs. 6.4.3	siehe Abs. 6.4.3	0.001
LSTM	*tanh*	4	75	0.001
GRU	ReLU	5	50	0.001

Tabelle 6.2: Die mit Grid Search optimierten Hyperparameter. Es fällt auf, dass in den meisten Fällen der *tanh* die beste Aktivierungsfunktion war.

Trainings bestimmt, wie gut das Modell ist. Dies ist für die Early Stopping Regularisierung notwendig (siehe Abschnitt 5.6.2). Die Testdaten werden erst verwendet, wenn die Entscheidung für ein Modell gefallen ist. Damit wird dann die Leistungsfähigkeit des Modells endgültig beurteilt. Das Aufteilen der Daten in Trainings-, Validierungs- und Testdaten erfolgte, nachdem sie mit dem Sliding Window Prinzip sequenziert wurden. Zunächst wurde der Datensatz in Trainings- und Testdaten aufgeteilt. Die Testdaten sind dann die letzte Sequenz innerhalb des sequenzierten Datensatzes. Ihre Länge entspricht somit genau den Eingabeschritten (1 Tag) für die Eingabe und dem Vorhersagehorizont (4 Tage) für die erwartete Ausgabe. Die Trainingsdaten sind die restlichen Sequenzen. Die Aufteilung der Trainingsdaten in Trainings- und Validierungsdaten wurde prozentual festgelegt. Die letzten 15% der Sequenzen wurden als Validierungsdaten verwendet. Es wurde auch erprobt, die Validierungsdaten stattdessen zufällig aus den Trainingsdaten zu ziehen. Dies wirkte sich allerdings negativ aus. Aus dem gesamten Datensatz werden also aus n Sequenzen $0.85n - 1$ Sequenzen für das Training verwendet.

6.6 Modellbewertung

Zur Beurteilung der Genauigkeit eines Modells gibt es verschiedene Metriken. Diese werden im folgenden zunächst erklärt. Anschließend werden die Validierungsfehler der Modelle anhand einer dieser Metriken verglichen, bevor schließlich für das finale Modell mehrere Fehlerkennzahlen berechnet werden.

6.6.1 Möglichkeiten zur Fehlerbestimmung

Mean Absolute Error

Der Mean Absolute Error (MAE) ist durch folgende Gleichung definiert:

$$e_{mae} = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{6.14}$$

[82, S. 170].

Da der absolute Betrag aller Fehler aufsummiert wird, wächst der MAE auch, falls also die Prognose $\hat{y}_i$ kleiner ist als der tatsächliche Wert y_i. Demnach verhält sich der MAE

umgekehrt proportional zur Güte des Modells. Je höher der MAE, desto größer ist der Fehler des Modells.

Mean Squared Error

Der Mean Squared Error (MSE) wird definiert als:

$$e_{mse} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{6.15}$$

[50, S. 170].

Durch die Quadrierung im Zähler führt eine geringere Anzahl betragsmäßig großer Fehler zu einem höheren MSE als eine größere Anzahl betragsmäßig kleinerer Fehler. Außerdem erhöhen dadurch sowohl negative als auch positive Abweichungen der Vorhersage $\hat{y}_i$ vom Originalwert y_i den MSE. Da im Zähler die Fehler vorzeichenneutral addiert werden, bedeutet ein geringerer MSE ein besseres Modell. Der Wertebereich des MSE ist $0 \leq e_{mse} \leq \infty$. Ein perfektes Modell würde bedeuten, dass die einzelnen Abweichungen $\hat{y}_i - y_i = 0$ sind für alle i. In diesem Fall wäre die Summe und somit auch der MSE $e_{mse} = 0$.

Root Mean Squared Error

Der Root Mean Squared Error (RMSE) ist die Quadratwurzel des MSE und wird definiert als:

$$e_{rmse} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{6.16}$$

[82, S. 282]. Beziehungsweise unter Verwendung des MSE wie in Gleichung (6.15) beschrieben:

$$e_{rmse} = \sqrt{e_{mse}} \tag{6.17}$$

Durch die Quadratwurzel, wird die Einheit des Fehlers wieder auf die Einheit der Eingabedaten zurückgeführt und liegt nicht mehr quadriert vor.

Bestimmtheitsmaß

Das sogenannte Bestimmtheitsmaß, das auch Determinationskoeffizient oder R^2 genannt wird, setzt die Streuung der Prognose ins Verhältnis zur Streuung der Originaldaten [29, S. 150 f.]. R^2 wird demnach über folgende Formel definiert:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{6.18}$$

Typ	Trainingsfehler (MSE)	Validierungsfehler (MSE)	Epoche
MLP	1.233	0.444	116
CNN	0.781	0.045	20
LSTM	0.807	0.104	14
GRU	0.711	0.062	215

Tabelle 6.3: Trainings- und Validierungsfehler sowie entsprechende Epoche der verschiedenen Modelle. Trainiert wurden die Modelle mit optimierten Hyperparametern. Die Modelle werden in ihrer jeweils besten Ausprägung verglichen. Die Epoche gibt an, in welcher Epoche die beste Ausprägung erreicht wurde. Es wurden vier Tage vorhergesagt, als Eingabe wurden 24 Stunden verwendet. Die Fehler sind auf drei Nachkommastellen gerundet.

[29, S. 150 f.]. Dabei ist $\hat{y}_i$ ein vorhergesagter Wert, y_i der dazugehörige Originalwert, und $\bar{y}$ das arithm. Mittel der Originaldaten. Diese Metrik findet bei Regressionen Anwendung. Der Wertebereich der Funktion ist nach [29, S. 150] $0 \leq R^2 \leq 1$. Es ist jedoch durchaus denkbar, dass der Bruch einen größeren Wert als 1 annimmt, wodurch $R^2 < 0$ wäre. Dies geschieht, wenn der Zähler, also die Fehler der Modellvorhersagen, größer sind als die Abweichung der tatsächlichen Werte vom Mittelwert. Ist $R^2 < 0$ sind die Vorhersagen des Modells also schlechter, als die eines Modell, das immer $\bar{y}$ vorhersagt. Für ein Modell dessen Vorhersage immer $\bar{y}$ ist, wäre $\hat{y}_i = \bar{y}$ und somit im Bruch $\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$ Zähler und Nenner gleich. Für ein solches Modell wäre also $R^2 = 0$. Je näher das Ergebnis bei 1 ist, desto besser erklärt das Modell die Streuung und desto besser ist somit das Modell [29, S. 150 f.]. Bei einem Modell, das perfekte Vorhersagen liefert, wenn also gilt $y_i = \hat{y}_i$ für alle i, ist der Zähler des Bruchs $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0$. Somit wäre dann das Bestimmtheitsmaß $R^2 = 1 - 0 = 1$.

6.6.2 Fehlerauswertung der erstellten Modelle

In Tabelle 6.3 werden die Validierungsfehler der erzeugten Modelle dargestellt. Dabei wird entsprechend der Rastersuche und des in Abschnitt 5.6.2 erläuterten Early Stoppings das jeweils beste Modell verglichen. Die Modelle wurden für maximal 400 Epochen trainiert, wobei nach 50 Epochen ohne Verbesserung des Validierungsfehlers das Early Stopping aktiviert wurde. Außerdem wurde die PCA nur bei den Modellen angewendet, bei denen sie zu einer Verbesserung des Validierungsfehlers geführt hat (siehe Abschnitt 6.2). Interessant ist, dass der Validierungsfehler bei allen Modellen geringer ist als der Trainingsfehler. Abbildung 6.8 zeigt den Verlauf des Trainings- und Validierungsfehlers während dem Training des CNN. Der Validierungsfehler war während des gesamten Trainings geringer als der Trainingsfehler. Wie auch in anderen Vergleichen (siehe Abschnitt 6.1.5) führte das GRU-Netz in diesem Fall zu einem besseren Ergebnis als das LSTM-Netz.

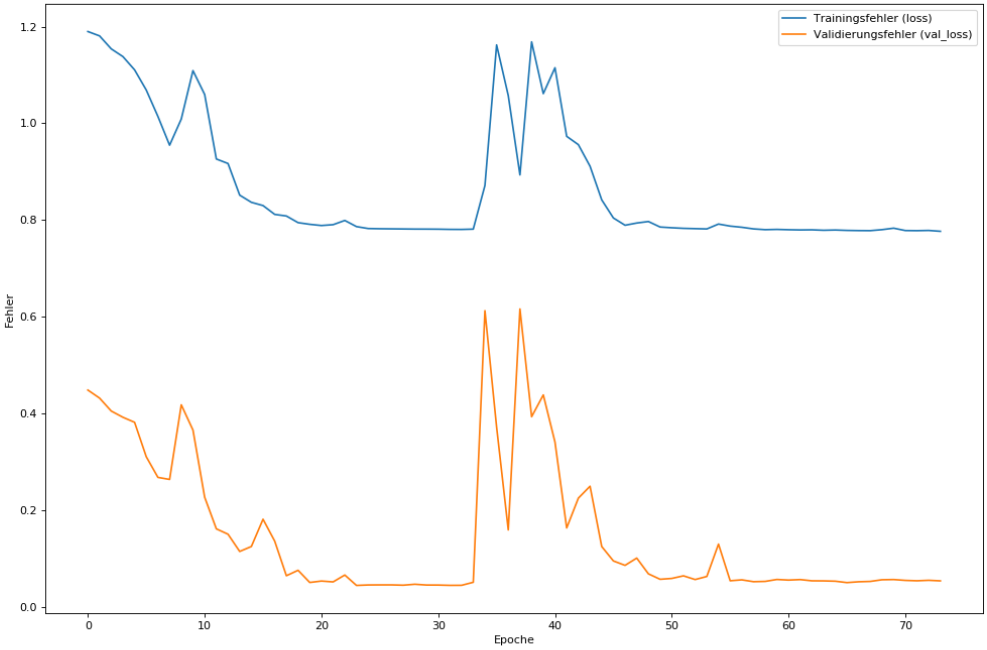

Abbildung 6.8: Fehlerverlauf beim Training des CNN. Es ist erkennbar, dass der Validierungsfehler zu jeder Zeit geringer war als der Trainingsfehler. Ein langsam steigender Validierungsfehler, was für ein Overfitting sprechen würde (siehe Abschnitt 5.5) ist nicht erkennbar. Die Early Stopping Regularisierung war also effektiv.

6.6.3 Interpretation der Fehlerauswertung

Die Auswertung der Fehler und damit der Leistungsfähigkeit der Modelle zeigt eindeutig, dass das MLP für diesen Anwendungsfall nicht geeignet ist. Dessen Validierungsfehler ist um Faktor 10 schlechter als der des besten Modells. Die anderen drei Modelle erzeugen alle bessere Prognosen. Das CNN hat den geringsten Fehler. Dies könnte damit zusammenhängen, dass das CNN hauptsächlich kurze Abhängigkeiten berücksichtigt (siehe Abschnitt 6.1.2). Ein Erklärungsansatz dafür, dass die Validierungsfehler bei allen Modellen geringer sind als die Trainingsfehler ist, dass der herausstechende Ausreißer mit knapp 70% im Trainingsdatensatz vorkommt (siehe Abbildung 4.4). Im Validierungsdatensatz sind hingegen keine besonders hohen Werte enthalten. Die Validierungsdaten sind regulärer, beziehungsweise „einfacher“ als die Trainingsdaten. Dass das GRU-Netz einen geringeren Validierungs- sowie Testfehler erzeugt als das LSTM-Netz bekräftigt die Ergebnisse aus [15]. Insgesamt lässt sich sagen, dass für eine bessere Vorhersage vermutlich historische Daten über einen längeren Zeitraum nötig wären.

Typ	MSE	RMSE	R^2
CNN	0.756	0.869	0.860

Tabelle 6.4: Testfehler des ausgewählten CNN Modells. Die Tabelle enthält die Werte für den Mean Squared Error (MSE), den Root Mean Squared Error (RMSE) und das Bestimmtheitsmaß (R^2). Die Werte sind auf drei Nachkommastellen gerundet.

6.6.4 Entscheidung für ein Modell

Die Entscheidung fiel auf das CNN, da hier der Fehler und die Epochendauer am geringsten ist. Jedoch erscheint es sinnvoll, bei längerem Einsatz des Systems und somit größerer Datenmenge, den Vergleich zwischen LSTM, GRU und CNN erneut durchzuführen. Es wäre denkbar, dass es bei der Metrik langfristige Abhängigkeiten gibt, die in den vorliegenden vier Wochen nicht vorkommen. Dann könnten die rekurrenten Netze zu besseren Ergebnissen führen. Die Fehlerauswertung des CNNs für die Testdaten findet sich in Tabelle 6.4. Abbildung 6.9 zeigt den Vergleich zwischen der Vorhersage sowie den Testdaten.

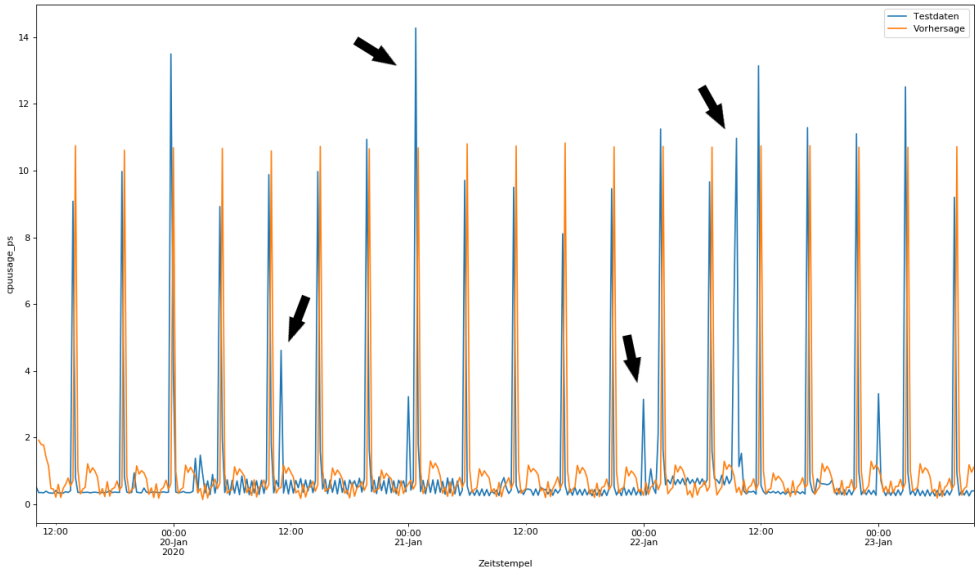

Abbildung 6.9: Plot der Vorhersage und der Testdaten. Man erkennt, dass das Modell ein Muster gelernt hat, einige Spitzen allerdings nicht vorhersagt. Es wäre möglich, dass diese Spitzen schlicht nicht vorhersagbar sind, da sie aus unerklärlichen, einmaligen Ereignissen entstanden sind. Genauso wäre es aber denkbar, dass sie sich mit einem größeren Datensatz lernen ließen.

Das CNN wurde vor der Integration in die Webanwendung noch einmal trainiert, allerdings ohne Validierungsdaten oder Testdaten abzuspalten. Stattdessen wurde der gesamte Datensatz als Trainingsdaten verwendet. Somit ist die Trainingsdatenmenge größer und das entstandene Modell präziser. Hierbei wurden dann nicht mehr die Early Stopping Regularisierung verwendet, sondern bis zu der Epoche trainiert, die

sich davor durch das Early Stopping als optimal ergeben hat. Das CNN wurde also mit allen Daten für 20 Epochen trainiert. Das so entstandene CNN-Modell wird für die Integration verwendet.

6.6.5 Modell für zweite Serverinstanz

Das Modell wurde nur für eine Serverinstanz erstellt. Da die anderen Serverinstanzen andere historische Daten haben, die sich stark unterscheiden können, kann das trainierte Modell nicht für andere Serverinstanzen verwendet werden. Für die Integration in die Webanwendung sollte in dieser Arbeit noch ein zweites Modell für eine andere Serverinstanz erstellt werden (siehe Abschnitt 8.5.2). Dafür müsste der gesamte Prozess beginnend bei dem Zusammenführen der einzelnen Wochen zu einem Datensatz bis hin zum Speichern dieses zweiten Modells auf dem Server erneut durchgeführt werden. Hierbei geht es lediglich um den optischen Effekt in der Webanwendung und nicht darum, ein optimiertes Modell zu erstellen. Daher wurde, ohne einen Vergleich durchzuführen, ein CNN mit den gleichen Hyperparametern verwendet, die schon für die erste Serverinstanz zum Einsatz kamen.

Abbildung 6.10 zeigt den Verlauf der Metrik cpuusage_ps für die zweite Serverinstanz. Es sind ebenfalls zwei deutliche Lastspitzen zu erkennen. Während bei dem ersten Diagramm (siehe Abbildung 4.4) allerdings noch regelmäßig Auslastungen von knapp 10% auftraten, ist dies bei diesem Server nicht festzustellen. Auch für diese Zeitreihe wurden Kennzahlen berechnet. Die Auswertung befindet sich in Tabelle 6.5. Diese Kennzahlen sind ähnlich zur ersten Zeitreihe (siehe Tabelle 4.2), allerdings beträgt das arithmetische Mittel nur die Hälfte.

In Tabelle 6.6 werden der Trainings- und Validierungsfehler aufgeführt. Der Verlauf dieser Fehler, der in Abbildung 6.12 dargestellt wird, zeigt allerdings, dass sich diese Fehler während des Trainings nicht signifikant verbesserten. Auch die Vorhersage ist nicht zufriedenstellend. Der Testfehler wird in Tabelle 6.7 festgehalten und Abbildung 6.11 zeigt den grafischen Vergleich zwischen Vorhersage und Testdaten. Zusammenfassend lässt sich sagen, dass dieses Modell keine zufriedenstellende Prognose liefert. Der R^2 Wert, der sogar leicht negativ ist, macht dies deutlich. Das bekräftigt die Vermutung, dass für jeden Server, für den eine Vorhersage erstellt werden soll, der gesamte Prozess inklusive Modelloptimierung durchgeführt werden muss. Nichtsdestotrotz wurde auch dieses Modell nochmal mit allen Daten trainiert und dann in die Webanwendung integriert.

Nachdem die Modelle nun fertig trainiert sind, wird im Folgenden die Integration in die Webanwendung behandelt.

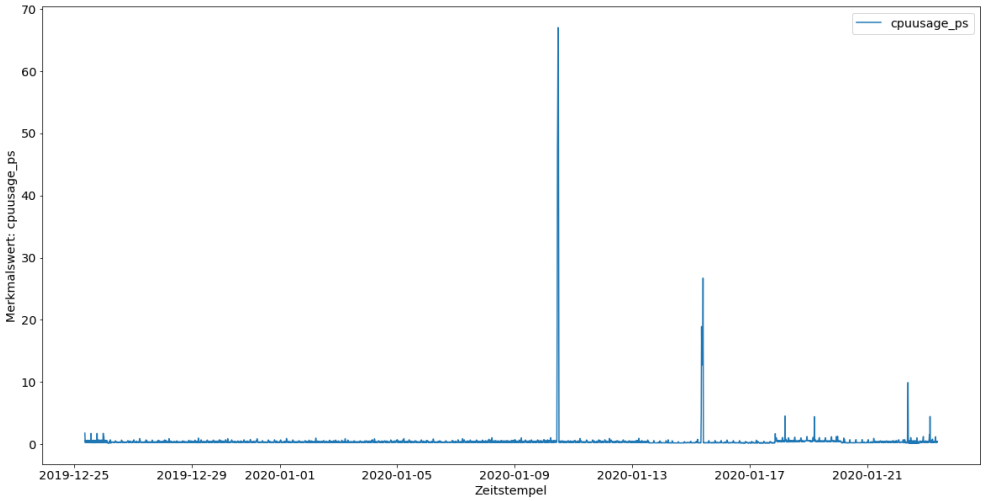

Abbildung 6.10: Plot der CPU-Auslastung für eine zweite Serverinstanz. Verwendete Metrik: cpuusage_ps.

Kennzahl	Wert
Arithm. Mittel	0.53
Standardabweichung	2.47
Minimum	0.12
Maximum	67.03
25%-Quantil	0.27
50%-Quantil / Median	0.41
75%-Quantil	0.45

Tabelle 6.5: Kennzahlen zur Zeitreihe cpuusage_ps für die zweite Serverinstanz. Die Werte wurden auf zwei Nachkommastellen gerundet. Es ergibt sich ein ähnliches Bild wie in für die erste Serverinstanz (siehe Tabelle 4.2)

Typ	Trainingsfehler (MSE)	Validierungsfehler (MSE)	Epoche
CNN	1.323	0.039	11

Tabelle 6.6: Trainings- und Validierungsfehler sowie entsprechende Epoche des CNNs für die zweite Serverinstanz. Die Fehler sind auf drei Nachkommastellen gerundet.

Typ	MSE	RMSE	R^2
CNN	0.394	0.628	-0.004

Tabelle 6.7: Testfehler des ausgewählten CNN Modells für die zweite Serverinstanz. Die Tabelle enthält die Werte für den Mean Squared Error (MSE), den Root Mean Squared Error und das Bestimmtheitsmaß (R^2). Die Werte sind auf drei Nachkommastellen gerundet. Es ist erkenntlich, dass dieses Modell eine deutlich schlechtere Vorhersage erzeugt. Der R^2 ist sogar leicht negativ.

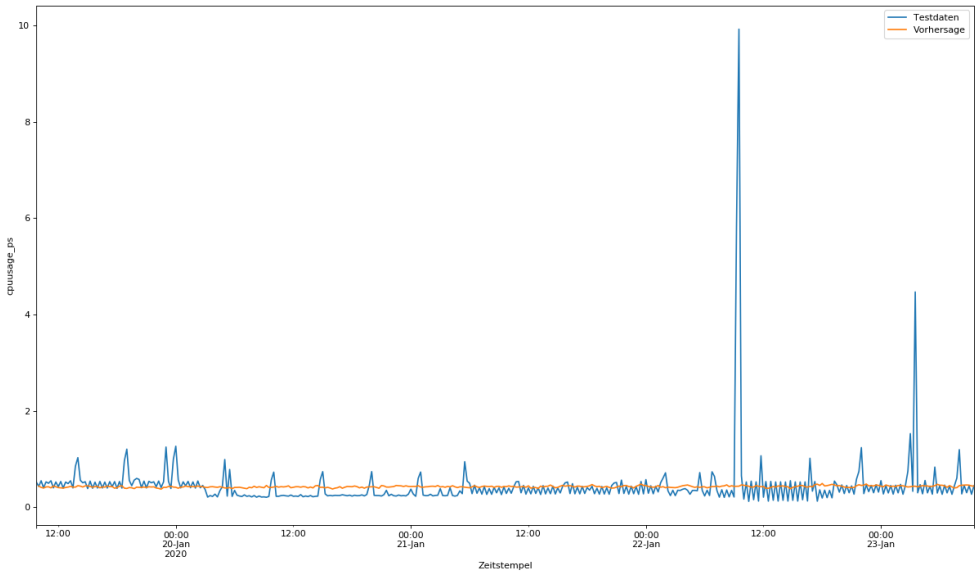

Abbildung 6.11: Plot der Vorhersage und der Testdaten für die zweite Serverinstanz. Die Vorhersage erscheint nicht sehr gut. Das Modell sagt nahezu einen konstanten Wert vorher.

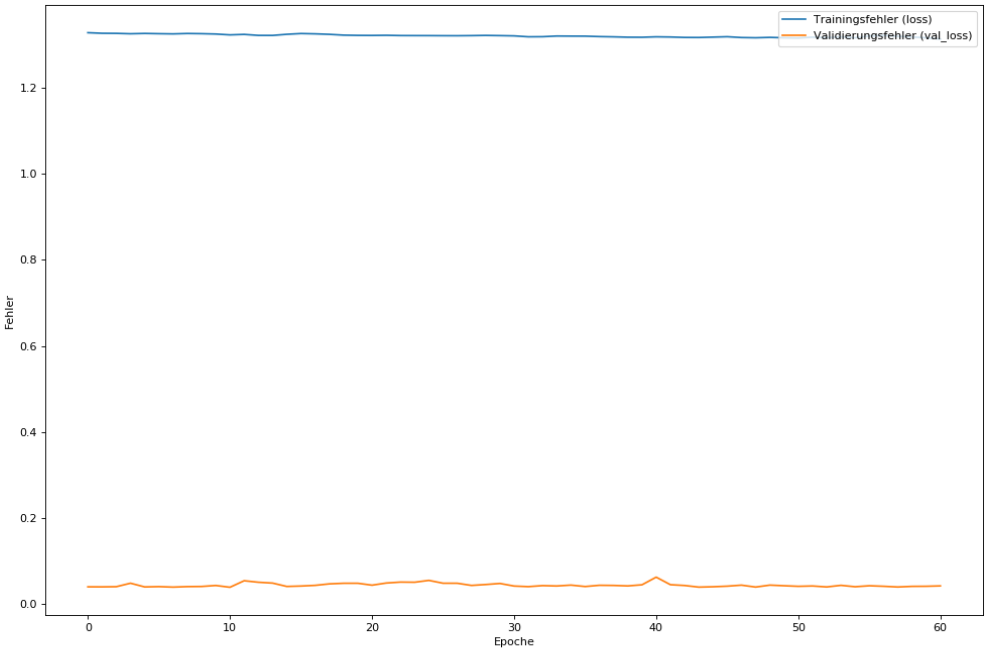

Abbildung 6.12: Fehlerverlauf beim Training des CNNs für die zweite Serverinstanz. Es ist erkennbar, dass der Validierungsfehler erneut zu jeder Zeit geringer war als der Trainingsfehler. Außerdem nimmt keiner der beiden Fehler im Verlauf des Trainings in relevantem Maße ab.

Kapitel 7

Möglichkeiten zur Integration der Prognosen in die Webanwendung

In Kapitel 3 wurde die Entscheidung für Python als Programmiersprache zur Umsetzung der Zeitreihenvorhersage erläutert. In dieser Arbeit sollen die Vorhersagen in eine Webanwendung integriert werden, und müssen daher mit dem Browser kompatibel sein. Da Python aber nicht nativ von Browsern unterstützt wird, ist es nötig, die Skripte entweder auf andere Weise im Browser auszuführen, oder die Prognosen anderweitig in der Webanwendung verfügbar zu machen, sodass diese visualisiert werden können. In diesem Kapitel werden zwei verschiedene Ansätze der Einbettung diskutiert und verglichen. Aufgrund des Vergleichs wird für den konkreten Anwendungsfall dieser Arbeit der passende Ansatz ausgewählt

7.1 Durchführen des Machine-Learning im Browser

Im Folgenden werden verschiedene Technologien vorgestellt, mit denen es möglich ist, Machine-Learning im Browser durchzuführen. Dies kann sowohl in der Form umgesetzt werden, dass das Modell Client-seitig trainiert wird, als auch dergestalt, dass der Client das bereits trainierte Modell entweder aus dem eigenen Speicher oder von extern lädt und lediglich die Prognosen erstellt.

7.1.1 Tensorflow.js

Tensorflow ist ein Framework für die Programmiersprache Python, das im Bereich Machine Learning verwendet wird und schon in Abschnitt 3.1.5 vorgestellt wurde. Es findet Anwendung in mehreren Google Diensten, bei Twitter, Airbnb, PayPal und vielen weiteren Unternehmen [11]. Tensorflow.js ist eine Javascript-Implementierung des Python-Frameworks. Damit können Machine-Learning-Modelle sowohl Client-seitig im Browser als auch Server-seitig unter Node.js erstellt und trainiert werden [95]. Trainierte Modelle können gespeichert und geladen werden [95]. Motivation für Tensorflow.js war die große und stetig wachsende Javascript-Community [90, S.

1]. Die größten Vorteile von Tensorflow.js sind die Lauffähigkeit auf nahezu allen Systemen durch die Abstraktion der Browser von der Hardware. Außerdem die Möglichkeit, Browser-APIs für beispielsweise Webcam und Mikrofon zu nutzen, sowie die Option, die Benutzerdaten ausschließlich Client-seitig zu verarbeiten und so Sicherheitsbedenken zu vermeiden [90, 107]. Um eine möglichst gute Performance zu erreichen, greift Tensorflow.js zum Ausführen der Matrixoperationen, die im Zuge der Machine-Learning-Algorithmen nötig sind, über die WebGL-API auf die Grafikkarte zu. Während die GPU rechnet, ist die CPU in der Lage, andere Operationen, zum Beispiel weiteren Javascript-Code, auszuführen [90, S. 6 f.]. Zwar ist Tensorflow.js deutlich schneller als reine Javascript Implementierungen, jedoch ist es drei bis zehn mal langsamer als die Python-Variante [90, S. 5 f.], die statt WebGL das CUDA-Toolkit von NVIDIA Grafikkarten verwendet [35].

7.1.2 WebAssembly

WebAssembly ist ein offener Standard, der es ermöglicht, neben Javascript weitere Programmiersprachen wie C und C++ im Browser ausführbar zu machen. Es ist zwar möglich, direkt in WebAssembly zu programmieren, der üblichere Weg ist aber das Kompilieren des Ausgangsprogramms nach WebAssembly, beispielsweise mittels Emscripten [65]. Eins der Hauptziele von WebAssembly ist es, eine möglichst gute Performance zu bieten [65]. WebAssembly benötigt in nahezu allen Untersuchungen im Vergleich zu nativem Code maximal die doppelte Rechenzeit [39, S. 197]. Derzeit ist es nicht ohne größere Umwege möglich, interpretierte Programmiersprachen - zu denen Python zählt - nach WebAssembly zu kompilieren. Eine neuartige Möglichkeit dafür bietet Pyodide [55], das in Abschnitt 7.1.4 vorgestellt wird. Eines der nächsten Ziele der WebAssembly-Entwickler ist es, einen Garbage-Collection-Mechanismus zu implementieren [39, S. 198]. So könnte für Programmiersprachen, die nicht hardwarenah sind, eine Kompilierung zu WebAssembly ermöglicht werden.

7.1.3 Iodide

Iodide ist eine Browser-basierte Entwicklungsumgebung, die mit dem Notebook-Prinzip ähnlich wie JupyterHub [75] funktioniert. Das Ziel der Iodide-Entwickler ist es, (daten-)wissenschaftliches Programmieren mittels Javascript im Browser zu ermöglichen [18]. Ein zentraler Unterschied zu JupyterHub ist, dass der Code und die Berechnungen Client-seitig ausgeführt werden [18] und nicht wie bei Jupyter auf einem Server in der Cloud [75]. Um die Entwicklung mit Iodide interaktiv zu gestalten, wird das Notebook in zwei Seiten aufgeteilt. Auf der linken Seite befindet sich der Editor und die rechte Seite enthält die sogenannte „Report-Preview“, die Konsole und den „Workspace“, der Informationen über die vorhandenen Variablen enthält. In der Report-Preview kann der Entwickler direkt das Ergebnis des Codes im Editor sehen [47]. In den sogenannten „chunks“, den einzelnen Abschnitten im Editor, die Schritt-für-Schritt ausgeführt werden können, kann neben Javascript auch HTML, CSS und Markdown verwendet werden. Teilt man einen Report per Link, wird der Code beim Betrachter automatisch komplett ausgeführt und das Endergebnis des Reports dargestellt [18]. Es

besteht aber trotzdem die Möglichkeit, den Editor anzeigen zu lassen und den Code zu bearbeiten [18]. Iodide befindet sich zum Zeitpunkt des Schreibens allerdings noch in einem frühen Alphazustand [48] und [18] rät von der Verwendung für kritische Systeme ab. Durch die Integration von Python wurde aus Iodide Pyodide, das im kommenden Abschnitt vorgestellt wird.

7.1.4 Pyodide

Mittels WebAssembly wurde Python in Iodide integriert und so nur wenige Wochen nach dem ursprünglichen Projekt *Pyodide* geschaffen [49]. Motivation dafür waren die Feststellungen, dass Javascript keine Bibliotheken aus dem Bereich Data Science hat, die mit denen aus Python konkurrieren können, und dass Wissenschaftler üblicherweise mit Python arbeiten [24]. Pyodide bietet über die Kombination von CPython, Emscripten und WebAssembly einen Python-Interpretierer an und enthält zum Zeitpunkt des Schreibens 35 Pakete, darunter die im Data Science Bereich besonders verbreiteten Pandas, Numpy, Matplotlib und in Teilen SciPy [49, 24]. Python kann in einem Iodide-Notebook verwendet werden, indem der Block mit `\%\% py` gekennzeichnet wird [47]. So kann letztendlich Pyodide in Iodide verwendet werden. Mit HTML, CSS und Markdown sind ansonsten die gleichen Technologien wie in Iodide verwendbar. Die Grenze zwischen Iodide und Pyodide ist fließend. Zusätzlich lassen sich über `pyodide.pyimport()` innerhalb eines Javascript-Blocks Objekte aus Python-Blöcken importieren und somit beispielsweise Funktionen oder Variablen verwenden [25]. Das verdeutlicht die Interoperabilität zwischen Javascript und Python. Pyodide-Implementierungen sind in Mozillas Firefox bis zu 12-mal und in Google Chrome bis zu 16-mal langsamer als in nativem Python [24]. Wie Iodide ist auch Pyodide noch im Alphazustand und [24] bezeichnet es als experimentelles Projekt.

7.2 Bereitstellung der Prognosewerte

Neben der Möglichkeit, das Machine-Learning Client-seitig auszuführen, besteht die Option, dem Client lediglich die Resultate daraus bereitzustellen. Dies ist die übliche Herangehensweise. Zu diesem Ergebnis kommt auch die Überlegung, dass alle in Abschnitt 7.1 vorgestellten Methoden darauf abzielen, bestehende Server-seitige Implementierungen im Browser verfügbar zu machen. Es gibt mehrere Beispiele aus der Praxis für Implementierungen dieses Ansatzes. Einige davon werden im Folgenden mit einem kurzen Blick auf die Realisierung vorgestellt.

Das amerikanische Unternehmen Uber verwendet Machine-Learning um eine Simulationsplattform zu betreiben [13]. Nachdem das Modell trainiert wurde, werden die Ergebnisse in einer Datenbank und einem gesonderten Speicher-Dienst persistiert. Anschließend kann das Backend der Software auf diese zugreifen, wenn eine Simulation durchgeführt werden soll.

Google verwendet zur Handschrifterkennung in ihrem Dienst Gboard ein RNN [30]. Die Modelle dafür werden zuerst in Tensorflow trainiert, um die Größe zu minimieren in ein Tensorflow Lite Modell konvertiert und anschließend direkt in die App integriert.

Ein weiteres Beispiel liefert Coca Cola [7]. Als die Website „MyCokeRewards.com“ durch eine Mobile-first Webapp ersetzte, benötigten Sie für eine bessere User-Experience eine einfachere Möglichkeit, um die Zahlenkombination einzugeben. Bisher war es nötig, den Code, der zur Verifikation des Kaufs verwendet wird, händisch einzugeben. Die Lösung war ein eigenes, auf Machine-Learning-basierendes OCR-System, das bessere Resultate lieferte als die erhältlichen Module. Das trainierte Modell kann anhand eines Bildes der Zahlenkombination vom Smartphone aus den Code auslesen. Kann der erkannte Code validiert werden, wird der Inhalt freigeschaltet. Andernfalls kann der Benutzer die automatisch ausgelesenen Ziffern manuell korrigieren. In diesem Fall wird diese Korrektur und das zugehörige Bild verwendet, um zukünftige Vorhersagen zu verbessern .

Auch Airbnb verwendet Künstliche Neuronale Netzwerke [105]. Das Modell wird zur Klassifizierung genutzt. Es wird erkannt, welcher Raumtyp, beispielsweise Küche oder Badezimmer, auf den hochgeladenen Bildern vorhanden ist. Da sie dieses Modell mit mehreren Millionen Bildern trainierten, war es nötig, den Vorgang zu parallelisieren und über AWS in der Cloud auszuführen. Somit ist ein Client-seitiges Training unmöglich.

In allen diesen Beispielen, wurde das Modell nicht Client-seitig trainiert. Sie belegen, dass der Ansatz, dem Client nur die Ergebnisse des Modells zur Verfügung zu stellen, nicht nur etabliert ist, sondern auch in vielen Unternehmen produktiv eingesetzt wird.

7.3 Entscheidung für einen Ansatz

Die Vorteile der in Abschnitt 7.1 erwähnten Ansätze entstehen aus der Ausführung des Machine-Learning-Codes im Browser. Würde einer dieser Ansätze in dieser Arbeit verwendet, könnte allerdings keiner der Vorteile genutzt werden und einige Merkmale würden sich gar negativ auswirken. Die Begrenzung des Machine-Learning auf einen Client hätte zur Folge, dass das Training oder zumindest das Erstellen der Prognosen, falls ein trainiertes Modell geladen würde, von jedem Client eigenständig ausgeführt werden müsste. Verschiedene Betrachter könnten unterschiedliche Prognosen erhalten. Gegenläufige Aussagen könnten Verwirrung stiften. Der große Pluspunkt dieser Methoden, dass durch den Browser eine Abstrahierung von der Hardware geschieht und mittels der Browser-APIs leicht auf Webcam und Mikrofon zugegriffen werden kann, ist für dieses Projekt irrelevant, da die Ausgangsdaten Server-seitig bereitgestellt werden müssen. Zudem sind Iodide und Pyodide noch nicht ausgereift und die Entwickler raten von einer Verwendung in kritischen Systemen ab. Ein weiterer Nachteil dieses Ansatzes ist, dass die Daten, die als Prognosegrundlage dienen, jedem Client einzeln zur Verfügung gestellt werden müssen. Dies würde unnötige Last erzeugen, sowohl während dem Transport auf der gesamten Strecke, als auch bei der anschließenden Verarbeitung beim Client.

Der native Ansatz bringt zwar den Nachteil mit sich, dass eine Serverstruktur vorhanden sein muss, um das Machine-Learning auszuführen und den Clients die Prognosen

zur Verfügung zu stellen. Das Vorhandensein einer solchen Serverstruktur ist jedoch die grundlegende Annahme dieser Arbeit. Insgesamt ist eine Server-seitige Umsetzung Ressourceneffizienter, da weniger Last auf der Transportstrecke liegt und das Modelltraining sowie die Prognosenerzeugung nur einmalig für alle Clients ausgeführt werden müssen. Auch die Autoren von [76] setzen die Integration in eine Webapp in dem Kapitel „Einbettung eines Machine-Learning-Learning-Modells in eine Webanwendung“ mittels Server-Client-Struktur um.

Abschließend lässt sich sagen, dass von den in Abschnitt 7.1 vorgestellten Möglichkeiten Tensorflow.js aufgrund der Reife und des Ursprungs in Tensorflow die sinnvollste Möglichkeit ist, Machine-Learning im Browser durchzuführen. Dennoch ist aus oben genannten Gründen im Vergleich eine Server-seitige Umsetzung, zumindest für den Anwendungsfall dieser Arbeit, die bessere Alternative. Daher fiel die Entscheidung, das Modell auf einem Server zu trainieren und auch das Erstellen der Prognose Server-seitig zu implementieren. Die Vorhersagen sollen anschließend den Clients über eine Serverschnittstelle zur Verfügung gestellt werden.

Kapitel 8

Umsetzung des gewählten Ansatzes

Es wurde entschieden, das Trainieren des Modells und das Prognostizieren auf einem Server durchzuführen und die Prognosen den Clients zur Verfügung zu stellen. Der gesamte Prozess der Arbeit wird in Abbildung 8.1 dargestellt. Im Folgenden wird die Implementierung dieses Ansatzes beschrieben und das Ergebnis vorgestellt. Dafür wird zunächst auf die Webanwendung und die verwendeten Technologien eingegangen. Anschließend wird die Architektur und das Verfahren zur Prognosenbereitstellung erläutert. Daraufhin werden die verwendeten Visualisierungstechniken sowie deren Erweiterung um die Vorhersage erklärt.

8.1 Die Webanwendung DC-Cubes

Die Webanwendung, in die die Prognosen integriert werden sollen, heißt *DC-Cubes* und wird in dieser Form seit Sommer 2019 entwickelt. Der Verwendungszweck der Anwendung ist die Analyse und Optimierung der Auslastung einer IT-Infrastruktur. Dafür sollen mehrere Metriken zu den Servern, wie zum Beispiel die RAM- und CPU-Nutzung oder die I/O-Zugriffe in die Webanwendung geladen und dort visualisiert werden. Warum das Monitoring einer IT-Infrastruktur wichtig ist, wurde in Abschnitt 1.1 dargelegt. Die Optimierungsmöglichkeiten, die aus dem Betrachten der vergangenen Daten entstehen, sollen durch die Integration von Prognosen, wie in 1.2 beschrieben, erweitert werden. Zum Zeitpunkt des Schreibens wird in dem Monitoring-Dashboard nur eine Metrik, die CPU-Auslastung, angezeigt. Abbildung A.6 (siehe Anhang) zeigt einen Screenshot der Webanwendung mit dem neuen Datensatz.

8.2 Verwendete Technologien

Im Folgenden werden die wichtigsten Technologien, die im Zusammenhang mit der Webanwendung verwendetet wurden, kurz erläutert.

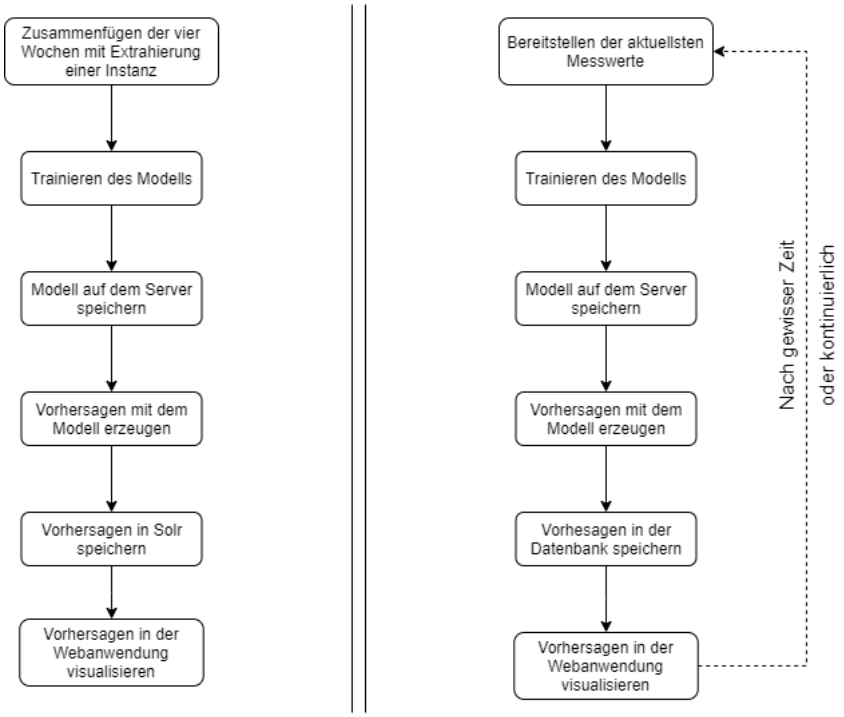

Abbildung 8.1: *Links:* Der abstrahierte Gesamtprozess dieser Arbeit von der Verarbeitung der Daten bis hin zur Visualisierung der Prognose. *Rechts:* Mit einer Verallgemeinerung der Datenverarbeitung und Datenbank kann auch der gesamte Prozess verallgemeinert werden. Außerdem lässt sich dann das iterative Verhalten des Prozesses darstellen.

8.2.1 React

Die Anwendung wurde zunächst mit dem Framework Angular umgesetzt. DC-Cubes sollte allerdings als Plugin für die Monitoring-Lösung Grafana einsetzbar gemacht werden. Da Grafana die Plugin-Spezifikation allerdings von Angular auf React geändert hatte [44], wurde die Webanwendung in React neu implementiert.

React ist eine JavaScript Bibliothek zum Erstellen von Benutzeroberflächen [91, S. 313]. Eine React-Anwendung besteht aus einer Wurzel-Komponente, in die alle anderen Komponenten gerendert werden. Jede Komponente kann beliebig viele andere Komponenten enthalten. In React sind die Komponenten in der JSX-Sprache verfasst, in der sowohl JavaScript als auch traditioneller HTML-Quellcode genutzt werden kann. Zwei weitere wichtige Konzepte bei den Komponenten sind der „state“ und die „properties“[91]. Properties werden von der Eltern- an die Kindkomponente übergeben und beeinflussen meistens das Renderverhalten bzw. den Inhalt. Der state ist der Zustand der Komponente und kann verschiedene Variablen enthalten [91]. Wird der state geändert, wird die Komponente neu gerendert [94].

8.2.2 D3.js

Die Bibliothek *D3.js* ermöglicht eine auf Daten basierte DOM-Manipulation, beispielsweise in Form von SVGs [6]. Dafür sind mehrere Funktionen implementiert, mit denen

sich unter anderem Diagramme zeichnen lassen. In DC-Cubes wird D3.js für den 2d-Plot verwendet. Die Bibliotheksfunktionen zum Zeichnen von Linienplots werden auf die vorhandenen Daten angewendet und so der 2d-Plot als SVG-Element erzeugt.

8.2.3 three.js

three.js ist eine JavaScript Bibliothek, die als Framework für WebGL fungiert [21] aber auch andere Renderer verwenden kann [70]. Somit ermöglicht sie das Erstellen von 3d-Szenen und bietet unter anderem Kontrolle über die Perspektive, das Licht und Interaktionen. In DC-Cubes wird three.js verwendet, um die 3d-Ansicht zu erzeugen. Dieser Visualisierung liegt die Stadt-Metapher zugrunde.

Stadt-Metapher in der 3d-Visualisierung

Bei der Stadt-Metapher werden die zu visualisierenden Objekte als Gebäude einer Stadt repräsentiert. Dabei können zum Beispiel die Form, Höhe und Position der Gebäude verwendet werden, um Informationen zu kodieren. Diese Metapher wurde mehrfach eingesetzt, um Software(-Systeme) zu visualisieren [71, 101] und der Vorteil gegenüber klassischen Ansichten untersucht [78]. In DC-Cubes wird jeder Server durch ein Gebäude dargestellt. Ein Block der Stadt entspricht einem Server Cluster. Ein Cluster ist ein Zusammenschluss von Servern und anderer Hardware, die nach außen wie ein Server agieren [16]. In der Anwendung existieren vier Cluster wobei jedes Cluster acht Server-Instanzen beinhaltet. Insgesamt werden also 32 Gebäude angezeigt. Die Cluster sind gleichmäßig auf zwei Rechenzentren aufgeteilt. Die Höhe eines Gebäudes spiegelt die Auslastung wider. Je höher das Gebäude, desto höher die Auslastung. Die Farbgebung soll ebenfalls Informationen repräsentieren, dies ist zum Zeitpunkt des Schreibens allerdings noch nicht implementiert.

8.2.4 Solr

Als Datenbank zum Speichern der Messwerte wird *Apache Solr* verwendet. Solr ist keine herkömmliche Datenbank, sondern eine Open-Source Suchplattform, die auf Apache Lucene aufbaut [2]. Dennoch wird in dieser Arbeit Solr auch als Datenbank bezeichnet, da es in diesem Projekt wie eine Datenbank verwendet wird und durch andere Technologien ausgetauscht werden könnte. Um Datenbank-typische CRUD-Funktionalitäten zu ermöglichen, funktioniert der Index bei Solr über Dokumente, die Felder mit jeweils Feldname und Feldinhalt enthalten [46]. Diese Funktionalitäten können über HTPP-Requests an den Solr-Server genutzt werden [92]. Ein Solr Core ist ein Index, der eine eigene Konfiguration, dazu zählt beispielsweise die Dokumentenstruktur, haben kann [93]. In DC-Cubes gab es zunächst nur einen Core für die historischen Daten. Im Zuge dieser Arbeit wurde ein weiterer Core für die Prognose hinzugefügt (siehe Abbildung 8.2 auf Seite 83). Die Konzepte aus Solr lassen sich zum Teil mit denen einer Relationalen Datenbank wie beispielsweise MySQL vergleichen. Ein Solr Core entspricht dann einer Tabelle und ein Feld einer Spalte in dieser Tabelle.

8.2.5 Node.js

Node.js ist eine asynchrone Plattform, die es ermöglicht, eventbasierten JavaScript-Code Server-seitig auszuführen [10, S. 7]. Da Node.js leichtgewichtig ist, eignet es sich besonders gut für Anwendungen mit großen Datenmengen, die in kurzer Zeit transportiert werden müssen [10, S. 8]. Daher eignet es sich für den Anwendungsfall dieser Arbeit. Aufgrund der Asynchronität kann der Node.js-Server die Daten aus Solr auch noch zur Verfügung stellen, wenn viele Clients gleichzeitig Anfragen stellen.

8.3 Backend-Architektur

Da das Backend für die Bereitstellung der Prognosen zuständig sein soll und die Monitoring-Lösung bisher nur aus der React-Webapp und einem Solr-Core bestand, war es nötig, die bestehende Architektur zu erweitern. Das Backend sollte in der Lage sein, ein Python-Skript auszuführen, welches für das Machine-Learning zuständig ist, sowie die daraus resultierenden Prognosen in einem neuen Solr-Core zu persistieren. Die Entscheidung fiel auf einen Node.js-Sever, der mittels der Bibliothek *ExpressJS* mehrere REST-Endpunkte anbietet.

8.4 Konzept zur Bereitstellung der Prognosewerte

Für die Bereitstellung der Prognosewerte musste zunächst ein grobes Konzept entwickelt werden. Dieses wird nun beschrieben. Beim Start des Servers soll eine Prognose erstellt werden. Dafür wird das trainierte Modell, das auf dem Server gespeichert ist, geladen und die historischen Daten aus der Datenbank geholt. Basierend auf diesen Daten wird mit dem Modell eine Prognose erstellt, die anschließend in der Datenbank gespeichert wird. Dort gibt es einen Core für die historischen und einen für die prognostizierten Werte. Clients können analog zu den historischen Daten die Vorhersagen aus der Datenbank abfragen. Die historischen Daten werden beim Start der Webanwendung geladen. Die Abfrage der Prognosen geschieht erst, wenn die Anzeige der Prognosen aktiviert wird. Dies wurde so mit dem Unternehmen abgestimmt. Es wäre genauso denkbar, die Prognosen ebenfalls beim Aufruf der Webanwendung zu laden. So wäre die initiale Startzeit etwas länger, dafür würde die Wartezeit für die Abfrage beim Aktivieren der Prognosenanzeige wegfallen. Außerdem bietet das Monitoring-Dashboard dem Benutzer eine Schaltfläche, um eine neue Vorhersage zu erzeugen. Das Modell wird dabei nicht neu trainiert, sondern es werden lediglich erneut die - jetzt aktualisierten - historischen Daten aus der Datenbank geholt und die neuesten Werte für die Vorhersage verwendet.

8.5 Visualisierung der Prognosen in der Webanwendung

Im Folgenden wird kurz die Ausgangssituation der Webanwendung beschrieben und anschließend erklärt, wie die Visualisierung um die Prognose erweitert werden soll.

8.5.1 Ausgangssituation

Die Entwicklung der Webanwendung seitens des Unternehmens begann schon vor dieser Arbeit. Zunächst soll die vorhandene Visualisierung, auf der die Integration aufbaut, beschrieben werden. Die Webanwendung bietet zwei Visualisierungen: eine 2d-Ansicht, die Liniendiagramme beinhaltet, sowie eine 3d-Ansicht, die die Stadtmetapher verwendet (siehe Abschnitt 8.2.3). Die 2d-Ansicht beinhaltet dabei alle Daten aus dem ausgewählten Zeitraum und visualisiert für alle Instanzen den Minimal-, Maximal- und Durchschnittswert. Damit dient sie einerseits als Überblick über den zeitlichen Verlauf der Metrik, andererseits kann sie auch als Navigation innerhalb der Zeitachse verwendet werden. Dabei ist es möglich, durch einen Klick auf eine Stelle im Plot einen Zeitpunkt auszuwählen, beziehungsweise durch ein Ziehen mit gedrückter Maustaste, eine Zeitperiode auszuwählen. Wird eine Zeitperiode ausgewählt, wird in der 3d-Ansicht die Aggregation dargestellt. Die Auswahl des Zeitpunkts bzw. -raums ist auch in der Menüleiste am oberen Rand der Anwendung möglich. Dort befindet sich auch eine Checkbox, mit der der Benutzer die Anzeige der Vorhersage aktivieren kann (siehe Abbildung A.7 im Anhang). Beim Start der Anwendung werden nur die historischen Messwerte angezeigt.

8.5.2 Erweiterung um Prognose

Im Zuge dieser Arbeit wurden die bestehenden Visualisierungen erweitert. In der Menüleiste wurde ein Knopf eingefügt, über den der Anwender manuell das Erstellen einer neuen Vorhersage $\hat{y}'$ anstoßen kann. Dies kann zum Beispiel dann sinnvoll sein, wenn die Messwerte der letzten Stunden ungewöhnlich scheinen, bzw. die Vorhersagen $\hat{y}$ nicht gut genug waren. Da diese Messwerte bei der neuen Vorhersage als Eingabe verwendet werden, und die nächsten Stunden der echten Zeit in der neuen Vorhersage relativ gesehen früher sind als in der alten, ist anzunehmen, dass $\hat{y}'$ genauer ist. Es ist vorgesehen, dass die Anwendung zunächst nur mit einem Client betrieben wird. In einem Mehr-Client-Aufbau könnte es sinnvoll sein, das Neuerstellen der Prognose restriktiver zu behandeln. Diese Option könnte zum Beispiel nur für Administratoren aktiviert sein.

Zusätzlich wird der Benutzer in der 3d-Ansicht über einen Warnhinweis darüber informiert, dass es sich bei den angezeigten Daten um eine Prognose handelt, beziehungsweise, dass der Zeitraum in der Zukunft liegt. Eine weitere Änderung ist die Anzeige der Vorhersage im 2d-Plot. Zusätzlich signalisiert eine vertikale Linie das Ende der aufgezeichneten Werte und den Beginn der Prognose. Wie oben erläutert werden in der Anwendung insgesamt 32 Server-Instanzen visualisiert. Jede Instanz braucht ein eigenes Modell, da davon ausgegangen werden muss, dass die historischen Messwerte unterschiedlich sind. In Abschnitt 4.5 wurde für alle kommenden Schritte eine Server-Instanz ausgewählt. In den Daten sind jedoch weitaus mehr als 32 Server-Instanzen vorhanden. Es gab keine Regelung, welche Instanzen angezeigt werden sollen. Zukünftig soll die Anwendung so erweitert werden, dass alle tatsächlich vorhandenen Instanzen berücksichtigt werden. Für jede angezeigte Instanz muss die Datenvorverarbeitung, das Trainieren und Speichern des Modells sowie Erzeugen der

Prognose durchgeführt werden. Für diese Arbeit wurde dieser Prozess allerdings neben der ursprünglichen Ausführung nur ein weiteres Mal durchgeführt. Die vier Cluster sind in zwei Rechenzentren aufgeteilt. Für je ein Rechenzentrum wurden demnach die historischen Daten einer Server-Instanz verwendet. Alle 16 Instanzen innerhalb eines Rechenzentrums haben in der Anwendung also die gleichen historischen Werte und verwenden für die Prognose das gleiche Modell.

8.6 Technische Umsetzung

8.6.1 Speichern der historischen Daten

Bei der technischen Umsetzung wurde davon ausgegangen, dass die Daten in dem historischen Solr Core der in Abschnitt 4.7 beschriebenen Transformation unterzogen wurden. Dieser Core wurde hierfür mit einem neuen Schema initialisiert, um diesem Datenformat gerecht zu werden, da die für die Vorhersage verwendeten Merkmale davor nicht gespeichert wurden. Außerdem mussten die Daten aus dem vierwöchigen Datensatz in Solr gespeichert werden, da bisher, wie in Abschnitt 4.2 erklärt, ein anderer Datensatz in einem anderen Format visualisiert wurde. Die zwei Instanzen, für die die Datenvorverarbeitung und Modellerstellung durchgeführt wurden, wurden für je ein Rechenzentrum verwendet. Alle 16 Instanzen eines Rechenzentrums verwenden also die historischen Daten einer Instanz aus dem Datensatz.

8.6.2 Erzeugen und Speichern der Prognosen

Um die Prognosen zu erstellen wurden die beiden trainierten Modelle auf dem Server gespeichert. Der Node.js Server erzeugt beim Start über das Paket `child_process` einen Python-Prozess, mit dem ein vorbereitetes Python-Skript ausgeführt wird. Dieses Skript holt zunächst über einen HTTP-GET-Request aus Solr die historischen Messwerte. Dann werden die Daten nach ihren jeweiligen Instanzen aufgeteilt. Für jedes Instanz werden dann die folgenden Schritte ausgeführt, wobei davon ausgegangen wird, dass die Modelle bereits über die dafür vorgesehene Keras-Funktion `load_model()` geladen sind:

1. Die Daten werden Standardisiert. Die PCA wird nicht durchgeführt, da sie sich leicht negativ auf die Genauigkeit des Modells auswirkte.
2. Die letzten Werte über den Zeitraum von einem Tag werden abgeschnitten. Dies entspricht der Eingabe für die Modelle und die Anzahl der Werte ist die gleiche, die beim Sequenzieren verwendet wurde.
3. Ausgehend von der Instanz wird ein Modell ausgewählt.
4. Mit dem Modell wird aus der Eingabe eine Vorhersage erstellt.
5. Es wird ein Pandas `DataFrame`-Objekt erzeugt, dessen Struktur dem Schema des Vorhersage Cores entspricht.
6. Die Vorhersage wird in das `DataFrame`-Objekt eingefügt und das Objekt in einer Liste gespeichert.

Nachdem dieser Ablauf für alle Instanzen durchgeführt wurde, werden die 32 `DataFrame` -Objekte aus der Liste in einen `DataFrame` konkateniert. Auf diesem `DataFrame` wird dann wie schon beim Speichern der historischen Daten über die `DataFrame.apply()` Funktion für jede Zeile eine Funktion aufgerufen, die über einen HTPP-POST-Requests die Prognose an den Solr Core für die Vorhersagen schickt. Dieser Ablauf wird in Abbildung 8.2 dargestellt.

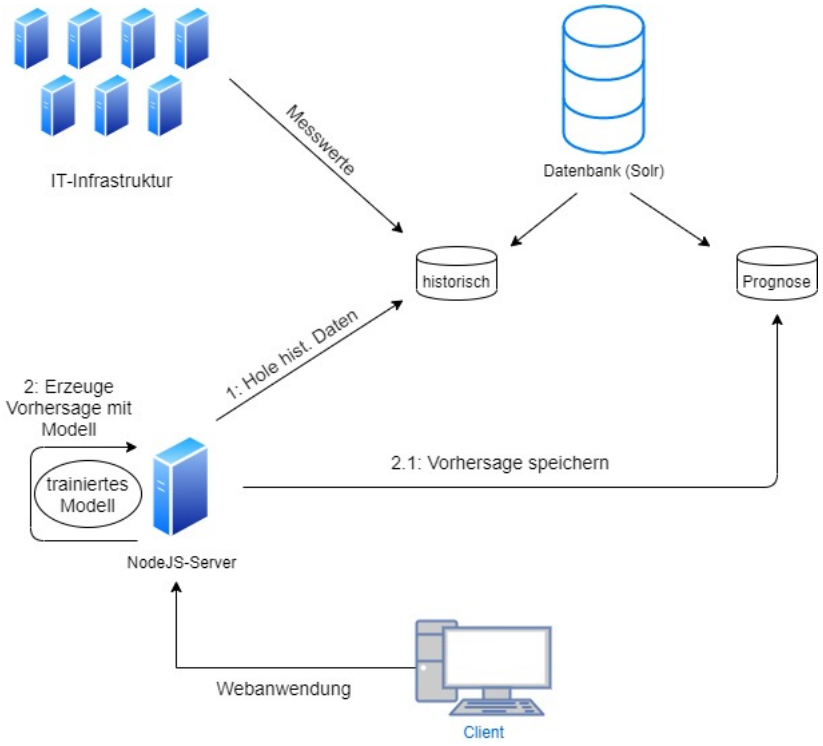

Abbildung 8.2: Architektur zur Prognosenerzeugung und -anzeige

8.6.3 Laden der Prognose in die Webanwendung

Da die Prognosen in der gleichen Form verfügbar gemacht werden wie die historischen Daten, konnten die bereits implementierten Funktionen so angepasst werden, dass sie wiederverwendet werden konnten. Auch hier werden die Daten analog zu dem Python-Skript per HTTP-GET-Request von der Datenbank abgefragt. Allerdings war es nötig, die Solr-Abfragen anzupassen. Zuvor wurden alle vorhandenen Felder und deren Werten in die Webanwendung geladen. Die Datenmenge war dafür nun zu groß. Außerdem sind alle neuen Felder, also die Metriken, die zur Vorhersage verwendet werden, für die Webanwendung aktuell irrelevant. Daher wurden die Abfragen so angepasst, dass nur noch relevante Felder, also solche, die für die Visualisierung notwendig sind, aus abgefragt werden.

8.6.4 Neue Prognosen erstellen

Über eine Schaltfläche kann ein Client das Erzeugen einer neuen Prognose anstoßen. Dafür wurde ein REST-Endpunkt `/startScript` angelegt, der das Python-Skript zur

Erstellung der Prognosen ausführt. Beim Klick auf die Schaltfläche wird ein HTTP-Request an diesen Endpunkt geschickt und somit das Python-Skript ausgeführt.

8.6.5 Anzeigen der Prognose aktivieren

Die Checkbox für die Anzeige der Prognose befindet sich in einer anderen Komponente als die tatsächliche Anzeige. Daher ist es nötig, das Event, das durch das Aktivieren der Checkbox in `QuickTimeSelection` ausgelöst wird, bis zur ersten gemeinsamen Eltern-Komponente zu propagieren, um es dann an die Anzeige-Komponente `TimeSeriesNavigationChart` weiterreichen zu können. In diesem Fall ist die Root-Komponente `App` die erste gemeinsame Eltern-Komponente. Abbildung A.8 (siehe Anhang) stellt den Ablauf dar.

8.6.6 Anzeigen der Prognose im 2d-Chart

In der Komponente `TimeSeriesNavigationChart` wird der 2d-Plot mittels der Bibliothek d3.js erstellt. Die Bibliotheksfunktionen werden zunächst genutzt, um die x- und y-Achse zu erstellen. Dabei wird je eine Achse für die historischen Daten (x_h, y_h) und eine für die aus Historie und Prognose kombinierten Daten erzeugt (x_c, y_c). Dies ist notwendig, damit im SVG immer nur der nötigste Wertebereich einer Achse angezeigt werden kann. Die Zeitachse sollte den Prognosezeitraum nur beinhalten, wenn die Prognose auch tatsächlich angezeigt wird. Eine Achse nur für die Prognose ist nicht nötig, da diese nie alleine angezeigt wird. Für die x-Achsen werden der minimale und maximale Wert aus den Daten ermittelt. Den y-Achsen wird die in den Daten vorhandene Zeitperiode zugrunde gelegt. Der Wertebereich ergibt sich für die y-Achse y_h aus dem frühesten und spätesten Zeitpunkt der historischen Daten. Für y_c aus dem frühesten Zeitstempel der historischen und dem spätesten Zeitstempel der prognostizierten Daten.

Über d3.js werden auch die Linien-Pfade für das SVG erstellt. Die SVG-Pfade werden in der Komponente gespeichert und zunächst werden für das Rendering des SVG-Elementes nur die Pfade für die historischen Daten verwendet. Wird die Prognosenanzeige aktiviert, kann der Pfad innerhalb des SVG-Elements ausgetauscht werden, ohne ihn erneut berechnen zu müssen. Bei Deaktivierung der Prognosenanzeige wird erneut der verwendete Pfad ausgetauscht. Die Komponente wird neu gerendert und im 2d-Plot werden nur noch die historischen Daten angezeigt. Der 2d-Chart mit Prognose wird in Abbildung A.9 (siehe Anhang) dargestellt. Über die Kippschalter können die minimalen, maximalen und gemittelten Kurven einzeln (de-)aktiviert werden. In Abbildung A.10 (siehe Anhang) wurden die minimale und maximale Prognose deaktiviert. Da für diese Arbeit lediglich zwei Serverinstanzen prognostiziert wurden, ist die minimale Kurve diejenige Prognose, die geringere Werte enthält. Die maximale Kurve ergibt sich dann aus dem anderen Modell. Da das zweite Modell nicht für die Daten optimiert wurde (siehe Abschnitt 6.6.5) werden auch negative Werte erzeugt. Diese sind zwar praktisch nicht möglich, werden aber dennoch so angezeigt. Ein optimiertes Modell sollte selbstverständlich nicht zu solchen Vorhersagen führen.

8.6.7 Anzeigen der Prognose im 3d-Chart

Für die Visualisierung im 3d-Chart konnten die vorhandenen Schnittstellen genutzt werden. Es wurde ein Warnhinweis hinzugefügt, der dem Nutzer signalisiert, dass gerade eine Prognose angezeigt wird. Zu diesem Zweck wird in der zuständigen Komponente der letzte historische Zeitstempel gespeichert. Falls in der 3d-Anzeige nur ein Zeitpunkt angezeigt wird, wird überprüft, ob dieser zeitlich hinter dem letzten historischen Wert liegt. Beim Anzeigen eines Zeitraums wird hingegen überprüft, ob eine der beiden Grenzen dieses Zeitraums später als der letzte historische Zeitstempel ist. Abbildung A.11 (siehe Anhang) zeigt diesen Hinweis.

8.6.8 Merge-Core

Für die Anwendung wurde noch ein dritter Core eingeführt, in dem die Indizes der beiden Cores vereint werden. Dies wurde allerdings nicht vom Autor dieser Arbeit implementiert. Dieser ist auch nicht nötig für die Funktionalität und wird deswegen nicht weiter berücksichtigt.

Kapitel 9

Zusammenfassung und Ausblick

Im Folgenden wird diese Arbeit und ihre Ergebnisse kurz zusammengefasst. Außerdem wird eine Empfehlung für den Lebenszyklus der Prognosemodelle gegeben. Schließlich wird ein Ausblick gegeben, wie die Zukunft des Monitoring-Webdashboards aussehen kann und welche Schritte dann nötig sind.

9.1 Zusammenfassung dieser Arbeit

Der Modellvergleich in dieser Arbeit zeigte, dass für die vorliegenden Daten ein CNN mit einer LSTM-Schicht zu den besten Prognosen führt. Es bleibt zu sehen, wie der Vergleich bei einem Datensatz über mehrere Monate oder gar Jahre ausfällt. Die Erprobung der Hauptkomponentenanalyse zeigte, dass es wie erwartet innerhalb der verschiedenen Metriken viele Informationen gibt, die sich in einem Raum niedriger Dimension darstellen lassen. Es ist zu erwarten, dass sich dies auch auf die Daten anderer Rechenzentren übertragen lässt. Allerdings sollte individuell evaluiert werden, ob die PCA für das Trainieren angewendet werden sollte.

Das der Arbeit zugrunde liegende Monitoring-Webdashboard ist noch nicht im Produktivsystem eingesetzt. Daher konnte die Integration auch nur soweit erfolgen, wie es mit dem Prototypen möglich war. Es wurden zwei Modelle erzeugt, sodass die Möglichkeiten der 2d- sowie 3d-Ansicht besser erkennbar sind. Dabei zeigte sich, dass es für jeden Datensatz und somit jeden Server notwendig ist, eine individuelle Modelloptimierung vorzunehmen. Die Vorhersagen wurden in die Webanwendung integriert, indem ein Server eingeführt wurde, der die trainierten Modelle lädt und die erzeugten Prognosen in einem Solr Core speichert. Dafür wurde die Solr Struktur erweitert und angepasst. Die Visualisierungen wurden mit den Prognosen angereichert. Bei der Integration wurde darauf geachtet, dass diese weitestgehend kompatibel ist mit der bestehenden Anzeige der historischen Daten. Der in dieser Arbeit angewendete Prozess lässt sich leicht so verallgemeinern, dass er auf viele Monitoring-Anwendungen appliziert werden kann (siehe Abbildung 8.1).

9.2 Lebenszyklus der Prognosemodelle

Der vorhandene Datensatz war mit einem Zeitraum von vier Wochen in seiner Aussagekraft immer noch eingeschränkt. Die Daten spiegeln nur einen kurzen Ausschnitt des Jahresverlaufes wieder und müssen nicht unbedingt für andere Monate repräsentativ sein. Des Weiteren ist die IT-Infrastruktur nicht statisch und ändert sich in vermutlich unregelmäßigen Abständen auf unbekannte Weise. Sogar wenn die Anwendung ihren Zweck erfüllt und somit zu Optimierungen führt, würden die daraus resultierenden Änderungen an der IT-Infrastruktur dazu führen, dass das Modell ungenauer wird. Denn das Modell basiert dann auf Daten, die von einer nicht mehr verwendeten IT-Infrastruktur erzeugt wurden. Änderungen an den Rechenzentren können aber auch durch den normalen Betriebsablaufs entstehen. Die verwendete Hardware oder Software könnte beispielsweise aufgrund von Investitionen oder ausgelaufenen Lizenzen geändert werden. Aus diesen Gründen ist es unumgänglich, dass das Training mit wachsender Datenmenge und Einsatzzeit erneut und mehrfach durchgeführt wird. Dafür wäre es denkbar, die erzeugten Prognosen mit den später tatsächlich eingetretenen Werten zu vergleichen. Übersteigt der Prognosefehler einen gewissen Schwellwert, sollte das Modell neu trainiert werden. In diesem Zuge könnte dann auch die in Abschnitt 6.6.4 angesprochene Neuevaluation der KNN-Typen durchgeführt werden. Eine weitere Möglichkeit besteht darin, das Trainieren als sogenanntes „Online Learning“ zu gestalten. Dabei wird das Modell kontinuierlich mit neuen Daten trainiert, ohne das Training für alle vorhandenen Daten komplett durchzuführen [82, S. 44]. Somit könnte genutzt werden, dass die verwendeten Messwerte ständig erhoben werden und zur Modellverbesserung verwendet werden können. Allerdings ist hier Vorsicht geboten, falls sich an der IT-Struktur größere Änderungen ergeben, da dann die zuvor erhobenen Daten ihre Aussagekraft verlieren und das Modell so negativ beeinflussen könnten. Dann sollten diese älteren Daten nicht mehr für das Training verwendet werden. Das Online-Learning müsste dann also abgebrochen und mit den ausschließlich neuen Daten erneut begonnen werden.

Insgesamt lässt sich also sagen, dass die Zeitreihenvorhersage im Kontext eines Rechenzentrums zwar große Vorteile bieten kann, die Realisation aufgrund der dynamischen Struktur allerdings Schwierigkeiten birgt.

9.3 Ausblick

Sollte die Webanwendung für die gesamte IT-Infrastruktur mit knapp 9.000 Servern eingesetzt werden, wird es vermutlich nötig sein, eine neue Architektur zu entwickeln. In der Webanwendung wird es dann nicht mehr möglich sein, die Rohdaten in dieser Form sinnvoll zu visualisieren. Eine Möglichkeit besteht darin, Aggregationen vorzunehmen. Aber selbst mit solchen Aggregationen muss ein Konzept entwickelt werden, dass für die dann sehr großen Datenmengen aller Server funktioniert. Auch für die Auswahl, das Trainieren und Optimieren sowie Speichern und Laden der Prognosemodelle muss für einen Anwendungsfall dieser Größenordnung ein spezielles Vorgehen entwickelt werden. Des Weiteren muss überprüft werden, ob die Visualisierungen

für eine so große Anzahl an Servern sinnvoll ist. Die 2d-Ansicht muss dann neu durchdacht werden. Werden die 9.000 Server nur durch die drei Kurven repräsentiert, gehen zu viele Informationen dabei verloren.

Im nächsten Schritt sollte die Anwendung zunächst in der bestehenden Form in das Produktivsystem integriert werden und die 32 Server visualisiert und prognostiziert werden. Zum Zeitpunkt des Schreibens gibt es dafür noch keinen festen Termin.

Literatur

[1] Albert A. Ahdoot. *Three Ways Artificial Intelligence Will Revolutionize Data Centers*. Data Center Knowledge. 14. Mai 2019. URL: https://www.datacenterknowledge.com/industry-perspectives/three-ways-artificial-intelligence-will-revolutionize-data-centers (besucht am 11. 03. 2020).

[2] *Apache Solr Reference Guide | Apache Solr Reference Guide 8.4*. URL: http://lucene.apache.org/solr/guide/8_4/ (besucht am 11. 03. 2020).

[3] Bundesagentur für Arbeit. *Geschäftsbericht 2018*. 2018.

[4] Yoshua Bengio, Patrice Simard und Paolo Frasconi. „Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), S. 157–166.

[5] Joe Blitzstein und Hanspeter Pfister. „CS109 Data Science". CS109 Data Science. Harvard University, 3. Sep. 2015. URL: http://cs109.github.io/2014/index.html (besucht am 11. 03. 2020).

[6] Mike Bostock. *D3.js - Data-Driven Documents*. URL: https://d3js.org/ (besucht am 11. 03. 2020).

[7] Patrick Brandt. *How Machine Learning with TensorFlow Enabled Mobile Proof-Of-Purchase at Coca-Cola*. Google Developers Blog. 21. Sep. 2017. URL: https://developers.googleblog.com/2017/09/how-machine-learning-with-tensorflow.html (besucht am 11. 03. 2020).

[8] Heinrich Braun. *Neuronale Netze*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. ISBN: 978-3-642-64535-8 978-3-642-60743-1.

[9] Rüdiger W. Brause. *Neuronale Netze: Eine Einführung in die Neuroinformatik*. Bearb. von Hans-Jürgen Appelrath u. a. Leitfäden der Informatik. Wiesbaden: Vieweg+Teubner Verlag, 1995. ISBN: 978-3-519-12247-0 978-3-322-93994-4.

[10] Mike Cantelon u. a. *Node. js in Action*. Manning Greenwich, 2014.

[11] *Case studies | TensorFlow*. URL: https://www.tensorflow.org/about/case-studies (besucht am 11. 03. 2020).

[12] Sucheta Chauhan und Lovekesh Vig. „Anomaly detection in ECG time signals via deep long short-term memory networks". In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2015, S. 1–7.

[13] Haoyang Chen und Wei Wang. *Gaining Insights in a Simulated Marketplace with Machine Learning at Uber*. Uber Engineering Blog. 24. Juni 2019. URL: https://eng.uber.com/simulated-marketplace/ (besucht am 11. 03. 2020).

[14] Kyunghyun Cho u. a. „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation“. In: *arXiv:1406.1078 [cs, stat]* (2. Sep. 2014).

[15] Junyoung Chung u. a. „Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling“. In: *arXiv:1412.3555 [cs]* (11. Dez. 2014).

[16] *Clustering: A basic 101 tutorial*. 3. Apr. 2002. URL: www.ibm.com/developerworks/aix/tutorials/clustering/clustering.html (besucht am 11. 03. 2020).

[17] Catherin Collin u. a. *Das Psychologie-Buch*. Dorling Kindersley Verlag, Sep. 2012. 352 S. ISBN: 987-3-8310-2209-0.

[18] Brendan Colloran. *Iodide: an experimental tool for scientific communication and exploration on the web – Mozilla Hacks - the Web developer blog*. Mozilla Hacks – the Web developer blog. 12. März 2019. URL: https://hacks.mozilla.org/2019/03/iodide-an-experimental-tool-for-scientific-communicatiodide-for-scientific-communication-exploration-on-the-web (besucht am 11. 03. 2020).

[19] Javier Contreras u. a. „ARIMA models to predict next-day electricity prices“. In: *IEEE transactions on power systems* 18.3 (2003). Publisher: IEEE, S. 1014–1020.

[20] *csv — CSV File Reading and Writing — Python 3.7.4 documentation*. 30. Sep. 2019. URL: https://docs.python.org/3/library/csv.html (besucht am 11. 03. 2020).

[21] Brian Danchilla. „Three.js Framework“. In: *Beginning WebGL for HTML5*. Hrsg. von Brian Danchilla. Berkeley, CA: Apress, 2012, S. 173–203. ISBN: 978-1-4302-3997-0.

[22] *Database Concepts*. Oracle Help Center. URL: https://docs.oracle.com/cd/B28359_01/server.111/b28318/memory.htm#CNCPT802 (besucht am 11. 03. 2020).

[23] Pedro Domingos. „A Few Useful Things to Know About Machine Learning“. In: *Commun. ACM* 55.10 (Okt. 2012), S. 78–87. DOI: 10.1145/2347736.2347755.

[24] Michael Droettboom. *Pyodide: Bringing the scientific Python stack to the browser – Mozilla Hacks - the Web developer blog*. Mozilla Hacks – the Web developer blog. 16. Apr. 2019. URL: https://hacks.mozilla.org/2019/04/pyodide-bringing-the-scientific-python-stack-to-the-browser (besucht am 11. 03. 2020).

[25] Michael Droettboom, Roman Yurchak und 0xflotus. *iodide-project/pyodide API-Reference*. GitHub. 16. Mai 2019. URL: https://github.com/iodide-project/pyodide/blob/master/docs/api_reference.md (besucht am 11. 03. 2020).

[26] Volkan Ş. Ediger und Sertaç Akar. „ARIMA forecasting of primary energy demand by fuel in Turkey“. In: *Energy Policy* 35.3 (1. März 2007), S. 1701–1708. ISSN: 0301-4215. DOI: 10.1016/j.enpol.2006.05.009.

[27] Thomas Elliot. *Aktuelles von Octoverse: Maschinelles Lernen*. Der GitHub Blog. 24. Jan. 2019. URL: https://github.blog/de/2019-01-24-the-state-of-the-octoverse/ (besucht am 11. 03. 2020).

[28] *Enterprise Manager Cloud Control Management Repository Views Reference*. URL: https://docs.oracle.com/cd/E73210_01/EMVWS/GUID-FEFC509F-FAD8-4374-AB96-67D83601F8EE.htm#EMVWS12354 (besucht am 11. 03. 2020).

[29] Ludwig Fahrmeir u. a. *Statistik*. Springer-Lehrbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. ISBN: 978-3-662-50371-3 978-3-662-50372-0.

[30] Sandro Feuz und Pedro Gonnet. *RNN-Based Handwriting Recognition in Gboard*. Google AI Blog. 7. März 2019. URL: http://ai.googleblog.com/2019/03/rnn-based-handwriting-recognition-in.html (besucht am 11. 03. 2020).

[31] Rui Fu, Zuo Zhang und Li Li. „Using LSTM and GRU neural network methods for traffic flow prediction“. In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC). Wuhan, Hubei Province, China: IEEE, Nov. 2016, S. 324–328. ISBN: 978-1-5090-4423-8. DOI: 10.1109/YAC.2016.7804912.

[32] Felix A. Gers und Jürgen Schmidhuber. „Recurrent nets that time and count“. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Bd. 3. IEEE, 2000, S. 189–194.

[33] Felix A. Gers, Jürgen Schmidhuber und Fred Cummins. „Learning to forget: Continual prediction with LSTM“. In: (1999).

[34] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 10. Nov. 2016. 801 S. ISBN: 978-0-262-33737-3.

[35] *GPU support*. TensorFlow. URL: https://www.tensorflow.org/install/gpu (besucht am 11. 03. 2020).

[36] Albert Greenberg u. a. „The cost of a cloud: research problems in data center networks“. In: *ACM SIGCOMM Computer Communication Review* 39.1 (31. Dez. 2008), S. 68. ISSN: 01464833. DOI: 10.1145/1496091.1496103.

[37] Klaus Greff u. a. „LSTM: A search space odyssey“. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), S. 2222–2232.

[38] Wolfgang Groß u. a. „Predicting Time Series with Space-Time Convolutional and Recurrent Neural Networks.“ In: *ESANN*. 2017.

[39] Andreas Haas u. a. „Bringing the Web Up to Speed with WebAssembly“. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2017. New York, NY, USA: ACM, 2017, S. 185–200. ISBN: 978-1-4503-4988-8. DOI: 10.1145/3062341.3062363.

[40] Norbert Henze. „Der Erwartungswert“. In: *Stochastik für Einsteiger: Eine Einführung in die faszinierende Welt des Zufalls*. Hrsg. von Norbert Henze. Wiesbaden: Springer Fachmedien, 2013, S. 78–85. ISBN: 978-3-658-03077-3.

[41] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-Term Memory“. In: *Neural Computation* 9.8 (1. Nov. 1997), S. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[42] Mél Hogan. „Data Center“. In: *Encyclopedia of Big Data*. Hrsg. von Laurie A. Schintler und Connie L. McNeely. Cham: Springer International Publishing, 2018, S. 1–4. ISBN: 978-3-319-32001-4.

[43] Heinrich Holland und Kurt Scharnbacher. *Grundlagen der Statistik*. Wiesbaden: Gabler, 2010. ISBN: 978-3-8349-2010-2 978-3-8349-8999-4.

[44] Peter Holmberg. *Writing React Plugins*. Grafana Labs. 26. März 2019. URL: https://grafana.com/blog/2019/03/26/writing-react-plugins/ (besucht am 11. 03. 2020).

[45] Mai Ibrahim, Marwan Torki und Mustafa ElNainay. „CNN based indoor localization using RSS time-series“. In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, S. 01044–01049.

[46] *Introduction to Solr Indexing | Apache Solr Reference Guide 8.4*. URL: http://lucene.apache.org/solr/guide/8_4/introduction-to-solr-indexing.html#introduction-to-solr-indexing (besucht am 11. 03. 2020).

[47] *Iodide-Demo*. URL: https://alpha.iodide.io/tryit (besucht am 11. 03. 2020).

[48] *iodide-project/iodide*. 3. Sep. 2019. URL: https://github.com/iodide-project/iodide (besucht am 11. 03. 2020).

[49] *iodide-project/pyodide*. 2. Sep. 2019. URL: https://github.com/iodide-project/pyodide (besucht am 11. 03. 2020).

[50] Ameet V Joshi. *Machine Learning and Artificial Intelligence*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-26621-9 978-3-030-26622-6.

[51] Naveen Joshi. *Exploring The Impact Of AI In The Data Center*. Forbes. 31. Mai 2019. URL: https://www.forbes.com/sites/cognitiveworld/2019/05/31/exploring-the-impact-of-ai-in-the-data-center/ (besucht am 11. 03. 2020).

[52] Michael S. Kaylen. „Vector autoregression forecasting models: Recent developments applied to the US hog market“. In: *American Journal of Agricultural Economics* 70.3 (1988), S. 701–712.

[53] Nikhil Ketkar. *Deep Learning with Python*. Berkeley, CA: Apress, 2017. ISBN: 978-1-4842-2765-7 978-1-4842-2766-4. (Besucht am 25. 12. 2019).

[54] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *arXiv:1412.6980 [cs]* (29. Jan. 2017).

[55] Almar Klein. *Python and WebAssembly*. URL: https://almarklein.org/python_and_webassembly.html (besucht am 11. 03. 2020).

[56] Bernd Klein. *Numerisches Python: Arbeiten mit NumPy, Matplotlib und Pandas*. München: Carl Hanser Verlag GmbH & Co. KG, 11. Juni 2019. ISBN: 978-3-446-45076-9 978-3-446-45363-0.

[57] Amit Konar. *Time-series prediction and applications*. Intelligent systems reference library. Cham: Springer, 2017. ISBN: 978-3-319-54597-4.

[58] *Konjunkturdaten im Überblick - Handelsblatt Online*. Library Catalog: www.handelsblatt.com. URL: https://www.handelsblatt.com/politik/konjunktur/konjunktur-daten/ (besucht am 11. 03. 2020).

[59] Sumit Kumar u. a. „Energy Load Forecasting using Deep Learning Approach-LSTM and GRU in Spark Cluster“. In: *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*. 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT). ISSN: null. Jan. 2018, S. 1–4. DOI: 10.1109/EAIT.2018.8470406.

[60] Bernd Leiner. *Grundlagen der Zeitreihenanalyse*. 4., völlig neu bearbeitete Auflage. Reprint 2018. Berlin, Boston: De Gruyter, 2018. ISBN: 978-3-486-24756-5. DOI: 10.1515/9783486797466.

[61] Marco Lippi, Matteo Bertini und Paolo Frasconi. „Short-Term Traffic Flow Forecasting: An Experimental Comparison of Time-Series Analysis and Supervised Learning“. In: *IEEE Transactions on Intelligent Transportation Systems* 14.2 (Juni 2013), S. 871–882. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2013.2247040.

[62] Zachary C. Lipton u. a. „Learning to diagnose with LSTM recurrent neural networks“. In: *arXiv preprint arXiv:1511.03677* (2015).

[63] Spyros Makridakis, Evangelos Spiliotis und Vassilios Assimakopoulos. „Statistical and Machine Learning forecasting methods: Concerns and ways forward“. In: *PLOS ONE* 13.3 (27. März 2018). Hrsg. von Alejandro Raul Hernandez Montoya, e0194889. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0194889.

[64] Pankaj Malhotra u. a. „Long short term memory networks for anomaly detection in time series“. In: *Proceedings*. Bd. 89. Presses universitaires de Louvain, 2015.

[65] MaxGraey. *WebAssembly Concepts*. MDN Web Docs. 25. Aug. 2019. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts (besucht am 11. 03. 2020).

[66] Wes Mckinney. „pandas: a Foundational Python Library for Data Analysis and Statistics“. In: *Python High Performance Science Computer* (1. Jan. 2011).

[67] Aidan Meyler, Geoff Kenny und Terry Quinn. „Forecasting Irish inflation using ARIMA models“. In: (1998).

[68] Maika Möbus. *TIOBE-Index: Python ist Programmiersprache des Jahres 2018*. entwickler.de. 9. Jan. 2019. URL: https://entwickler.de/online/development/python-programmiersprache-2018-579873748.html (besucht am 11. 03. 2020).

[69] Jojo Moolayil. *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*. Berkeley, CA: Apress, 2019. ISBN: 978-1-4842-4239-1 978-1-4842-4240-7.

[70] Mr.doob. *mrdoob/three.js*. 17. Feb. 2020. URL: https://github.com/mrdoob/three.js (besucht am 11. 03. 2020).

[71] Thomas Panas, R. Berrigan und John Grundy. „A 3D metaphor for software production visualization“. In: 16. Aug. 2003, S. 314–319. ISBN: 978-0-7695-1988-3. DOI: 10.1109/IV.2003.1217996.

[72] Mathias Parbel. *Umfrage: Python ist populärste Programmiersprache für Data Science*. heise online. 7. Feb. 2019. URL: https://www.heise.de/developer/meldung/Umfrage-Python-ist-populaerste-Programmiersprache-fuer-Data-Science-4300782.html (besucht am 11. 03. 2020).

[73] Razvan Pascanu, Tomas Mikolov und Yoshua Bengio. „On the difficulty of training Recurrent Neural Networks“. In: *arXiv:1211.5063 [cs]* (21. Nov. 2012).

[74] Fabian Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (Oct 2011), S. 2825–2830. ISSN: ISSN 1533-7928.

[75] *Project Jupyter*. URL: https://www.jupyter.org (besucht am 11. 03. 2020).

[76] Sebastian Raschka. *Machine learning mit Python und Scikit-learn und TensorFlow*. 2., aktualisierte und erweiterte Auflage. Frechen: mitp, 2018. 577 Seiten. ISBN: 978-3-95845-733-1.

[77] *Recurrent Layers - Keras Documentation*. URL: https://keras.io/layers/recurrent/ (besucht am 11. 03. 2020).

[78] Simone Romano u. a. „The city metaphor in software visualization: feelings, emotions, and thinking“. In: *Multimedia Tools and Applications* 78.23 (Dez. 2019), S. 33113–33149. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-019-07748-1.

[79] F. Rosenblatt. „The perceptron: A probabilistic model for information storage and organization in the brain“. In: *Psychological Review* 65.6 (1958), S. 386–408. ISSN: 1939-1471(Electronic),0033-295X(Print). DOI: 10.1037/h0042519.

[80] Adriane Rüdiger. *Was ist Hochverfügbarkeit*. DataCenter-Insider. 7. Mai 2019. URL: https://www.datacenter-insider.de/was-ist-hochverfuegbarkeit-a-821602/ (besucht am 11. 03. 2020).

[81] Ziya Şanal. „Differentialrechnung für multivariable Funktionen“. In: *Mathematik für Ingenieure: Grundlagen - Anwendungen in Maple*. Hrsg. von Ziya Şanal. Wiesbaden: Springer Fachmedien, 2015, S. 527–602. ISBN: 978-3-658-10642-3.

[82] Dipanjan Sarkar, Raghav Bali und Tushar Sharma. *Practical Machine Learning with Python*. Berkeley, CA: Apress, 2018. ISBN: 978-1-4842-3206-4 978-1-4842-3207-1.

[83] Tom Schaul, Ioannis Antonoglou und David Silver. „Unit Tests for Stochastic Optimization“. In: *arXiv:1312.6055 [cs]* (25. Feb. 2014).

[84] Rainer Schlittgen. *Angewandte Zeitreihenanalyse*. Lehr- und Handbücher der Statistik. München [u.a.]: Oldenbourg, 2001. ISBN: 978-3-486-71095-3.

[85] Rainer Schlittgen. *Zeitreihenanalyse*. 9. Aufl. Lehr- und Handbücher der Statistik. München: Oldenbourg Wissenschaftsverlag, 2001. ISBN: 978-3-486-71096-0.

[86] Heidrun Schumann und Wolfgang Müller. „Einleitung“. In: *Visualisierung: Grundlagen und allgemeine Methoden*. Hrsg. von Heidrun Schumann und Wolfgang Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, S. 1–3. ISBN: 978-3-642-57193-0.

[87] Skipper Seabold und Josef Perktold. „statsmodels: Econometric and statistical modeling with python“. In: *9th Python in Science Conference*. 2010.

[88] Sreelekshmy Selvin u. a. „Stock price prediction using LSTM, RNN and CNN-sliding window model“. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Sep. 2017, S. 1643–1647. DOI: 10.1109/ICACCI.2017.8126078.

[89] Bin Shi und S. S. Iyengar. *Mathematical Theories of Machine Learning - Theory and Applications*. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-17075-2 978-3-030-17076-9.

[90] Daniel Smilkov u. a. „TensorFlow.js: Machine Learning for the Web and Beyond“. In: *arXiv:1901.05350 [cs]* (16. Jan. 2019).

[91] Preston So. „React“. In: *Decoupled Drupal in Practice: Architect and Implement Decoupled Drupal Architectures Across the Stack*. Hrsg. von Preston So. Berkeley, CA: Apress, 2018, S. 313–334. ISBN: 978-1-4842-4072-4.

[92] *Solr Cores and solr.xml | Apache Solr Reference Guide 8.4*. URL: https://lucene.apache.org/solr/guide/8_4/solr-cores-and-solr-xml.html (besucht am 11. 03. 2020).

[93] *Solr Tutorial | Apache Solr Reference Guide 8.4*. URL: https://lucene.apache.org/solr/guide/8_4/solr-tutorial.html (besucht am 11. 03. 2020).

[94] *State and Lifecycle – React Docs*. URL: https://reactjs.org/docs/state-and-lifecycle.html (besucht am 11. 03. 2020).

[95] *tensorflow/tfjs*. 3. Sep. 2019. URL: https://github.com/tensorflow/tfjs (besucht am 11. 03. 2020).

[96] *TIOBE Index | TIOBE - The Software Quality Company*. URL: https://www.tiobe.com/tiobe-index/ (besucht am 11. 03. 2020).

[97] Michel Verleysen und Damien François. „The curse of dimensionality in data mining and time series prediction“. In: *International Work-Conference on Artificial Neural Networks*. Springer, 2005, S. 758–770.

[98] Jürgen Vogel. *Prognose von Zeitreihen*. 1. Aufl. SpringerLink : Bücher. Wiesbaden: Springer Gabler, 2015. ISBN: 978-3-658-06836-3.

[99] Ramon Wartala. *Praxiseinstieg Deep Learning: Mit Python, Caffe, TensorFlow und Spark eigene Deep-Learning-Anwendungen erstellen*. O'Reilly, 2. Jan. 2018. 296 S. ISBN: 978-3-96010-157-4.

[100] Gail Weiss, Yoav Goldberg und Eran Yahav. „On the Practical Computational Power of Finite Precision RNNs for Language Recognition“. In: *arXiv:1805.04908 [cs, stat]* (13. Mai 2018).

[101] Richard Wettel und Michele Lanza. „Visualizing Software Systems as Cities“. In: *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. Banff, AB, Canada: IEEE, Juni 2007, S. 92–99. ISBN: 978-1-4244-0599-2 978-1-4244-0600-5. DOI: 10.1109/VISSOF.2007.4290706.

[102] *Who is using scikit-learn? — scikit-learn 0.21.3 documentation*. URL: https://scikit-learn.org/stable/testimonials/testimonials.html (besucht am 11. 03. 2020).

[103] Andreas Wierse und Till Riedel. *Smart Data Analytics, Mit Hilfe von Big Data Zusammenhänge erkennen und Potentiale nutzen*. Berlin, Boston: De Gruyter Oldenbourg, 2017. ISBN: 978-3-11-046184-8. DOI: 10.1515/9783110463958.

[104] Sibylle Wilke. *Trends der Lufttemperatur*. Umweltbundesamt. Library Catalog: www.umweltbundesamt.de Publisher: Umweltbundesamt. 26. Juni 2013. URL: https://www.umweltbundesamt.de/daten/klima/trends-der-lufttemperatur (besucht am 11. 03. 2020).

[105] Shijing Yao, Qiang Zhu und Phillipe Siclait. *Categorizing Listing Photos at Airbnb*. Medium. 25. Mai 2018. URL: https://medium.com/airbnb-engineering/categorizing-listing-photos-at-airbnb-f9483f3ab7e3 (besucht am 11. 03. 2020).

[106] Xiao-Hu Yu, Guo-An Chen und Shi-Xin Cheng. „Dynamic learning rate optimization of the backpropagation algorithm“. In: *IEEE Transactions on Neural Networks* 6.3 (Mai 1995), S. 669–677. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/72.377972.

[107] Oliver Zeigermann. *Machine Learning im Browser: Was mit TensorFlow.js alles möglich ist*. entwickler.de. 5. Nov. 2018. URL: https://entwickler.de/online/machine-learning/tensorflow-js-browser-579865069.html (besucht am 11. 03. 2020).

Abbildungsverzeichnis

2.1 Temperaturverlauf in Deutschland. 8
2.2 Zahl der Arbeitslosen. 9
2.3 Zerlegung der Zeitreihe cpuusage_ps in ihre Komponenten. 10
2.4 Korrelogramm für die Zeitreihe cpuusage_ps. 12
2.5 Partielle Autokorrelationsfunktion für die Zeitreihe cpuusage_ps. 13

4.1 Der Data Science Prozess. 18
4.2 Plot zu dem ersten Datensatz. 20
4.3 Vergleich der Datensatzformate. 21
4.4 Plot der CPU-Auslastung. 26

5.1 Das Perzeptron nach Rosenblatt. 30
5.2 Schematischer Aufbau eines Künstlichen Neuronalen Netzes. 31
5.3 Die Sigmoid-Funktion. 33
5.4 Der Tangens hyperbolicus. 33
5.5 Die ReLU-Funktion. 34
5.6 Das Bias/Variance dilemma. 39

6.1 Prinzip der eindimensionalen Faltung für einen Kernel der Größe 3. . . . 45
6.2 Aufbau eines Recurrent Neural Networks nach [50, S. 124] 46
6.3 Aufbau eines LSTM-Blocks. 49
6.4 Gated Recurrent Unit 52
6.5 Vorbereitung des Datensatzes für die Vorhersage. 54
6.6 Das Sliding Window Prinzip. 58
6.7 Das Prinzip der Kreuzvalidierung aus [82, S. 290]. 60
6.8 Fehlerverlauf beim Training des CNN. 66
6.9 Vergleich der Vorhersage mit den Testdaten. 67
6.10 Plot der CPU-Auslastung für eine zweite Serverinstanz. 69
6.11 Plot der Vorhersage für die zweite Serverinstanz. 70
6.12 Fehlerverlauf beim Training des CNNs für die zweite Serverinstanz. . . 71

8.1 Gesamtprozess dieser Arbeit. 78
8.2 Architektur zur Prognosenerzeugung und -anzeige 83

A.1 Zerlegung der Zeitreihe cpuusage_ps in ihre Komponenten. 101
A.2 Plot der PGA-Nutzung. 102

A.3 Plot der Session-Nutzung. 102
A.4 Beispielhafte Darstellung der Faltung. 103
A.5 Plot der Metrik *dbCpuPs* 104
A.6 Vollansicht der Webanwendung. 104
A.7 Checkbox zur Aktivierung der Prognosenanzeige. 104
A.8 Propagieren des Checkbox-Events. 105
A.9 2d-Chart mit Prognose. 105
A.10 2d-Chart mit Prognose. 105
A.11 Prognosenanzeige im 3d-Chart. 106

Tabellenverzeichnis

4.1 Prinzip zur Vereinheitlichung der Zeitstempel. 24
4.2 Kennzahlen zur Zeitreihe cpuusage_ps. 25

6.1 Auswirkung der PCA auf das Training. 54
6.2 Die mit Grid Search optimierten Hyperparameter. Es fällt auf, dass in den meisten Fällen der *tanh* die beste Aktivierungsfunktion war. 63
6.3 Vergleich der Fehler der Modelle. 65
6.4 Testfehler des ausgewählten CNN Modells. 67
6.5 Kennzahlen zur Zeitreihe cpuusage_ps. 69
6.6 Trainings- und Validierungsfehler für die zweite Serverinstanz. 69
6.7 Testfehler des ausgewählten CNN Modells für die zweite Serverinstanz. 69

B.1 Bedeutung und Relevanz der Spalten im Datensatz. 108

Listings

C.1 Beispielhafte Dropout-Schicht in Keras 109
C.2 Implementierung des MLP mit Keras. 109
C.3 Implementierung des CNN mit Keras. 109
C.4 Implementierung des LSTM-Netzes mit Keras. 110
C.5 Implementierung des GRU-Netzes mit Keras. 110
C.6 Vereinheitlichen der Zeitstempel 111
C.7 Entfernen überflüssiger Zeilen 113
C.8 Transformieren der Features von Zeilen zu Spalten 113

Anhang A

Abbildungen

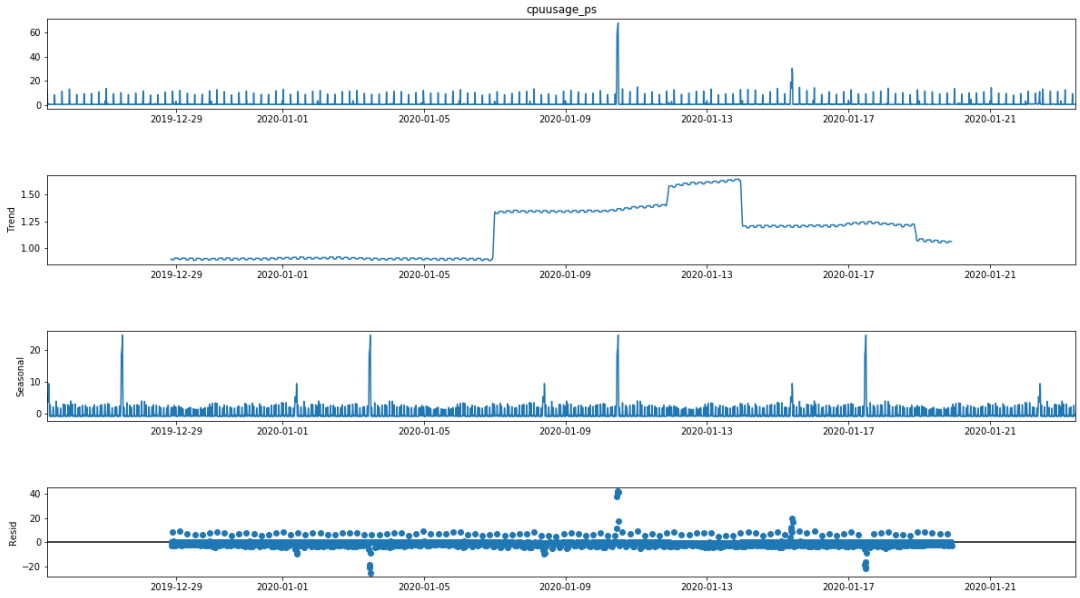

Abbildung A.1: Zerlegung der Zeitreihe cpuusage_ps in die Komponenten Trend, Saisonalität und Rest mit wöchentlicher Frequenz. Es kein klarer Aufwärts- oder Abwärtstrend erkennbar, was aber auch schon der Plot der Zeitreihe selbst zeigt. Eine Saisonalität ist festzustellen. Dennoch liegen mit der Rest-Komponente unerklärliche Einflüsse vor.

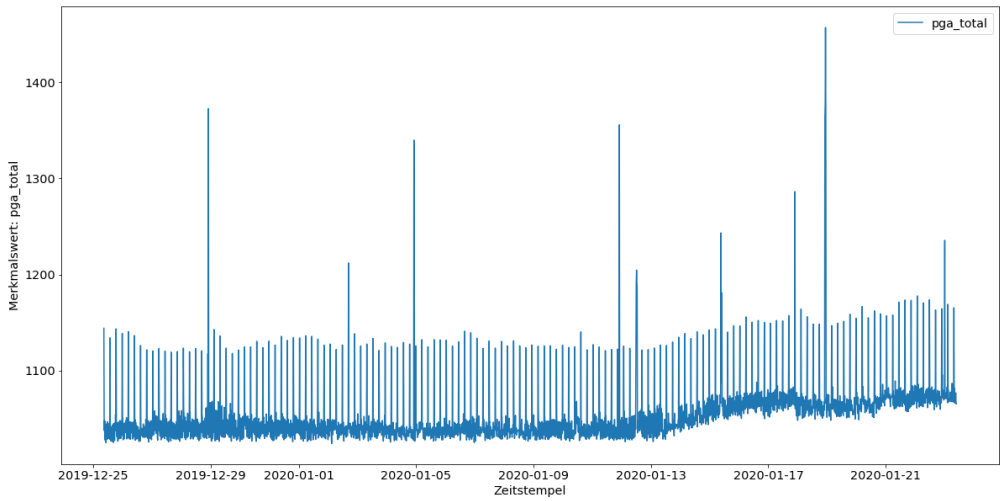

Abbildung A.2: Plot der PGA-Nutzung aus dem vierwöchigen Datensatz. Verwendete Metrik: pga_total. Ab dem 13. Januar ist ein steigender Trend erkennbar.

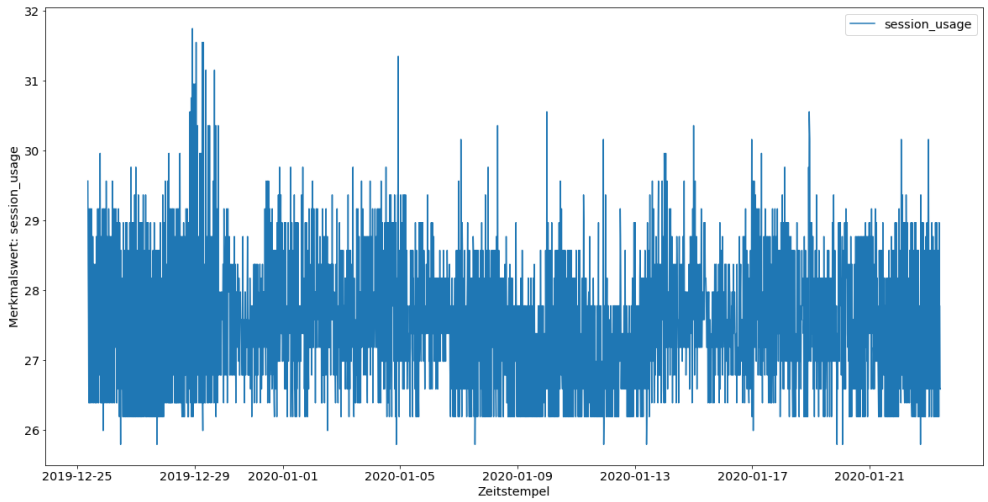

Abbildung A.3: Plot der Session-Nutzung aus dem vierwöchigen Datensatz. Verwendete Metrik: session_usage.

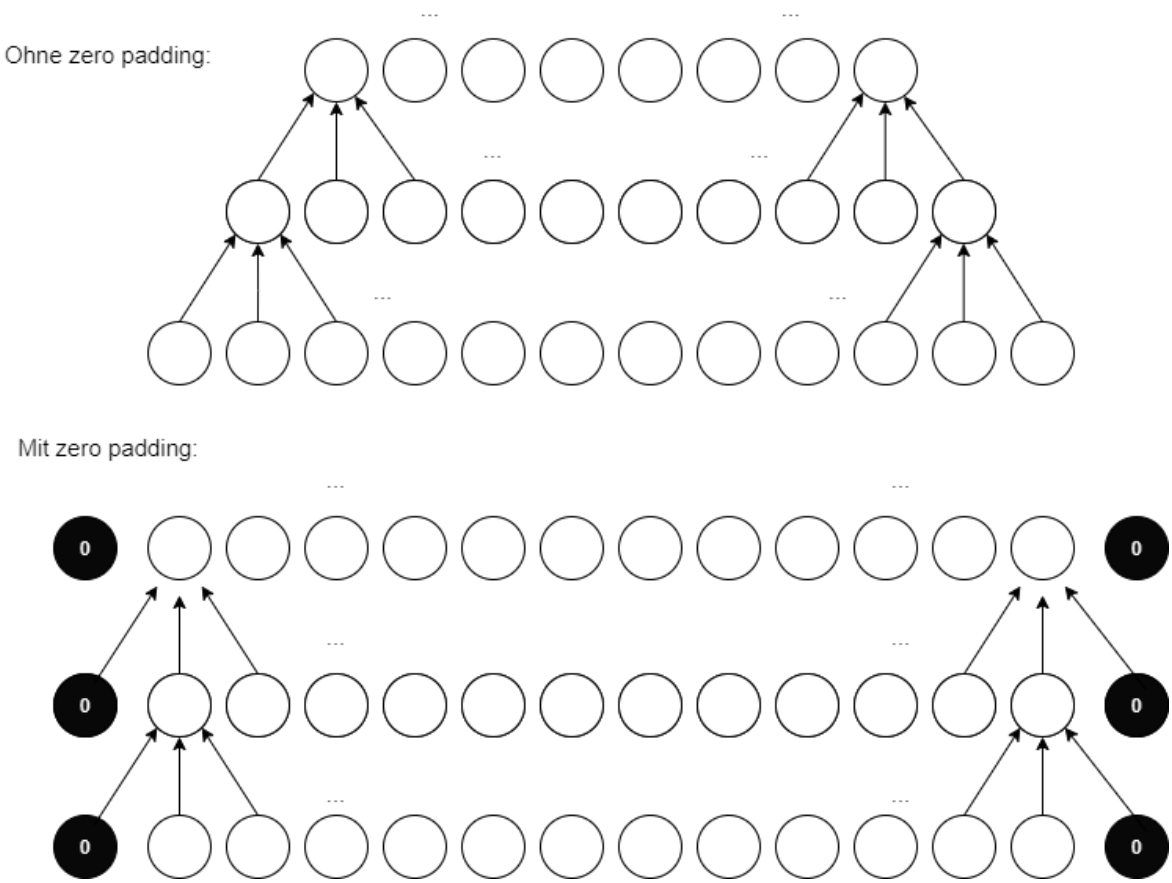

Abbildung A.4: Beispielhafte Darstellung der Faltung nach [34, S. 389] mit einem Kernel der Größe 3. *Oben:* Faltung ohne zero padding. Die Dimension der Daten schrumpft mit jeder Schicht. *Unten:* Faltung mit zero padding. Die schwarzen Elemente stellen die durch das Padding hinzugefügten Nullen dar. Die Dimension der Daten bleibt erhalten.

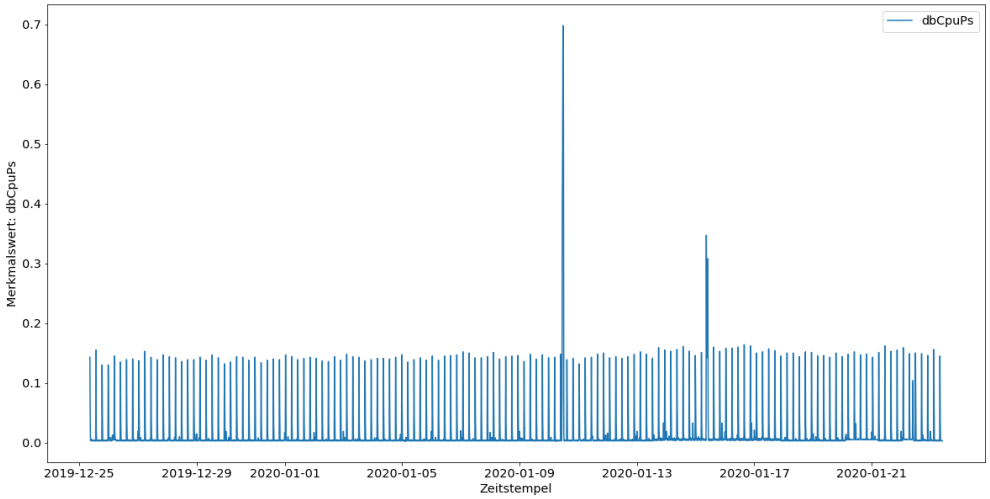

Abbildung A.5: Plot der Metrik *dbCpuPs* aus dem vierwöchigen Datensatz. Diese misst die CPU Nutzung durch die Datenbank.

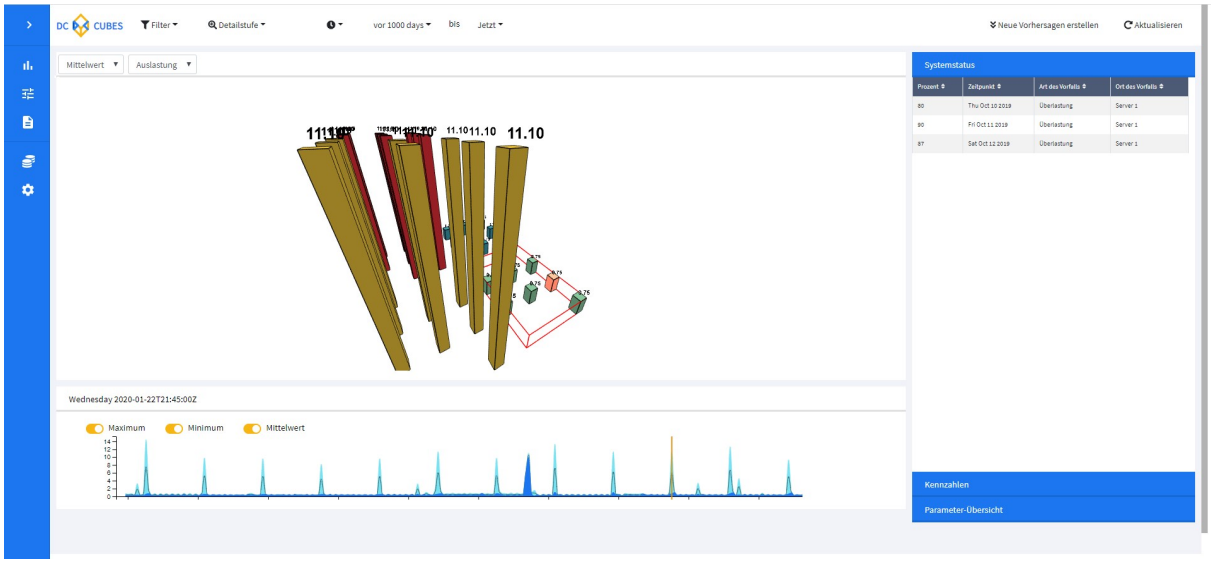

Abbildung A.6: Vollansicht der Startseite der dieser Arbeit zugrunde liegenden Webanwendung.

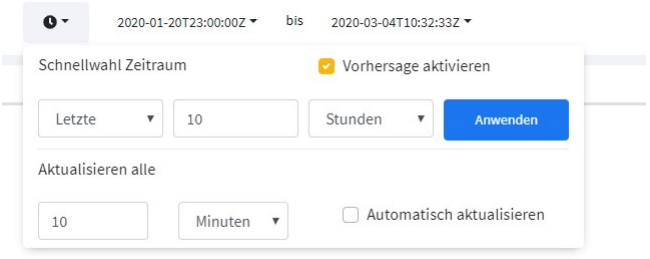

Abbildung A.7: Die Checkbox zur Aktivierung der Prognosenanzeige. Sie befindet sich in dem gleichen Menüpunkt wie die Auswahl der dargestellten Zeit.

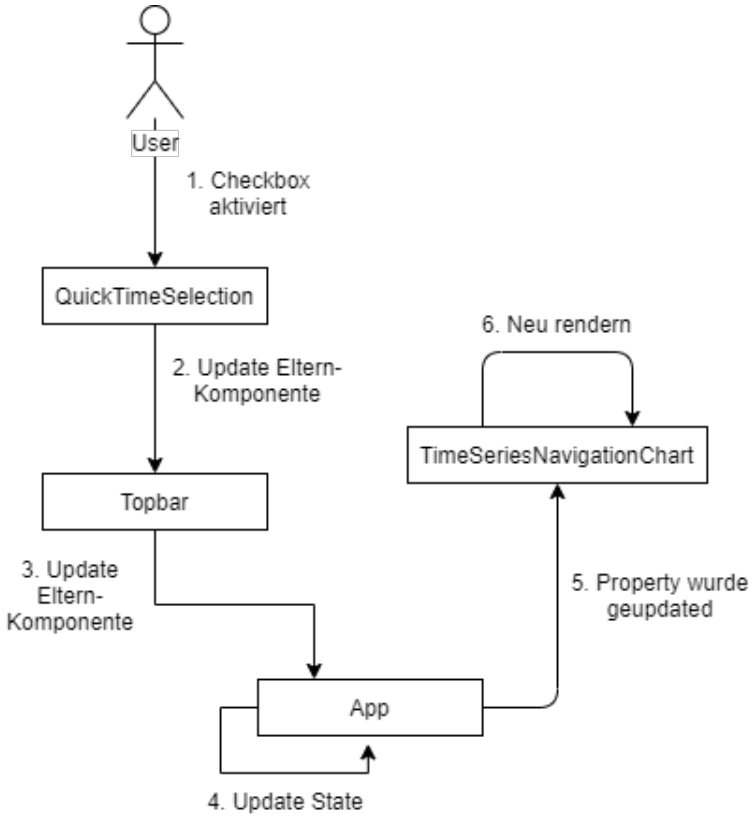

Abbildung A.8: Propagieren des Checkbox-Events über die Root-Komponente an die Anzeige-Komponente

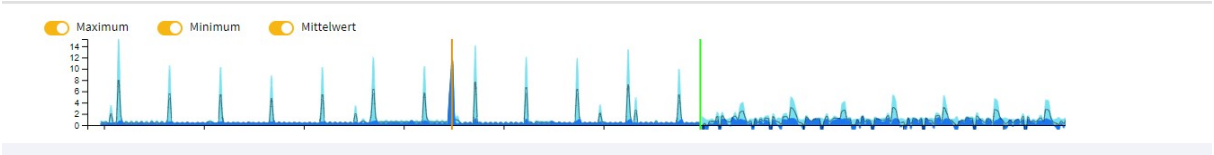

Abbildung A.9: 2d-Chart mit Prognose.

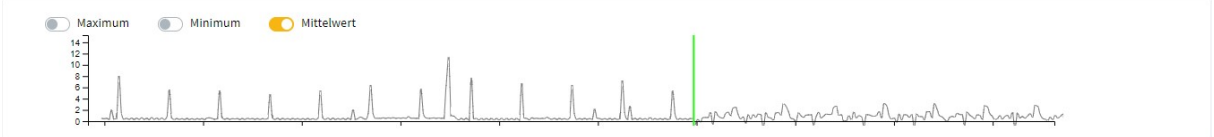

Abbildung A.10: Der 2d-Chart mit Prognose. Es wird nur der Durchschnitt der Prognosen aller Serverinstanzen angezeigt.

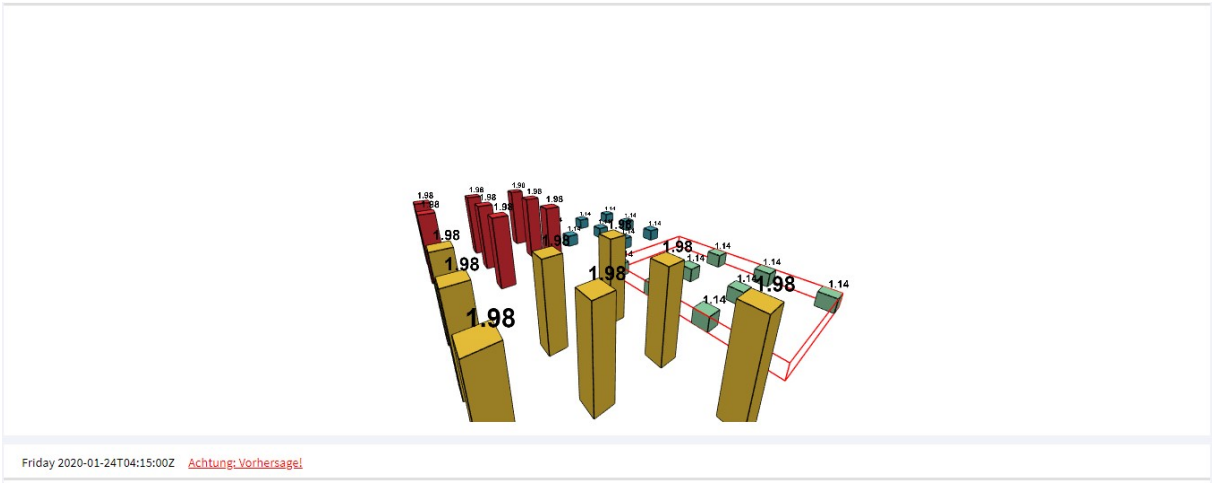

Abbildung A.11: Die Anzeige der Prognose im 3d-Chart. Durch den Warnhinweis wird dem Benutzer signalisiert, dass die Anzeige prognostizierte Werte enthält.

Anhang B

Tabellen

Name	**Beschreibung**	**Prognose-Relevanz**
TARGET_NAME	Das Verfahren, Datenbankinstanz	Ja
TARGET_TYPE	Art der verwendeten Datenbank (immer Oracle)	Nein
TARGET_GUID	Global eindeutige Identifikation der Datenbank	Nein
METRIC_NAME	Name der Metrik-Kategorie	Indirekt
METRIC_TYPE	Beschreibt den intern verwendeten Datentyp	Nein
METRIC_-COLUMN	Name der Spalte in der internen Tabelle, Featurekandidat	Ja
METRIC_GUID	Global eindeutige Identifikation der Metrik	Nein
METRIC_LABEL	Menschenlesbare Beschreibung des METRIC_NAME	Indirekt
COLUMN_LABEL	Menschenlesbare Beschreibung der METRIC_COLUMN	Indirekt
COLLECTION_-TIMESTAMP	Zeitpunkt, zu dem gemessen wurde	Ja
VALUE	Wert der Messung	Ja
KEY_VALUE[*]	Platzhalter-Spalten	Nein

Tabelle B.1: Bedeutung und Relevanz der Spalten im Datensatz.

Anhang C

Listings

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

n_steps = 10
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=n_steps))
model.add(Dropout(0.5)) # p_drop = 0.5
```

Listing C.1: Beispielhafte Dropout-Schicht in Keras

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

n_neurons = 50
model = Sequential()
model.add(Dense(n_neurons, activation='tanh', input_dim=n_input))
model.add(Dense(n_neurons, activation="tanh"))
model.add(Dense(n_neurons, activation="tanh"))
model.add(Dense(n_neurons, activation="tanh"))
model.add(Dense(pred_horizon))
model.compile(optimizer='adam', loss='mse')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience
    =50)
mc = ModelCheckpoint('mlp.h5', monitor='val_loss' , mode='min', verbose
    =1, save_best_only=True)

model.fit(x_train, y_train, validation_data = (x_val, y_val), epochs
    =400, callbacks=[es, mc])
```

Listing C.2: Implementierung des MLP mit Keras.

```
from keras.models import Sequential
from keras.layers import Conv1D
from keras.layers import LSTM
from keras.layers import Dense
from keras.callbacks import EarlyStopping
```

```
from keras.callbacks import ModelCheckpoint

model = Sequential()
model.add(Conv1D(filters=150, kernel_size=3, padding="same", activation='tanh', input_shape=(n_history, numberOfFeatures)))
model.add(Conv1D(filters=80, kernel_size=2, padding="same",activation='tanh'))
model.add(Conv1D(filters=60, kernel_size=2, padding="same",activation='tanh'))
model.add(LSTM(75, activation='tanh'))
model.add(Dense(pred_horizon))
model.compile(loss='mse', optimizer='adam')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
mc = ModelCheckpoint('cnn.h5', monitor='val_loss' , mode='min', verbose=1, save_best_only=True)

modelHistory = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=400, callbacks=[es, mc])
```

Listing C.3: Implementierung des CNN mit Keras.

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

model = Sequential()
model.add(LSTM(75, activation='tanh', return_sequences=True, input_shape=(n_history, numberOfFeatures)))
model.add(LSTM(75, activation="tanh", return_sequences=True))
model.add(LSTM(75, activation="tanh", return_sequences=True))
model.add(LSTM(75, activation="tanh", return_sequences=True))
model.add(LSTM(75, activation="tanh"))
model.add(Dense(pred_horizon))
model.compile(optimizer='adam', loss='mse')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
mc = ModelCheckpoint('switchedPca_lstm_multistep_multivariate.h5', monitor='val_loss' , mode='min', verbose=1, save_best_only=True)

model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=400, callbacks=[es, mc])
```

Listing C.4: Implementierung des LSTM-Netzes mit Keras.

```
from keras.models import Sequential
from keras.layers import GRU
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

model = Sequential()
```

```
model.add(GRU(50, activation='relu', return_sequences=True, input_shape
    =(n_history, numberOfFeatures)))
model.add(GRU(50, activation="relu", return_sequences=True))
model.add(GRU(50, activation="relu", return_sequences=True))
model.add(GRU(50, activation="relu", return_sequences=True))
model.add(GRU(50, activation="relu", return_sequences=True))
model.add(GRU(50, activation="relu"))
model.add(Dense(pred_horizon))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience
    =50)
mc = ModelCheckpoint('switchedPca_gru_multistep_multivariate.h5',
    monitor='val_loss' , mode='min', verbose=1, save_best_only=True)

model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=400,
    callbacks=[es, mc])
```

Listing C.5: Implementierung des GRU-Netzes mit Keras.

```
import pandas as pd
import pickle

ts = "COLLECTION_TIMESTAMP"
combinedDatasetForOneTargetName = "combined_pblm1.pkl"
with open(combinedDatasetForOneTargetName, "rb") as pklfile:
  df = pickle.load(pklfile)

def get10and15MinuteMetrics(df):
  tens = [];
  fifteens = [];
  allMetrics = df.METRIC_COLUMN.unique()
  for metric in allMetrics:
  samples = df[df.METRIC_COLUMN == metric].iloc[:5]
  if(len(samples) <= 1):
  print("not enough data:", metric)
  continue
  if(is10MinuteInterval(samples[ts])):
    tens.append(metric)
  else:
    fifteens.append(metric)
  return tens, fifteens


def is10MinuteInterval(timestamps):
  timedifferences = 0
  timestamps.reset_index(drop=True, inplace=True)
  limit = len(timestamps) - 1
  if(limit == 0):
    print("Only One timestamp:", timestamps)
    return True

  for i, curr in enumerate(timestamps):
    if i >= limit:
      break
    difference  = (timestamps[i+1] - curr).total_seconds()
```

```
    timedifferences += (difference / 60)
  return abs(10 - timedifferences/limit) < abs(15 - timedifferences/
    limit)

tens, fifteens = get10and15MinuteMetrics(df)


i = 0
def createUnifiedTimestamps(row):
  global df
  global i
  import math
  ts = row.COLLECTION_TIMESTAMP
  newMinute = getClosestMinute(ts.minute, row.is10MinuteInterval)
  newValue = math.nan #NaN initialisierung

  if(newMinute == 60): # ts.minute was e.g 59
    # newMinute will be set at end of method
    newMinute = 0;
    # add one Hour to the current timestamp
    deltaOneHour = pd.Timedelta(value=1, unit="hour")
    ts = ts + deltaOneHour

  if(row.is10MinuteInterval):
    if(newMinute == 0):
    # take it as it is
    newValue = row.VALUE
    elif(newMinute == 10):
      #do nothing, covered by min=20, for clarity no "else:"
      pass
    elif(newMinute == 20):
      # get mean between 10, 20 --> 15
      newValue = getMeanBetweenTwoTimestamps(df, ts, metric = row.
    METRIC_COLUMN, firstValue = row.VALUE)
      newMinute = 15
    elif(newMinute == 30):
      #take it as it is
      newValue = row.VALUE
    elif(newMinute == 40):
      #do nothing, covered by min=40
      pass
    elif(newMinute == 50):
      # get mean between 40,50 --> 45
      newValue = getMeanBetweenTwoTimestamps(df, ts, metric = row.
    METRIC_COLUMN, firstValue = row.VALUE)
      newMinute = 45
  row.VALUE = newValue #set it to nan to drop it later on

  ts = ts.replace(minute = newMinute, second = 0)
  i = i+1
  if(i % 10000 == 0):
    print("Erreicht: ", i)
  row.unifiedTimestamp = ts
  return ts

```

```
def getClosestMinute(actualMinute, is10Minute = False):
  #needs to include 60, but if its 60, we need to add 1 hour in method
   above
  possibleValues = [0, 15, 30, 45, 60]
  if is10Minute:
    possibleValues = [0, 10, 20, 30, 40, 50, 60]
  currentClosest = possibleValues[0]
  for val in possibleValues:
    if abs(actualMinute - val) < abs(actualMinute - currentClosest):
      currentClosest = val
  return currentClosest

newcol = "is10MinuteInterval"
df[newcol] = df.apply(lambda row: row.METRIC_COLUMN in tens, axis=1)
df["unifiedTimestamp"] = 0 # initialize it so it does exist for
   searching a matchingRow in getMeanBetweenTwoTimestamps
df["unifiedTimestamp"] = df.apply(createUnifiedTimestamps, axis=1)
```

Listing C.6: Vereinheitlichen der Zeitstempel

```
df["relevant"] = True
def checkForRelevance(row):
  ts = row.unifiedTimestamp
  min = ts.minute
  relevant = False
  if (min == 0 or min == 15 or min == 30 or min == 45):
    relevant = True
  row.relevant = relevant
  return relevant
df["relevant"] = df.apply(checkForRelevance, axis=1)
df = df.drop(df[df.relevant == False].index)
```

Listing C.7: Entfernen überflüssiger Zeilen

```
from math import nan
allMetrics = df.METRIC_COLUMN.unique()
for metric in allMetrics:
  df[metric] = nan
result = pd.DataFrame(columns = df.columns)
result["timestamp"] = df.unifiedTimestamp.unique()
def setValueToMetriccolumn(row):
  global result
  metric = row.METRIC_COLUMN
  ts = row.unifiedTimestamp
  (result.loc[result.timestamp == ts, metric]) = row.VALUE
  return row
df.apply(setValueToMetriccolumn, axis=1)
result = result.set_index("timestamp")
result = result.dropna(axis=1, how="all")
irrelevantCols = ['AUD_FILE_SIZE',
'BIN_FILE_SIZE',
'CRITICAL_INCIDENTS',
'FILE_SIZE',
'InstrumentationPresent',
'NeedToInstrument',
'OCMConfigured',
```

```
'SEVERITY_INDEX',
'Session Count',
'WARNING_INCIDENTS',
'XML_FILE_SIZE',
'alertLogName',
'archiveHungErrors',
'blockCorruptErrors',
'dumpAvail',
'dumpDir',
'dumpTotal',
'dumpUsed',
'dumpUsedPercent',
'genericErrStack',
'genericErrors',
'if_type',
'machine',
'mediaFailureErrors',
'scn_intrinsic_growth_rate',
'sessTerminateErrors',
'timeLine',
'traceFileName',
'username',
'waitClassName']
result.drop(columns= irrelevantCols, inplace=True)
```

Listing C.8: Transformieren der Features von Zeilen zu Spalten

Anhang D

Inhaltsverzeichnis der CD

Bachelorarbeit.pdf Die Bachelorarbeit als PDF-Dokument.

Latex Die Latex-Quellen.

Modelle/final_cnn_pblm1.h5 Das CNN-Modell der ersten Serverinstanz.

Modelle/final_cnn_pblm2.h5 Das CNN-Modell der zweiten Serverinstanz.

Python-Skripte/ACF.ipynb ACF, PACF und Komponentenzerlegung.

Python-Skripte/concatWeeks.ipynb Zusammenführen der einzelnen Wochen.

Python-Skripte/createChunks.ipynb Blockweises Einlesen des Datensatzes.

Python-Skripte/GeneratePickleForTargetName.ipynb Filtern nach einer Serverinstanz.

Python-Skripte/GridSearch.ipynb Rastersuche.

Python-Skripte/MLSkript.py

Python-Skripte/PlotFeatures.ipynb Plotten verschiedener Metriken.

Python-Skripte/PushForecastHistoricData.ipynb Speichern der vierwöchigen Daten in Solr. Erzeugen und Speichern der Prognose.

Python-Skripte/TimestampAnpassung.ipynb Vereinheitlichung der Zeitstempel. Transformation der Metriken von Zeilen zu Spalten.

Python-Skripte/train_plot.ipynb Vergleich der Modelle. Erzeugen der finalen Modelle.

Webanwendung/App.tsx Root-Komponente der Webanwendung.

Webanwendung/backend.ts Der Node.js Server.

Webanwendung/CubesVisualization.tsx Komponente der 3d-Ansicht.

Webanwendung/QuickTimeSelection.tsx Beinhaltet die Checkbox zur Anzeige der Prognosen.

Webanwendung/SolrDataService.ts Regelt die HTTP-Requests an Solr.

Webanwendung/TimeseriesNavigationChart.tsx Komponente der 2d-Ansicht.

Webanwendung/Topbar.tsx Obere Menüleiste, beinhaltet `QuickTimeSelection`.