Bachelorarbeit

Studiengang Wirtschaftsinformatik

Machine Learning für Zeitreihen – Modell zur Vorhersage von Serverauslastungen

Ivan Levarda

Aufgabensteller/Prüfer	Prof. Dr. Stefan Wind
Arbeit vorgelegt am	23.03.2020
Durchgeführt bei	Capgemini Deutschland GmbH
	Bahnhofstraße 11C
	90402 Nürnberg
Betreuer	Dr. Eldar Sultanow
	eldar.sultanow@capgemini.com
Anschrift des Verfassers	Kemptener Straße 1
	87749 Hawangen

Abstract

Im Rahmen dieser Arbeit wurden Methoden zur Vorhersage von Zeitreihen untersucht. Für ein konzeptionelles 3D Monitoring- und Visualisierungs-Tool für die Rechenzentren einer deutschen Behörde sollten Machine Learning Modelle erforscht und entwickelt werden, die eine Vorhersage von Serverauslastungen ermöglichen. Ein durchgeführtes Vergleichsexperiment, in dem mehrere Machine Learning Modelle getestet worden sind, bestätigte die Möglichkeit einer akkuraten Vorhersage und ergab, dass das in der Arbeit vorgestellte und implementierte Convolutional Neural Network (CNN) die besten Ergebnisse für diesen konkreten Anwendungsfall liefert.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ARIMA ..... *Autoregressive Integrated Moving Average*
CNN ..... *Convolutional Neural Network*
KNN ..... *Künstliches neuronales Netz*
LSTM ..... *Long Short-Term Memory*
MLP ..... *Multilayer Perceptron*
MSE ..... *Mean Squared Error*
R^2 ..... *Bestimmtheitsmaß*
RMSE ..... *Root Mean Squared Error*
RNN ..... *Recurrent Neural Network*

Abbildungsverzeichnis

Formelverzeichnis

Tabellenverzeichnis

1 Einleitung

Mit immer größer werdenden Ansammlungen von Daten und der Verbreitung der Digitalisierung, wird es stets wichtiger, Werkzeuge und Methoden zu verwenden, die eine nützliche und effiziente Extraktion von Informationen ermöglichen.[1] Mit der konstant steigenden Rechenleistung sind neue Möglichkeiten weitläufig einsetzbar, mithilfe dessen aus großen Datenmengen Einblicke und Erkenntnisse gewonnen werden können. Eine Möglichkeit diese Erkenntnisse zu gewinnen ist durch Machine Learning. Diese Arbeit beschäftigt sich mit Machine Learning und der Vorhersage von Serverauslastungen für ein hochverfügbares Rechenzentrum.

1.1 Problemstellung

Im Rahmen einer Kooperation zwischen Capgemini und der Bundesagentur für Arbeit wurde die Forschung an einem neuartigen 3D Visualisierungs- und Monitoring-Tool für Rechenzentren gestartet. Die Bundesagentur für Arbeit betreibt drei hochverfügbare Rechenzentren mit über 10 000 Servern. Für diese IT-Landschaft, auf der einige für die Bevölkerung existenzkritische Anwendungen betrieben werden, wird ein umfangreiches und effizientes Monitoring benötigt. Die erwähnte Monitoring-Anwendung ist in Abbildung 1 dargestellt. Dieses Tool soll den Benutzer dabei unterstützen, in den Serverauslastungsdaten Anomalien und mögliche Ursachen für Fehler, die im Rechenzentrum aufkommen, zu finden und zu klassifizieren.[2]

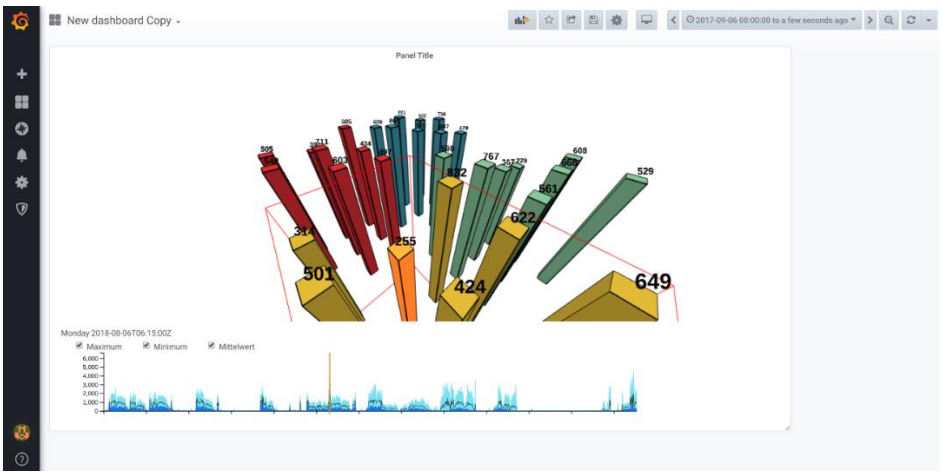


Abbildung 1: Prototyp des 3D Visualisierungs- und Monitoring Tools
Quelle: https://github.com/Sultanow/dc_cubes, 22.03.2020

[1] vgl. Fayyad et al., From data mining to knowledge discovery in databases, 1996, S. 37.

[2] vgl. Chircu et al., Visualization and Machine Learning for Data Center Management, 2019, S. 24.

Um die Prozesse, die für die Hochverfügbarkeit dieser Rechenzentren zuständig sind, zu unterstützen und effizienter zu gestalten, ist die Vorhersage von Serverauslastungszeitreihen bedeutend. Wenn eine zuverlässige Vorhersage der zukünftigen Entwicklung der Serverauslastungen möglich ist, können Lastspitzen oder Ausfallzeiten frühzeitig entdeckt und geeignete Gegenmaßnahmen eingeleitet werden. Dadurch können gegebenenfalls Kosten und Imageschädigungen erspart werden. Demzufolge sollen für die vorgestellte Anwendung Machine Learning Modelle untersucht, implementiert und getestet werden, die solch eine zuverlässige Prognose ermöglichen. Die dabei verwendeten Daten stellt die Bundesagentur für Arbeit zu Verfügung.

1.2 Zielsetzung

Mit dem Hintergrund der vorgestellten Problemstellung soll diese Arbeit aufzeigen, welche Machine Learning Methoden für die Vorhersage von Zeitreihen zur Verfügung stehen. Es wird erforscht, ob Machine Learning Modelle für den konkreten Fall, der Vorhersage von Serverauslastungen, eingesetzt werden können. Dazu wird überprüft, mit welcher Zuverlässigkeit und Genauigkeit für einen bestimmten Zeithorizont eine Vorhersage getätigt werden kann. Letztendlich ist das übergeordnete Ziel, ein Machine Learning Modell für diesen Anwendungsfall zu entwickeln und zu empfehlen.

1.3 Vorgehensweise

Die Vorgehensweise dieser Arbeit wird in Abbildung 2 veranschaulicht.

Abbildung 2: Aufbau der Arbeit
Quelle: eigene Darstellung

Die theoretischen Grundlagen und der State of the Art bei der Zeitreihenvorhersage mittels Machine Learning werden mithilfe von Literaturforschung ausgearbeitet. Dies erfolgt in Kapitel 2 und die dabei verwendete Methodik wird im nächsten Abschnitt vorgestellt. Darauffolgend wird eine Analyse und Konzeption in Kapitel 3 durchgeführt und das Vergleichsexperiment als Methodik vorgestellt, welche zur Beantwortung der Zielsetzung im vorherigen Abschnitt Verwendung finden wird. Um das Experiment durchführen zu können, werden die in der Literaturforschung recherchierten Modelle in Kapitel 4 implementiert. In Kapitel 5 werden die Ergebnisse des durchgeführten

Experiments vorgestellt und evaluiert, um die Zielsetzung zu beantworten und ein geeignetes Modell zu empfehlen.

1.4 Literaturanalyse

Für die Auffassung der theoretischen Grundlagen im zweiten Kapitel, wird die von Brocke et al. vorgeschlagene Methodik verwendet. Folgende Abbildung veranschaulicht das Vorgehen:

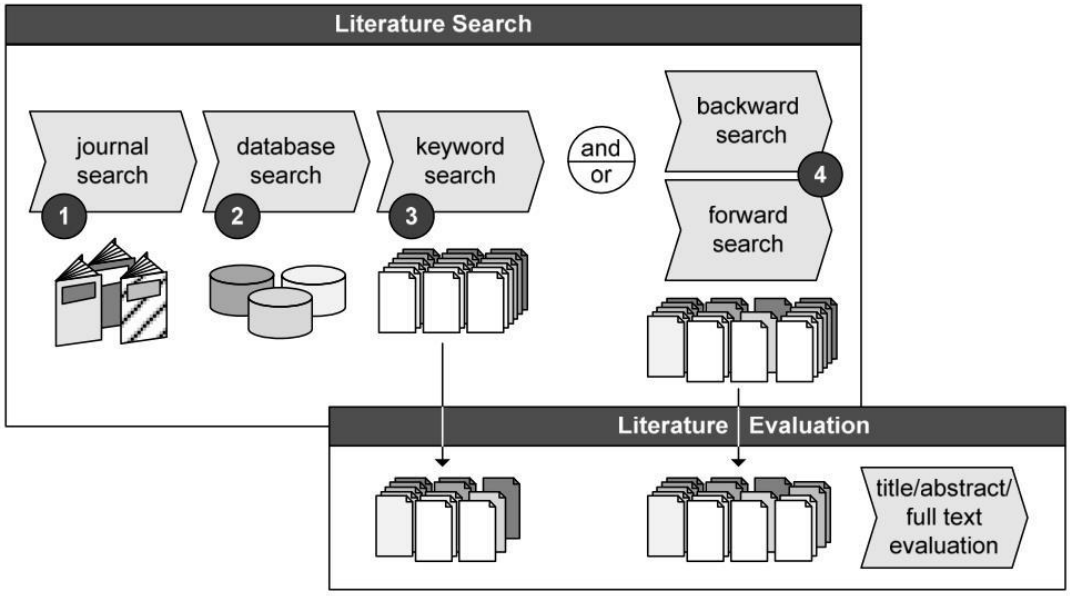


Abbildung 3: Prozess der Literaturanalyse
Quelle: (Vom Brocke et al., 2009, o. S.)

Die Literaturanalyse besteht nach Abbildung 3 aus der Literatursuche und der Literaturauswertung. Bei der Literatursuche werden zunächst qualitative Journals und Datenbanken identifiziert und mit passenden Zeichenketten und Filtern systematisch durchforstet. Auf den dadurch gefundenen Dokumenten werden optional Rückwärts- oder Vorwärtssuchen angewendet, um weitere relevanten Quellen zu finden.[3]

Die Rückwärtssuche identifiziert dabei relevante Literatur aus den zitierten Quellen des im Fokus liegenden Dokuments. Die Vorwärtssuche findet Literatur, die das im Fokus

[3] vgl. Vom Brocke et al., RECONSTRUCTING THE GIANT: ON THE IMPORTANCE OF RIGOUR IN DOCUMENTING THE LITERATURE SEARCH, 2009, o.S.

liegende Dokument zitiert haben.[4] In der Literaturauswertung nach Abbildung 3 wird für diese Arbeit die gefundene Literatur auf ihre Relevanz mittels des Abstracts überprüft und mithilfe der qualitativen Inhaltsanalyse nach Mayring ausführlicher untersucht.[5]

Für die Literatursuche werden folgende Zeichenketten verwendet:

- Time series AND (analysis OR forecasting OR prediction)
- (machine learning OR deep learning) AND time series AND OR (forecasting OR prediction)

Die Auswahl der Journale orientiert sich auf das Wirtschaftsinformatik Teilrating des VHB-JOURQUAL 3. Folgende Datenbanken dienten für die Suche weiterer wissenschaftlicher Dokumente: AISel, Springer, Science Direct, ResearchGate, IEEE und Google Scholar.

[4] vgl. Webster und Watson, Analyzing the past to prepare for the future: Writing a literature review, 2002, xvi.

[5] vgl. Mayring und Fenzl, Qualitative Inhaltsanalyse, 2019.

2 Theoretische Grundlagen

In diesem Kapitel werden die notwendigen theoretischen Grundlagen und der State of the Art für die Vorhersage von Zeitreihen mittels Machine Learning aufgezeigt. Diese dienen als Grundlage für die Analyse und Konzeption und die anschließende Implementierung.

2.1 Zeitreihenanalyse

Die Zeitreihenanalyse beschäftig sich mit der Untersuchung, Modellierung und Prognose von zeitlich geordneten Daten. Zeitlich geordnete Daten beinhalten in der Regel einen Bezug zu dem historischen Verlauf ihrer Aufzeichnung. So kann beispielsweise die heutige Entwicklung eines Umsatzes abhängig von der gestrigen Entwicklung geschätzt werden.
Die Zeitreihenanalyse untersucht diese Abhängigkeiten und weitere statistische Eigenschaften, unter anderem Stationarität, Trend, sowie Saisonalität und erstellt Modelle zur Abbildung und Prognose.[6]

2.1.1 Univariate und multivariate Zeitreihen

Hauptgegenstand dieser Arbeit sind Zeitreihen. Eine Zeitreihe ist eine chronologische Sequenz von Messwerten aufgenommen zu einem bestimmten Zeitpunkt. [7]

Eine Zeitreihe kann kontinuierlich oder diskret sein. Bei einer kontinuierlichen Zeitreihe werden die Daten ununterbrochen im Verlauf der Zeit aufgenommen. Wohingegen bei diskreten Zeitreihen die Daten zu bestimmten Zeitpunkten, meistens in einem konstanten Intervall, gemessen werden.

Zeitreihen sind in der Welt allgegenwärtig. So ist zum Beispiel die Entwicklung des Bruttoinlandsprodukts über die vergangenen 3 Jahre oder die Temperaturaufzeichnung der letzten 24 Stunden eine Zeitreihe.

Dabei gilt es zwischen **univariaten** und **multivariaten** Zeitreihen zu unterscheiden.

[6] vgl. Sibbertsen, Zeitreihenanalyse, 2011, S. 165.

[7] vgl. Montgomery et al., Introduction to Time Series Analysis and Forecasting, 2015, S. 2.

Abbildung 4 zeigt ein Beispiel für eine univariate Zeitreihe, bei der die CPU Auslastung in Prozent eines Servers im Intervall von zehn Minuten gemessen und aufgezeichnet worden ist. Dabei wird zu jedem Zeitpunkt x (Timestamp) jeweils ein y Wert (Host CPU Utilization %) gemessen.

	Timestamp	Host CPU Utilization (%)
0	2019-09-11 14:00:00	13.068484
1	2019-09-11 14:10:00	25.589807
2	2019-09-11 14:20:00	12.095209
3	2019-09-11 14:30:00	21.329274
4	2019-09-11 14:40:00	46.546902
...	...	...

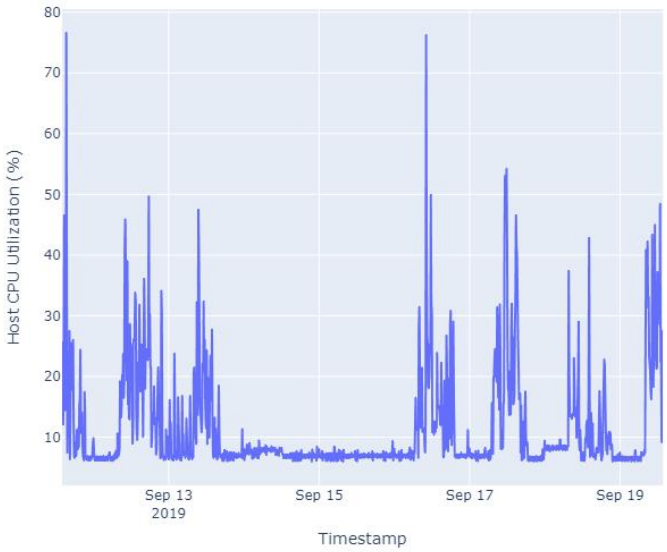
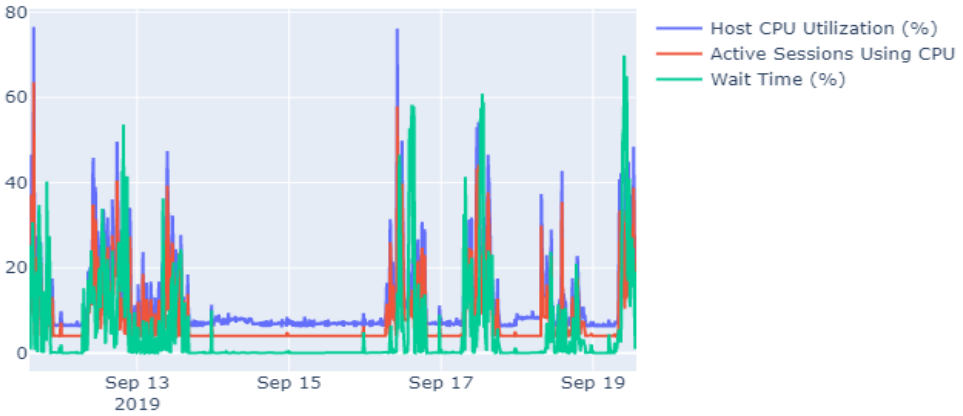


Abbildung 4: Univariate Zeitreihe als Graph und Tabelle
Quelle: eigene Darstellung

Im Unterschied dazu zeichnet eine multivariate Zeitreihe, wie in Abbildung 5 zu sehen ist, mehrere y Werte (Host CPU Utilization %, Active Sessions Using Cpu, Wait Time %) zu dem gleichen Zeitpunkt x (Timestamp) in einem bestimmten Intervall (10 min.) auf.



	Timestamp	Host CPU Utilization (%)	Active Sessions Using CPU	Wait Time (%)
0	2019-09-11 14:00:00	13.068484	10.008	3.044403
1	2019-09-11 14:10:00	25.589807	19.762	4.912818
2	2019-09-11 14:20:00	12.095209	8.736	0.886582
3	2019-09-11 14:30:00	21.329274	15.853	25.318306
4	2019-09-11 14:40:00	46.546902	37.193	13.480704
...	...	...	...	...

Abbildung 5: Multivariate Zeitreihe als Graph und Tabelle
Quelle: eigene Darstellung

Die Teile einer multivariaten Zeitreihe können einzeln auch als unabhängige univariate Zeitreihen analysiert werden. Werden diese voneinander getrennten Zeitreihen jeweils einzeln untersucht, können Informationen aus den Beziehungen zueinander nicht in eine Vorhersage hineinfließen. Solche Korrelationen können jedoch hilfreich sein, wenn es darum geht, Zukunftswerte vorherzusagen.[8]

2.1.2 Komponenten einer Zeitreihe

Weitere Komponenten mit denen sich die Zeitreihenanalyse befasst sind Saisonalität und Trend.

Saisonalität ist definiert als ein sich regelmäßig (jährlich, monatlich, täglich etc.) wiederholendes Verhalten in einer Zeitreihe.[9]
Bei Daten von Geschäften, oder wie in diesem Fall von einer Behörde, ist es von größerer Bedeutung die Saisonalität für eine Vorhersage richtig zu erfassen, da ihr Verlauf oft abhängig von der Art des Tages ist (Wochenende, Feiertag).[10]
Zum Beispiel haben die Niederlassungen der Bundesagentur für Arbeit am Wochenende geschlossen und daher ist, wie in der obigen Abbildung 4 zu sehen, die gemessene CPU-Auslastung an zwei Tagen deutlich geringer als üblich. Auch ist der Tag- und Nacht-Zyklus deutlich erkennbar, dabei mit höheren Auslastungen tagsüber als bei Nacht.
Die generelle Tendenz einer Zeitreihe über einen Zeitraum zu steigen, sinken oder stagnieren wird als Trend bezeichnet. Saisonalität und Trend einer Zeitreihe können oft bereits mithilfe eines Liniendiagramms einfach entdeckt werden.[11] Wobei der Trend durch eine Saisonbereinigung besser sichtbar gemacht werden kann.[12]

Beim Beobachten des Trends und der Saisonalität kann von folgendem Modell zur Dekomposition gebrauch genommen werden, um die beiden Komponenten besser zur Geltung zu bringen:

$$Xt = mt + st + Yt$$

Formel 1: Modell zur Dekomposition in Trend und Saisonalität

[8] vgl. Brockwell und Davis, Introduction to Time Series and Forecasting, 2016, S. 228.

[9] vgl. Montgomery et al., Introduction to Time Series Analysis and Forecasting, 2015, S. 15.

[10] vgl. Hupez et al., SARMA Time Series for Microscopic Electrical Load Modeling, 2017., S. 136

[11] vgl. Montgomery et al., Introduction to Time Series Analysis and Forecasting, 2015, S. 27.

[12] vgl. Vogel, Prognose von Zeitreihen, 2015, S. 53.

Dabei repräsentiert mt den Trend, st die saisonale Komponente und Yt weißes Rauschen, welches zu keinen der beiden Komponenten zugewiesen werden kann.[13]

Die Dekomposition der in Abbildung 4 vorgestellten Zeitreihe kann in Abbildung 6 betrachtet werden:

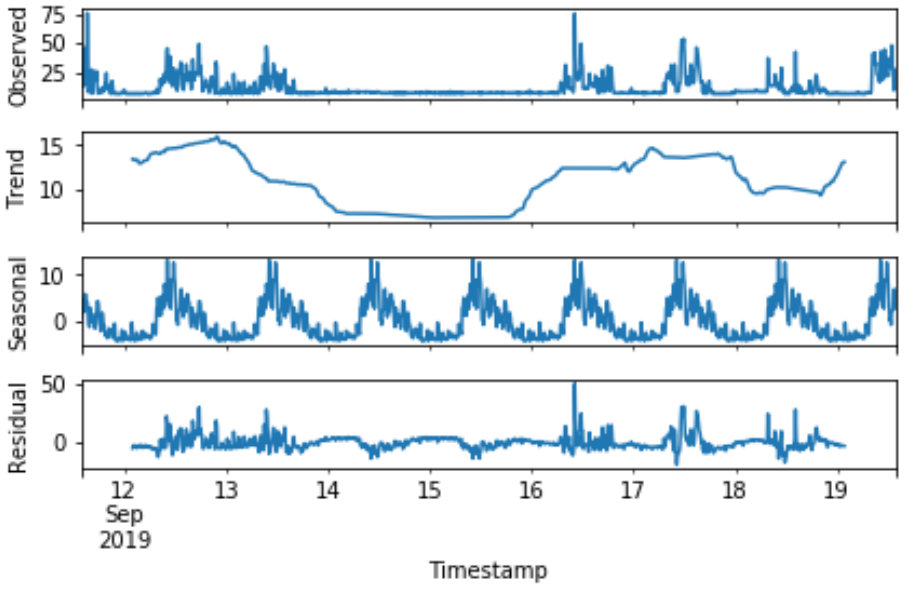


Abbildung 6: Dekomposition einer Zeitreihe in Trend und Saisonalität
Quelle: eigene Darstellung

Aus der Abbildung 6 entnommen ist *Observed* die tatsächlich beobachtete Zeitreihe, *Residual* das zuvor genannte weiße Rauschen, *Seasonal* die Saisonalität.

2.1.3 Stationarität

Eine Zeitreihe ist stationär, wenn sie einen konstanten Durchschnitt, Varianz und Kovarianz über ihren gesamten Verlauf beinhaltet.[14]

Stationarität trifft dabei in zwei Stärken auf. Eine Zeitreihe ist streng stationär, wenn ihre gemeinsame Wahrscheinlichkeitsverteilung für eine beliebige Beobachtungsmenge $\{t1, t2, \dots, tm\}$ exakt gleich mit der gemeinsamen Verteilung für $\{t1 + k, t2 + k, \dots, tm + k\}$ ist. Dabei ist t die Zeit und k die Verschiebung entlang der Zeitachse.
Des Weiteren ist eine Zeitreihe bereits dann schwach stationär, wenn die erste und zweite Beobachtung unabhängig von der Verschiebung entlang der Zeitachse ist.

[13] vgl. Brockwell und Davis, Introduction to Time Series and Forecasting, 2016, S. 20.

[14] vgl. Heinrich und Fleißner, Deep Intelligent Systems for Time Series Prediction, 2018, S. 2.

Eine streng stationäre Zeitreihe ist ebenso schwach stationär. Daher gilt für alle t und k:[15]

$$(i)\ \ E(x) = E(x2) = \cdots = E(xt) = \mu$$
$$(ii)\ \ Cov(x1, x1 + k) = Cov(x2, x2 + k) = \cdots = Cov(xt, xt + k)$$

Formel 2: Erwartungswert und Kovarianz einer stationären Zeitreihe

Die Untersuchung der Stationarität ist deshalb bedeutend, da es mehrere gut erforschte Methoden für die Vorhersage von stationären Zeitreihen gibt, wie zum Beispiel das ARMA (Autoregressive Moving Average) Modell. Wirtschaftszeitreihen wie Aktienkurse oder speziell in diesem Fall, Messungen von Serverauslastungen, sind selten stationär und bedürfen daher besonderem Vorgehen bei der Analyse und der Vorhersage.

2.1.4 Vorhersage

Bei der Vorhersage von Zeitreihen wird davon ausgegangen, dass im Zeitverlauf der beobachteten Zeitreihe unveränderliche Gesetzmäßigkeiten existieren, die es ermöglichen, von der Vergangenheit auf die Zukunft zu schließen. Ziel der Zeitreihenanalyse in Hinblick auf die Vorhersage ist es, ein geeignetes Modell zu finden, mit der sich die Zeitreihe abbilden lässt und mit dem zuverlässige Prognosen für die Zukunft getroffen werden können.[16] Dabei erfasst das Modell Muster in den Daten und setzt die statistische Beziehung zwischen den Vergangenheitswerten um. Das Model wird dann genutzt, um die erfassten Muster und Regelmäßigkeiten auf die Zukunft zu projizieren und somit eine Vorhersage zu treffen.[17]

Bei der Vorhersage ist es wichtig, den Vorhersagehorizont festzulegen. Ein Modell kann darauf abgestimmt sein, nur einen oder mehrere Intervallschritte in die Zukunft vorherzusagen. Daher wird zwischen **einschrittigen** $yt + 1$ (single-step) und **mehrschrittigen** $yt + n$ (multi-step) Vorhersagemethoden unterschieden.

Aufgrund von Komplikationen, wie der Ansammlung von Fehlern, geringerer Vorhersagegenauigkeit und größerer Unsicherheit, ist es generell schwieriger, eine mehrschrittige Vorhersage durchzuführen als eine einschrittige.[18]

---

[15] vgl. Koller, Prognose makroökonomischer Zeitreihen, 2014, S. 9.

[16] vgl. Koller, Prognose makroökonomischer Zeitreihen, 2014, S. 8.

[17] vgl. Montgomery et al., Introduction to Time Series Analysis and Forecasting, 2015, S. 5.

[18] vgl. Sorjamaa et al., Methodology for long-term prediction of time series, 2007, S. 2862.

Hauptsächlich gibt es drei Strategien für die mehrschrittige Vorhersage. Die rekursive, die direkte und die mehrfache Ausgabe (multiple output) Strategie.[19]

Die rekursive Strategie trainiert nur ein einschrittiges Modell und verwendet dessen Ausgabe rekursiv als Input für die Vorhersage des nächsten Schrittes und somit die Erstellung einer mehrschrittigen Vorhersage.[20]

Bei der direkten Strategie werden mehrere voneinander unabhängige einschrittige Modelle trainiert. Nach dem mit dem ersten Modell eine Vorhersage getätigt wurde, wird diese zusammen mit den historischen Daten für das Training des nächsten Modells genommen, um den $yt + 2$ Schritt zu vorhersagen usw. bis der Vorhersagehorizont gedeckt wird. Die Anzahl der Modelle die trainiert werden müssen, ist somit gleich dem Vorhersagehorizont. Eine mehrschrittige Vorhersage entsteht dann durch die chronologische Verbindung der Ausgaben der jeweiligen einschrittigen voneinander unabhängigen Modelle.[21]

Im Gegensatz zu den oben genannten Strategien, die je Modell immer einen Zukunftsschritt ausgeben, gibt das trainierte Modell bei der multiple output Strategie auf einmal eine mehrfache Ausgabe aus. Daher wird nur ein mehrschrittiges Modell erstellt, welches in der Lage ist mit dem Input von *n* historischen Schritten *x* zukünftige Schritte auf einmal vorherzusagen.[22]

Für die Vorhersage von Zeitreihen gibt es zahlreiche Modelle. Diese werden unterschieden in statistische Methoden und Deep Learning Methoden.
Im nächsten Abschnitt wird allgemein auf die Grundlagen von Machine Learning eingegangen. Danach werden statistische Machine Learning Methoden für Zeitreihen aufgezeigt und erklärt. Diese haben weiterhin eine feste Bedeutung in der Zeitreihenanalyse.
Im darauffolgenden Abschnitt wird primär der Fokus auf das Deep Learning und diesbezügliche Methoden zur Zeitreihenvorhersage gelegt.

[19] vgl. Bontempi et al., Machine Learning Strategies for Time Series Forecasting, 2013, S. 70.

[20] vgl. Sorjamaa et al., Methodology for long-term prediction of time series, 2007, S. 2862.

[21] vgl. Bontempi et al., Machine Learning Strategies for Time Series Forecasting, 2013, S. 71.

[22] vgl. Bontempi, Long term time series prediction with multi-input multi-output local learning, 2008, o.S.

2.2 Machine Learning

In diesem Abschnitt folgt eine allgemeine Definition und Erklärung von Machine Learning.
Gegen Ende dieses Abschnitts wird Bezug auf den Einsatz von Machine Learning bei der Vorhersage von Zeitreihen genommen und die Methoden und Modelle im nächsten Kapitel genauer durchleuchtet und vorgestellt.

Erste Machine Learning Algorithmen erschienen bereits 1970, aber der Trend um künstliche Intelligenz und Machine Learning entstand erst vor kurzer Zeit.
Durch die exponentiell gestiegene Rechenleistung und die große Verfügbarkeit und Sammlung von Daten ist es nun möglich, Machine Learning auf umso komplexere Probleme und weitgefächerte Bereiche anzuwenden.[23]

Wie in Abbildung 7 verdeutlicht wird, ist Machine Learning ein Teilgebiet der künstlichen Intelligenz. Aufgabe der künstlichen Intelligenz ist es, intellektuelle Aufgaben zu automatisieren, die normalerweise von Menschen durchgeführt werden. Es umfasst maschinelles und tiefes Lernen, aber schließt auch Methoden ein, die sich nicht mit „Lernen“ befassen. Ein Beispiel dafür ist die symbolische künstliche Intelligenz, mit der Grundannahme, dass menschliche Intelligenz durch einen entsprechend hohen handcodierten Satz an Regeln zur Manipulation von Wissen nachgebildet werden kann. Dieser Ansatz erwies sich als geeignet bei genau definierten logischen Problemen wie dem Schachspiel.

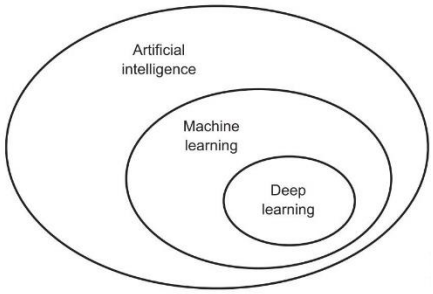


Abbildung 7: Teilmengen der Künstlichen Intelligenz
Quelle: Chollet, 2018, S. 4

Jedoch bei komplexeren Problemen, die nicht genau mithilfe von logischen Regeln definiert werden können, wie die Spracherkennung oder die Bildklassifizierung, erweist sich der Ansatz mit symbolischer KI als ungeeignet. Für solche komplexeren Probleme bietet sich das Machine und Deep Learning an.[24]

Beim klassischen Programmieren mit Hinblick auf den Ansatz der symbolischen KI, definiert der Mensch die Regeln (das Programm), mithilfe dessen eingegebene Daten zu einem Ergebnis verarbeitet werden. Ein Machine Learning System wird nicht explizit

[23] vgl. Louridas und Ebert, Machine Learning, 2016, S. 110.

[24] vgl. Chollet, Deep learning with Python, 2018, S. 4.

programmiert, sondern trainiert. Das System bekommt zu einer Aufgabe relevante Daten und zu diesen die passenden Ergebnisse als Input.

Es findet statistische Strukturen und Beziehungen aus diesen Daten und bildet selbstständig einen Satz von Regeln für die automatisierte Lösung dieser Aufgabe (siehe Abbildung 8). Daher ist die grundlegende Idee hinter Machine Learning, dass ein Computer durch Beispielsdaten zur Lösung einer Aufgabe trainiert wird und dann das Gelernte an Daten ausführt, die es noch nicht begegnet oder verarbeitet hat.[25]

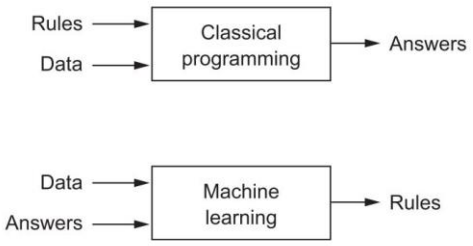


Abbildung 8: Unterscheidung klassische Programmierung zu ML
Quelle: Chollet, 2018, S. 4

2.2.1 Supervised Learning

Das überwachte Lernen (Supervised Learning) ist ein Konzept, welches einen Satz von Trainings – oder Beispieldaten erfordert. Dieser Trainingsdatensatz wird vor dem lernen durch den Trainer oder Überwacher erstellt.[26]

Beim überwachten Lernen beinhaltet der Trainingssatz die Daten und zu diesen Daten den richtigen Output als Lösung der gestellten Aufgabe. Die Daten sind demnach gekennzeichnet. Das System oder Modell lernt die Abbildung eines Inputs auf einen bestimmten Output. Ein Beispiel welches dies verdeutlich, ist etwa folgendes: ein Schüler bekommt mehrere Aufgaben und die dazugehörigen Lösungen. Folglich soll der Schüler mithilfe dessen die Lösung für zukünftige ähnliche ihm noch nicht bekannte Aufgaben selbständig herausfinden.[27]

Das überwachte Lernen beinhaltet **klassifizierende** Algorithmen. Diese Algorithmen bekommen als Input einen Datensatz und die Klasse jedes einzelnen Datenfeldes.

Ziel ist es, einem Computer beizubringen, neue Daten einer bestimmten Klasse zuzuordnen. Dies ist bei der Klassifikation von Bildern der Fall.

[25] vgl. Chollet, Deep learning with Python, 2018, S. 5.

[26] vgl. Konar und Bhattacharya, Time-Series Prediction and Applications, 2017, S. 5.

[27] vgl. Louridas und Ebert, Machine Learning, 2016, S. 110–113.

Des Weiteren zählen Regressionsalgorithmen zum überwachten Lernen. Ziel einer **Regression** ist es, auf Basis von bestimmtem Input dazugehörige Werte vorherzusagen.[28] Daraus lässt sich schon erahnen, dass die Vorhersage von Zeitreihen als ein Regressionsproblem des überwachten Lernens erfasst werden kann. Wie ein Supervised Learning Problem in Bezug auf Zeitreihen formuliert werden kann, wird in Kapitel 3.3 gezeigt.

2.2.2 Unsupervised Learning

Die Methoden des überwachten Lernens benötigen einen Trainingsdatensatz welcher schon die Ergebnisse, die es zu vorhersagen versucht, gelabelt beinhaltet. Jedoch ist dieser Vorteil, dass die zugehörigen Labels in den Daten vorhanden sind, nicht immer gegeben.
Um nichtsdestotrotz Erkenntnisse und Zusammenhänge aus den Daten zu bekommen, eignet sich unüberwachtes Lernen (Unsupervised Learning). Hierbei versucht das Modell Zusammenhänge, Muster und Strukturen aus den Daten zu erlernen, ohne die Hilfe einer Überwachung wie der Annotation der richtigen Outputs zu diesbezüglichen Inputs.[29] Beim unüberwachten Lernen muss der Rechner selbstständig die Lösung finden. Ein Beispiel dafür wäre, einem Schüler eine Menge an Mustern zu geben, mit der Aufgabe, herauszufinden, wie es zu diesen Mustern gekommen ist.[30]

Zum unüberwachten Lernen gehören **Clustering** Algorithmen. Diese versuchen Zusammenhänge in den Input-Daten zu erkennen und sie entsprechend ihrer Eigenschaften und Attributen zu Clustern zu gruppieren.[31]
Ein beliebter Algorithmus für das Clustering ist der k-means-Algorithmus [32]

[28] vgl. Louridas und Ebert, Machine Learning, 2016, S. 113.

[29] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 38.

[30] vgl. Louridas und Ebert, Machine Learning, 2016, S. 113.

[31] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 39.

[32] vgl. Louridas und Ebert, Machine Learning, 2016, S. 113.

Ein weiteres Kerngebiet des unüberwachten Lernens ist die **Dimensionsreduktion**. Dimensionsreduzierende Algorithmen zielen darauf ab, die Anzahl der Eigenschaften (Dimensionen) des Inputdatensatzes zu reduzieren. Der daraus entstandene reduzierte und komprimierte Datensatz bildet im Idealfall die Schlüsseleigenschaften der vorherigen Daten besser ab. [33]

2.2.3 Reinforcement Learning

Beim bestärkenden Lernen (Reinforcement Learning) festigt ein System seine Kenntnisse für die Lösung eines Problems durch einen Bestrafungs- und Belohnungsmechanismus. Ein Agent (Software oder Hardware) agiert in seinem Umfeld und eine beobachtende Instanz misst, wie viel Strafe oder Belohnung der Agent, basierend auf der Effektivität der Entscheidungen, die er mit dem Hinblick auf ein bestimmtes Ziel trifft, bekommt. Der Agent merkt sich die Strafe oder die Belohnung während der Lernphase und nutzt diese bei der Entscheidungsfindung für seine nächste Aktion, die ihm das beste Ergebnis zu einer gegebenen Umgebungssituation erbringen wird.[34]

2.2.4 Machine Learning für Zeitreihen

Wie bereits erwähnt, lässt sich die Vorhersage einer Zeitreihe als ein überwachtes Regressionsproblem modellieren.
Um Vorhersagen zu treffen, können statistische oder Deep Learning Modelle verwendet werden. Beide, sowohl statistische als auch Deep Learning Modelle, werden in dieser Arbeit vorgestellt und implementiert. Letztendlich werden beide Arten der Zeitreihenvorhersage miteinander verglichen und bewertet.

Im nächsten Abschnitt 2.3 werden statistische Modelle vorgestellt, darunter das ARIMA (Autoregressive Integrated Moving Average) Modell und Facebooks Prophet Modell.
Im darauffolgenden Abschnitt 2.4 werden Deep Learning Modelle MLP (Multilayer Perceptron), RNN (Recurrent Neural Network), LSTM (Long Short-Term Memory) und CNN (Convolutional Neural Network) detaillierter aufgezeigt.

Support Vector Machines und Random Forest Regression werden in dieser Arbeit nicht näher betrachtet, generell ist jedoch eine Zeitreihenvorhersage auch mit diesen Modellen möglich.

---

[33] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 40.

[34] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 42–43.

2.3 Statistische Machine Learning Methoden zur Vorhersage von Zeitreihen

Statistische Machine Learning Methoden sind im Gegensatz zum Deep Learning seit einigen Jahren für die Vorhersage von Zeitreihen im Einsatz und daher gut erforscht.
Sie werden in dieser Arbeit untersucht, da sie gegebenenfalls eine bessere Vorhersagegenauigkeit als Deep Learning Modelle, bei geringerer Speicher- und Zeitkomplexität, bieten.

Allerdings können die statistischen Modelle, die in dieser Arbeit vorgestellt werden, keinen Gebrauch von den Beziehungen der multivariaten Zeitreihen machen. Sprich, das Training ist nur an univariaten Zeitreihen möglich. Dies kann dazu führen, dass Deep Learning doch eine genauere Prognose erstellen könnten.[35]

2.3.1 ARIMA – Modelle

Der autoregressive integrierte gleitende Mittelwert, engl. **A**uto**r**egressive **i**ntegrated **m**oving **a**verage (ARIMA), ist ein Verfahren, entwickelt für die Abbildung oder Approximation von Zeitreihen zum Zweck der Analyse und Prognose. Die Anwendung von ARIMA für Zeitreihen geht auf Box und Jenkins und das von ihnen vorgeschlagene Box-Jenkins-Programm zurück.[36]

ARIMA ist die Erweiterung des ARMA Models, welches aus zwei Teilen besteht. Erstens aus dem autoregressiven Teil (**AR**). Dieser geht davon aus, dass zukünftige Werte einer Zeitreihe mithilfe linearer Kombinationen, die auf vergangenen Werten der Zeitreihe basieren, beschrieben werden können. Die Formel einer Vorhersage mit dem **AR**[p]-Modell ist folgende:

$$y_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i y_{t-i}$$

Formel 3: AR Modell

Dabei ist p die Ordnung, c eine Konstante, die Koeffizienten φ_i reelle Zahlen für die Gewichtung und ε_t weißes Rauschen. Durch p wird definiert, wie viele vergangene Werte für das Modell in Betracht genommen werden.[37]

---

[35] vgl. Selvin et al., Stock price prediction using LSTM, RNN and CNN-sliding window model, 2017, S. 1643.

[36] vgl. Contreras et al., ARIMA models to predict next-day electricity prices, 2003, S. 1014.

[37] vgl. Vogel, Prognose von Zeitreihen, 2015, S. 79–80.

Der zweite Teil ist der gleitende Mittelwert **MA**. Dieser nimmt an, dass Werte von Zeitreihen durch Schätzfehler abgebildet werden können. Dabei wir die Differenz des Rauschens zwischen dem geschätzten Wert und dem wirklich eingetroffenen Wert hergenommen. Die Formel einer Vorhersage mit dem **MA**[q]-Modell lautet wie folgt:

$$y_t = c + \varepsilon_t + \sum_{i=1}^{q} \vartheta_i \varepsilon_{t-i}$$

Formel 4: MA Modell

Dabei ist q die Ordnung, ε_t zentriertes weißes Rauschen, c eine Konstante, ϑ_i die Koeffizienten für die Gewichtung und ε_{t-i} die Differenzen der Rauschfehler. q definiert wie viele vergangene Rauschdifferenzen für das Modell hergenommen werden.[38]

Das AR[p] - und MA[q] - Modell kombiniert ergibt das **ARMA**[p, q] – Modell:

$$y_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{i=1}^{q} \vartheta_i \varepsilon_{t-i}$$

Formel 5: ARMA Modell

ARMA Modelle sind nur für die Abbildung und Prognose von linearen stationären Prozessen bzw. Zeitreihen geeignet. Mithilfe von einer Autokorrelations- (AC) und einer partiellen Autokorrelationsfunktion (PAC) kann getestet werden, ob Stationarität besteht.

Die Möglichkeit, eine nichtstationäre Zeitreihe in eine stationäre zu transformieren besteht, indem die Differenz zweier aufeinanderfolgender Werte der Zeitreihe gebildet werden. Dabei wird so oft die Differenz aufeinander folgender Werte genommen, bis die Zeitreihe stationär ist und das ARMA Modell für die Vorhersage verwendet werden kann. Die Häufigkeit, wie oft differenziert werden muss bis Stationarität erreicht wird, ist mit dem Parameter d beschrieben.

Abschließend müssen die vorhergesagten Werte d – mal integriert werden, um zur ursprünglichen nichtstationären Zeitreihe zu passen. Dieser ganze Prozess wird als **ARIMA**[p,d,q] bezeichnet.[39]

Beinhaltet eine Zeitreihe auch eine saisonale Komponente, so kann ARIMA zu SARIMA erweitert werden.

---

[38] vgl. Vogel, Prognose von Zeitreihen, 2015, S. 90.

[39] vgl. Konar und Bhattacharya, Time-Series Prediction and Applications, 2017, S. 4.

2.3.2 Facebook Prophet

Prophet ist ein von Facebook entwickeltes Modell zur Vorhersage von Zeitreihen basierend auf der Forschung von Sean Taylor und Benjamin Letham.

Es zielt darauf ab, eine Lösung für eine größtmögliche Variation von Vorhersageproblemen und für eine größtmögliche Anzahl an Anwendern, die mitunter kein Spezialwissen über Zeitreihenvorhersage verfügen, zu sein.
Die Implementierung des Modells ist für Python und R als Open Source verfügbar. Es wurde explizit so entwickelt, dass die Parameter eingestellt werden können, ohne die Details des darunterliegenden Modells zu kennen. [40] Dies grenzt es von ARIMA ab, bei dem durchaus Kenntnisse über das Modell, als auch über die Beschaffenheit der Zeitreihe für die richtige Einstellung der Modellparameter notwendig sind.

Prophet verwendet ein zerlegbares Modell mit Trend, Saisonalität und Feiertagen als die drei Hauptkomponenten beschrieben mit folgender Gleichung:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Formel 6: Facebook Prophet Gesamtmodell

Dabei ist $g(t)$ der Trend und modelliert nichtperiodische Veränderungen der Zeitreihe, $s(t)$ bildet die periodischen Veränderungen die zum Beispiel durch eine wöchentliche oder jährliche Saisonalität hervorgebracht werden, $h(t)$ repräsentiert die Effekte von Feiertagen über einen oder mehreren Tagen auf die Zeitreihe. ε_t steht für alle anderen übriggebliebenen Veränderungen, die durch die vorhergenannten Funktionen nicht erfasst werden konnten.[41]

Für die Trendkomponente $g(t)$ kommt je nach Beschaffenheit der Zeitreihe entweder ein modifiziertes logistisches Wachstumsmodell, mit sich über die Zeit verändernder Wachstumsrate und oberer Schranke der Wachstumskapazität, oder ein lineares Modell infrage.[42]

Die Saisonalität $s(t)$ kann mithilfe von einer Fourier Reihe wie folgt approximiert werden:

[40] vgl. Taylor und Letham, Forecasting at scale, 2017, S. 2–5.

[41] vgl. Taylor und Letham, Forecasting at scale, 2017, S. 7.

[42] vgl. Taylor und Letham, Forecasting at scale, 2017, S. 8–9.

$$s(t) = \sum_{n=q}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right)$$

Formel 7: Saisonale Komponente des Prophet Modells

P ist dabei die erwartete reguläre Periode, in der die Saisonalität einer Zeitreihe auftritt (z.B. P = 365,25 für jährliche oder P = 7 für wöchentliche Daten). Um die Saisonalität nachbilden zu können, ist die Schätzung der 2N Parameter $\beta = [a_1, b_2, \dots, a_N, b_n]^T$ notwendig.

Dies wird erreicht, indem eine Matrix von Vektoren für die Saisonalität für jeden der historischen und für alle zukünftigen Werte t der Zeitreihe gebildet wird. Beispielsweise mit einer jährlichen Saisonalität und N=10:

$$X(t) = \left[cos\left(\frac{2\pi(1)t}{365{,}25}\right), \dots, sin\left(\frac{2\pi(10)t}{365{,}25}\right) \right]$$

Die Saisonale Komponente ist dann:

$$s(t) = X(t)\beta$$

mit $\beta \sim N(0, \sigma^2)$.

Mit einem höheren N kann das Modell an sich schneller ändernde Saisonalitätsmuster angepasst werden, jedoch mit der Gefahr des Overfittings.[43]

Feiertage und Events verursachen große und einigermaßen vorhersagbare Schocks (abrupte Steigung oder Senkung) in geschäftlichen Zeitreihen. Diese können nicht einfach zyklisch wie die Saisonalität definiert werden, da nicht alle Feiertage in regelmäßig definierten Perioden fallen (beispielsweise Ostern). Die Auswirkungen von Feiertagen sind von Jahr zu Jahr ähnlich und es ist wichtig, diese in der Vorhersage mit zu beachten. Der Benutzer kann deshalb eine benutzerdefinierte Liste an Feiertagen dem Modell übergeben. Dieses wird dann auch die entsprechenden Tage vor und nach dem Feiertag gesondert in Betracht ziehen, da auch diese einen Effekt haben können.[44]

2.4 Deep Learning Methoden zur Vorhersage von Zeitreihen

Wie in Kapitel 2.2 verdeutlicht, ist das Deep Learning ein Teilgebiet des Machine Learnings. Als Deep Learning wird das Lernen durch mehrere aneinandergereihten

[43] vgl. Taylor und Letham, Forecasting at scale, 2017, S. 11–12.

[44] vgl. Taylor und Letham, Forecasting at scale, 2017, S. 12–13.

Schichten (Layer) verstanden. Jede Schicht repräsentiert und verarbeitet die Eingabedaten und lernt dies in einem automatischen Prozess, während das gesamte Modell Trainingsdaten ausgesetzt ist. Die Anzahl der Schichten eines Modells wird als Tiefe bezeichnet. Dabei können moderne Deep Learning Modelle zehn bis hunderte von Schichten umfassen. Andere Methoden des Machine Learnings können sich nur mit einer oder zwei Schichten der Datenrepräsentation befassen, wie die im Kapitel 2.3 vorgestellten statistischen Modellen für Zeitreihen Vorhersage, und werden deshalb häufig als Shallow Learning bezeichnet.[45]

In Abbildung 9 ist das Konzept der Schichten und deren Repräsentationen zu sehen. Als Input wird ein Bild einer handgeschriebenen Zahl eingegeben. Jede der hierarchisch aufeinander aufbauenden Schichten zerlegt den ursprünglichen Input in seine eigene zuvor gelernte Abstrakte Repräsentation und übergibt diese an die nächste Schicht weiter. Die letzte Schicht ist dann in der Lage, basierend auf der Verarbeitung der Schichten davor, zu erkennen um welche handgeschriebene Zahl es sich handelt.

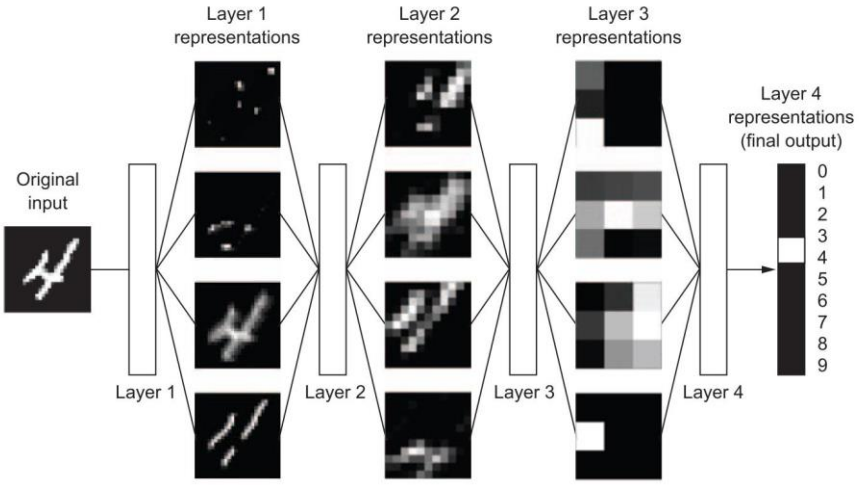


Abbildung 9: Deep Learning Schichten und ihre Repräsentation der Eingabedaten
Quelle: (Chollet, 2018, S. 9)

Beim Deep Learning werden diese Schichten mithilfe von künstlichen neuronalen Netzwerken, abgekürzt KNN, gelernt.

2.4.1 Neuronale Netzwerke und das Multilayer Perceptron

Innovationen bei KNN Architekturen und die Verfügbarkeit von günstiger Rechenleistung brachten das Deep Learning mittels KNNs zum Aufblühen. KNNs können

[45] vgl. Chollet, Deep learning with Python, 2018, S. 8.

dabei für alle Zwecke des Machine Learnings (Klassifikation, Regression, Clustern, Dimensionsreduktion) verwendet werden.[46]

Der Aufbau von KNNs ist inspiriert vom menschlichen Gehirn. Ein KNN hat Schichten von mehreren miteinander verbundenen Knoten. Diese miteinander verbundenen Knoten können als Netzwerk von Neuronen betrachtet werden. Wie in Abbildung 10 demonstriert, hat ein typisches KNN eine Input Schicht, eine Output Schicht und mindestens eine versteckte Schicht (Hidden Layer) zwischen der Input- und der Output Schicht. [47] Ein KNN, dass mindestens diese Anforderungen an die Architektur erfüllt, wird auch als ein Multilayer Perceptron (MLP) bezeichnet. [48]

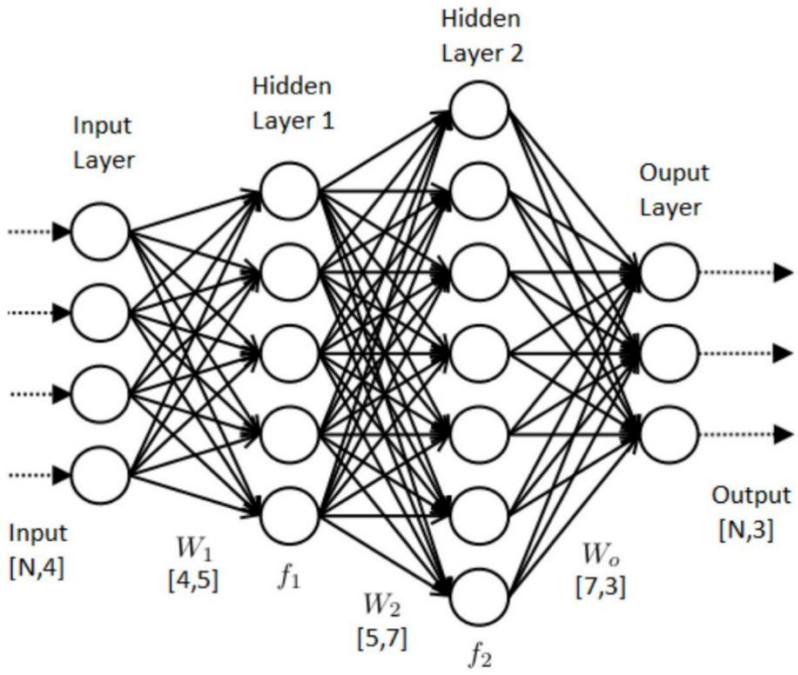


Abbildung 10: Künstliches neuronales Netz
Quelle: https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4, 22.03.2020

Jeder Knoten bzw. jedes künstliche Neuron einer Schicht ist voll verbunden mit allen Neuronen der nachfolgenden Schicht. Die Stärke einer Verbindung ist durch einen Faktor w (Weight) gekennzeichnet. Die Neuronen, die sich in der gleichen Schicht befinden sind untereinander nicht verbunden. Jedes Neuron hat demnach mehrere gewichtete Inputs. Die gewichteten Inputs kommen von den Outputs der Neuronen der

[46] vgl. Louridas und Ebert, Machine Learning, 2016, S. 114–115.

[47] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 31.

[48] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 32.

vorherigen Schicht. Sie werden aufsummiert und mittels einer Aktivierungsfunktion zu Outputs transformiert. Die Outputs werden an die Neuronen der nachfolgenden versteckten Schichten und irgendwann schließlich zur Output Schicht weitergereicht. Deshalb werden solche neuronalen Netze auch als Feedforward-Netze bezeichnet.

Abbildung 11 veranschaulicht den internen Aufbau eines künstlichen Neurons. Zu sehen sind die zuvor erwähnten gewichteten Inputs, die Aufsummierung dieser und die anschließende Transformation mittels einer Aktivierungsfunktion zu Outputs. Die beliebtesten Aktivierungsfunktionen sind Sigmoid, TanH und ReLu.

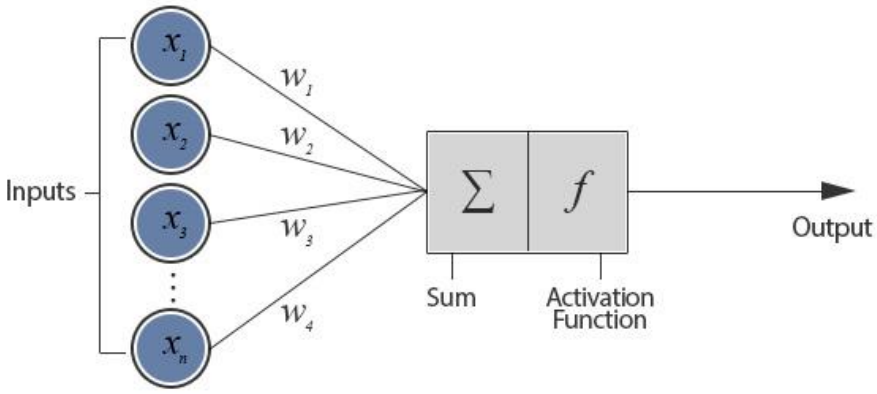


Abbildung 11: Aufbau eines künstlichen Neurons
Quelle: https://www.cc.gatech.edu/~san37/post/dlhc-fnn/, 22.03.2020

Die Spezifikation wie eine Schicht ihre Input Daten verarbeitet und transformiert, ist gespeichert in den Gewichtungen w (Weights) der Verbindungen der Neuronen. In diesem Kontext bedeutet lernen, einen Satz von Werten für die Gewichtungen aller Schichten zu finden, und zwar so, dass das gesamte Netzwerk beispielhafte Inputs korrekt zu dem zu ihnen assoziierten Ziel-Output abbildet.[49]

Um den Output des neuronalen Netzes kontrollieren zu können, ist es nötig zu messen, inwieweit der Output sich zu dem gewünschten Ergebnis unterscheidet. Dafür wird eine Loss-Funktion hergenommen. Diese nimmt die Vorhersagen des KNNs und die eigentlich zu erwartenden Ergebnissen, vergleicht diese und berechnet eine Bewertung, wie gut das neuronale Netzt seine Aufgabe bei diesem Beispiel erfüllte.[50]

Diese berechnete Bewertung wird dann genutzt, um die Gewichtungen der Verbindungen mithilfe eines Optimizers so anzupassen, dass die zukünftige Bewertung

[49] vgl. Chollet, Deep learning with Python, 2018, S. 10.

[50] vgl. Chollet, Deep learning with Python, 2018, S. 10.

sich eventuell verbessern wird. Dieser Prozess der Nachjustierung wird Backpropagation genannt und ist in Abbildung 12 skizziert.

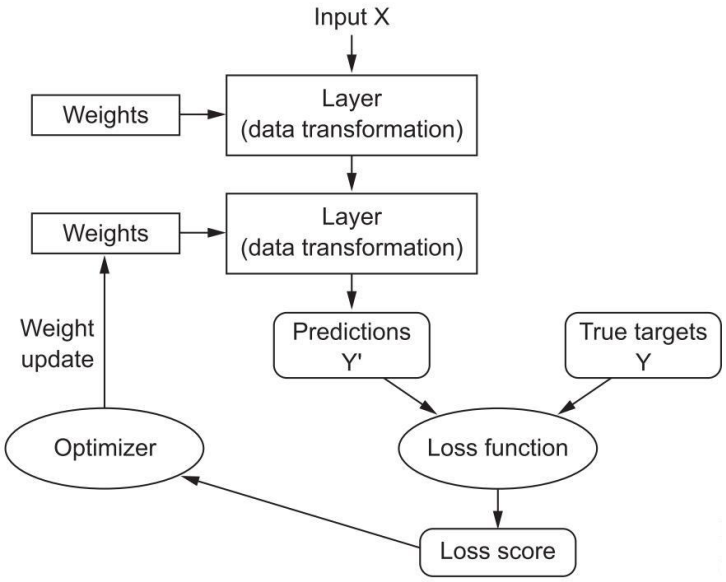


Abbildung 12: Anpassung der Verbindungsgewichtungen
Quelle: (Chollet, 2018, S. 58)

Im Folgenden wird detaillierter darauf eingegangen, wie die Trainingsprozedur bzw. das Lernen stattfindet.

Voraussetzung für das Training ist ein entsprechender Trainingsdatensatz, der nach den Anforderungen des Supervised Learnings umgeformt wurde. Zu Beginn des Trainings sind alle Gewichtungen des neuronalen Netzwerks zufällig gewählt, sodass alle Schichten beliebige Transformationen an den Input Daten ausüben. Der entsprechende Output wird danach weit entfernt von dem gewünschten und richtigen Ergebnis liegen und die Bewertung der Loss-Funktion schlecht sein. Jedoch werden nach jedem Beispiel aus dem Trainingsdatensatz die Gewichtungen der Layer angepasst, sodass sich die Bewertung durch die Loss-Funktion verbessert. Das ist die sogenannte Trainingsschleife, die so oft wiederholt wird, bis die Gewichtungen gefunden werden, bei der die Loss-Funktion die beste Bewertung hat und der Unterschied zwischen dem Output des Models und der tatsächlich richtigen Ergebnisse minimal ist. So entsteht am Ende das trainierte KNN.[51]

[51] vgl. Chollet, Deep learning with Python, 2018, S. 11.

2.4.2 Recurent Neural Network

Ein rekurrentes neuronales Netzwerk, abgekürzt RNN, ist eine spezielle Art von neuronalem Netzwerk, die es erlaubt, Informationen basierend auf vergangenem Wissen zu persistieren. Dies wird mit Hilfe einer geschleiften Architektur erreicht.
RNNs werden oft für Daten, die in Form von Sequenzen auftauchen verwendet, etwa wie die Vorhersage des nächsten Wortes in einem Satz oder wie in diesem Fall den zukünftigen Wert einer Zeitreihe.[52]
Das RNN verarbeitet Sequenzen indem es über dessen Elemente iteriert und sich den Zustand der Informationen merkt, die es bisher gesehen hat.[53]

Abbildung 13 zeigt die Struktur eines RNN in kompakter Weise (links) und in der aufgerollten Form (rechts) abhängig von der Länge der eingegebenen Sequenz.

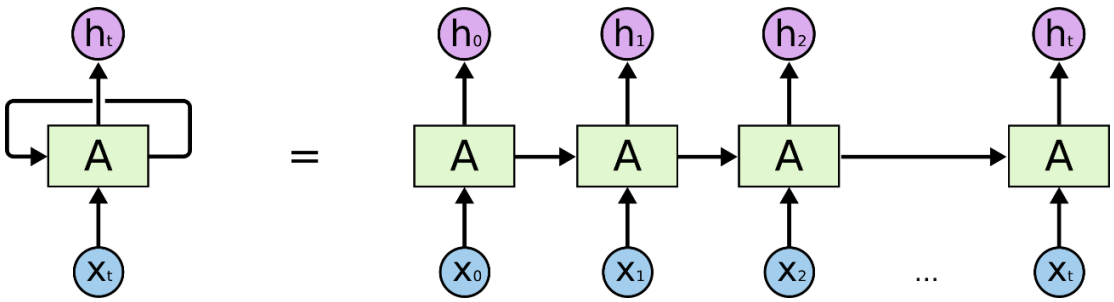


Abbildung 13: Aufbau eines Recurrent Neural Networks
in kompakter (links) und aufgerollter Form (rechts)
Quelle: https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 22.03.2020

Ein rekurrentes neuronales Netzwerk ist im Aufbau ähnlich zu dem im vorherigen Abschnitt beschriebenen feedforward neural network. Allerdings gibt es beim RNN zusätzlich Verbindungen zwischen den Neuronen der gleichen Schicht. Wie links in der Abbildung 13 dargestellt, bekommt ein RNN Neuron einen entsprechenden Input, produziert einen Output und schickt diesen einmal nach außen und zudem zurück in die eigene Architektur zum nächsten RNN Neuron. Diese kompakte Form kann horizontal entlang einer Zeitreihe aufgerollt werden (rechts in der Abbildung).
Zu jedem Zeitpunkt t bekommt das RNN einen Input $x(t)$ als auch den eigenen Output von dem vorherigen Zeitpunkt $h(t-1)$. Für den ersten Zeitpunkt $x(0)$ gibt es keinen vorherigen Output einer RNN Zelle, sodass dieser mit 0 gesetzt ist. Ein rekurrentes

[52] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 33.

[53] vgl. Chollet, Deep learning with Python, 2018, S. 196.

neuronales Netz bekommt also als Input eine Sequenz, in diesem Fall einen historischen sequenziellen Abschnitt einer Zeitreihe, und gibt wiederum eine Sequenz als Output, nämlich eine Vorhersage basierend auf den historischen Werten. Da der Output eines RNN Neurons zum Zeitpunkt t eine Funktion von allen vorherigen Inputs ist, kann gesagt werden, dass ein RNN eine Art Erinnerungsvermögen hat.[54]

Auch wenn RNNs auf den ersten Blick sehr gut für die Prognose von Zeitreihen geeignet erscheinen, da sie dafür mehrere Vergangenheitswerte mit in Betracht ziehen, weisen sie ein gravierendes Manko auf. Sie sind nämlich nur für vergleichsweise kurze Sequenzen geeignet. Auch wenn sie theoretisch zu einem Zeitpunkt Inputinformationen bewahren, aus Zeitpunkten, die sie vor langer Zeit gesehen haben, so ist es praktisch nicht möglich, diese in der Vergangenheit länger zurückliegende Abhängigkeiten zu lernen. Falls ein RNN eine lange Sequenz verarbeitet, vergisst es währenddessen nach und nach die ersten Inputs der Sequenz. Dies ist als das Vanishing Gradient Problem bekannt und wurde von Schmidhuber, Hochreiter und Bengio in den frühen 1990er Jahren erforscht.[55]

[54] Vgl. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019, S. 498–501.

[55] vgl. Chollet, Deep learning with Python, 2018, S. 202.

2.4.3 Long Short-Term Memory

Long Short-Term Memory Netzwerke, abgekürzt LSTM, sind eine weiterentwickelte Art von RNNs die das zuvor genannte Vanishing Gradient Problem lösen. Entwickelt wurde es von Hochreiter und Schmidhuber ausgehend aus ihrer Forschung zum Vanishing

Gradient Problem. LSTMs können längere Sequenzen als RNN verarbeiten, ohne dass Informationen die länger in der Sequenz zurückliegen verloren gehen. Dies ist durch eine komplexere Struktur der LSTM Neuronen gegenüber den RNN Neuronen zu verdanken.[56]

Abbildung 14 zeigt links den internen Aufbau eines RNN Neurons. Dieser ist, verglichen mit dem Aufbau eines LSTM Neurons, relativ simpel. Lediglich eine Tanh Funktion dient als Aktivierungsfunktion.

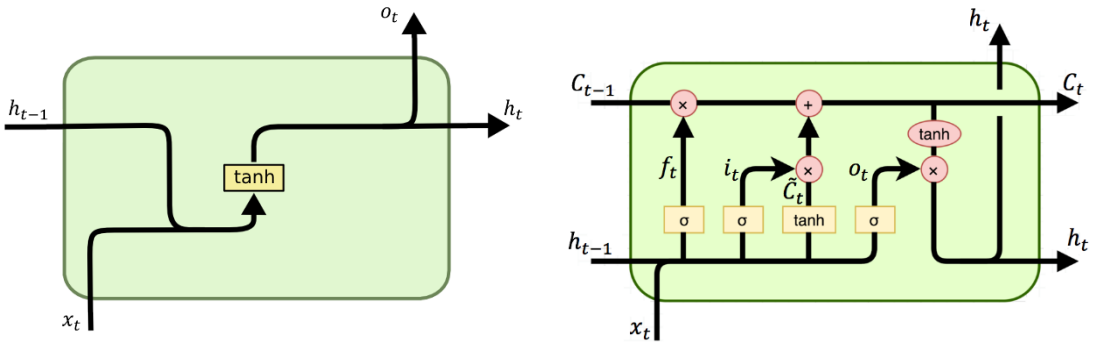


Abbildung 14: Der Aufbau eines RNN (links) und eines LSTM Neurones (rechts)
Quelle: http://dprogrammer.org/rnn-lstm-gru, 22.03.2020

Der interne Aufbau eines LSTM Neurons in der rechten Abbildung besitzt drei Tore (Gates), die gegenüber dem RNN Neuron zusätzliche Funktionalität bringen. Dabei werden ein Forget-Gate, ein Input-Gate und ein Output-Gate verwendet.
Ct ist der Zellenzustand und kann als kurzeitiges (short-term), ht der versteckte (hidden) Zustand, als langfristiges (long-term) Gedächtnis gesehen werden.[57] Das Forget Gate f_t entscheidet darüber, welche Informationen behalten und welche verworfen werden. Der versteckte Zustand h_{t-1} aus dem vorherigen LSTM Neuron und der aktuelle Input Wert x_t der zu verarbeitenden Zeitreihe wird miteinander verrechnet und an die Sigmoid Funktion σ übergeben. Dabei kommen Werte zwischen 0 und 1 raus. Ist der Wert näher an 0 wird er vergessen, ist er näher an 1 wird er beibehalten.

[56] vgl. Chollet, Deep learning with Python, 2018, S. 202.

[57] vgl. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019, S. 515.

Das Input-Gate i_t aktualisiert den Zellenzustand C_t. Auch hier entscheidet der Output zwischen der Sigmoid Funktion σ zwischen 1 und 0 inwieweit der durch die Tanh Funktion ermittelte Wert $\tilde{C}_t$ den Zellenzustand C_t verändert. Der ursprüngliche Zellzustand C_{t-1} wird also zuerst mit dem Forget-Gate f_t multipliziert, was dazu führen kann, dass er gelöscht ("vergessen") werden kann, wenn f_t nahe 0 ist. Danach wird der Wert des Input-Gates i_t dazu addiert, was dazu führt, dass dieser, falls nötig, verändert wird und der neue Zellenzustand C_t entsteht.
Zu guter Letzt beeinflusst das Output-Gate den versteckten Zustand h_t. Dieser beinhaltet die längerfristigen Informationen der vergangenen Inputs. Der ursprüngliche versteckte Zustand h_{t-1} wird gemeinsam mit dem aktuellen Input-Wert x_t in eine Sigmoid Funktion eingegeben. Zudem wird der kürzlich aktualisierte Zellenzustand C_t durch tanh verarbeitet. Die Ergebnisse der Sigmoid und Tanh Funktion werden miteinander multipliziert und es entsteht der neue versteckte Zustand h_t. Die zwei neuen werte C_t und h_t werden dann an die nächste LSTM Zelle übertragen. [58]

2.4.4 Convolutional Neural Network

Convolutional Neural Networks, abgekürzt CNNs, sind eine spezielle Art von neuronalen Netzwerken für die Verarbeitung von gitterartigen Daten. Beispiele für gitterartige Daten sind das eindimensionale Gitter bzw. ein Vektor von Beobachtungen zu einem Zeitpunkt einer Zeitreihe, oder Bilder, die als zweidimensionales Gitter von Pixeln (Höhe x Breite) gesehen werden können. Aus dem Namen Convolutional Neural Network lässt sich erahnen, dass eine mathematische Operation namens Convolution (dt. Faltung) ausgeführt wird. CNNs sind daher künstliche neuronale Netze, die mindestens in einem der hidden Layer eine Convolution der Input Daten ausführen.[59]

Die Convolution ist in Abbildung 15 dargestellt. Sie besteht aus einem Input, einem Receptive Feld, einem Kernel und der resultierenden Feature Map.
Der Input kann in zweidimensionaler oder eindimensionaler Form sein. Ein Teilbereich des Inputs wird in den Fokus gestellt (Receptive Field in der Abbildung ein 3x3 Feld) und wird mit dem Kernel verrechnet (in diesem Fall multipliziert). Der Kernel ist eine Matrix aus Gewichten. Die Gewichte sind zu Beginn zufällig gewählt und werden beim Training des gesamten neuronalen Netzes angepasst. Die aus der Multiplikation des Receptive Felds mit dem Kernel resultierenden Produktwerte werden addiert und in der Feature

[58] vgl. Nguyen, Illustrated Guide to LSTM's and GRU's: A step by step explanation, 2018, o.S.

[59] vgl. Goodfellow et al., Deep learning, 2016, S. 330.

Map eingetragen. Danach wird das Receptive Feld um die Anzahl des sogenannten Strides (hier 1) nach rechts, oder falls am rechten Rand und es möglich ist (2D Input), nach unten verschoben.

Die neuen durch das Receptive Field im Fokus liegenden Inputwerte, werden erneut wie zuvor beschrieben berechnet. Dies wird solange wiederholt, bis der gesamte Input abgearbeitet worden ist.

Die durch diese Prozedur erhaltene Feature Map hat eine niedrigere Dimension als der ursprüngliche Input und wird als Eingabe für die nachfolgende Schicht verwendet.[60]

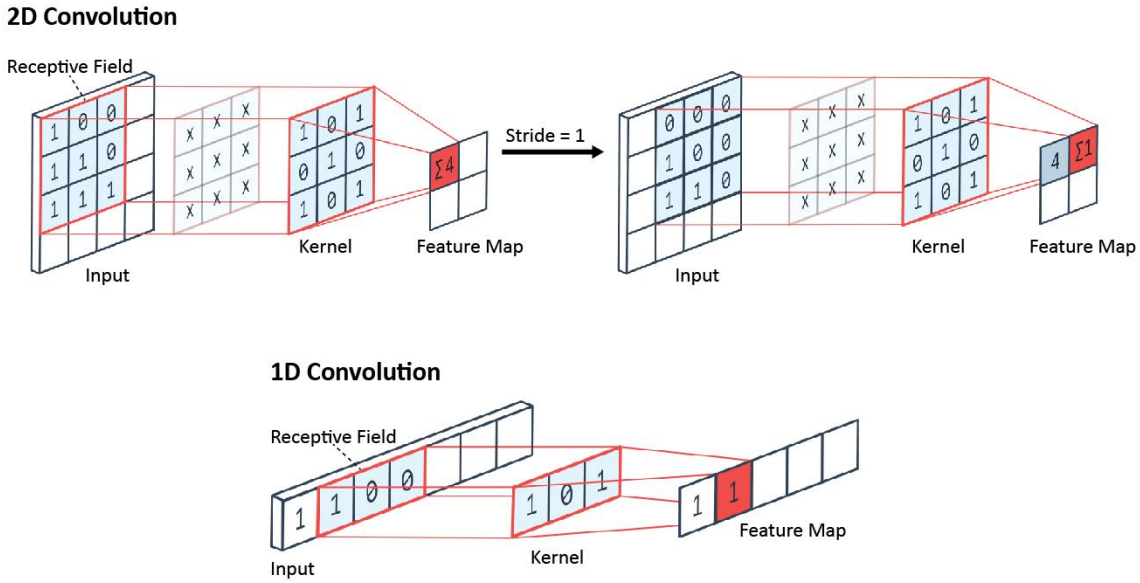


Abbildung 15: 2D Convolution (oben) und 1D Convolution (unten)
Quelle: In Anlehnung an https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/ 2d-convolution-block, 22.03.2020

Meistens wird nach der Convolution Schicht eine Pooling Schicht geschaltet. Die Pooling Schicht arbeitet sich ähnlich über die Inputdaten durch und fasst diese zusammen, etwa indem es den Durchschnitt, das Maximum oder das Minimum des derzeitigen Receptive Fields bestimmt.[61] In Abbildung 16 ist das Max Pooling zu sehen, bei dem aus dem Perceptive Field immer der jeweils maximale Wert herausgefiltert wird.

[60] vgl. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019, S. 450.

[61] vgl. Goodfellow et al., Deep learning, 2016, S. 339–342.

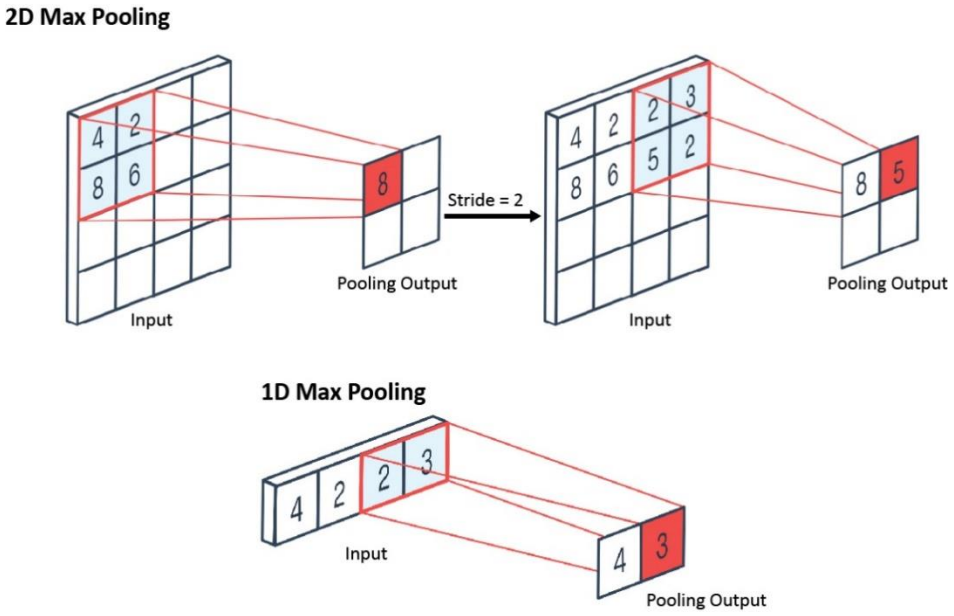


Abbildung 16: 2D Max Pooling (oben) 1D Max Pooling (unten)
Quelle: In Anlehnung an https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-max-pooling-block, 22.03.2020

Die Verwendung von Pooling Schichten reduziert die Anzahl der Koeffizienten der Future Maps, die von nachfolgenden Schichten verarbeitet werden müssen. Dies führt dazu, dass auch die Rechenlast und die Speichernutzung gesenkt werden. [62]

Zudem ermöglicht das Pooling die Verarbeitung des Inputs in Hierarchien, indem sie aneinander gereihte Convolution Schichten zu dem Blick auf ein größeres Input Fenster zwingt.[63] Dies wird deutlicher, wenn man Abbildung 17 betrachtet. Auf dieser ist beispielhaft der Aufbau eines CNNs mit mehreren Convolutional- sowie Pooling-Schichten dargestellt.

[62] vgl. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019, S. 456–457.

[63] vgl. Chollet, Deep learning with Python, 2018, S. 127–128.

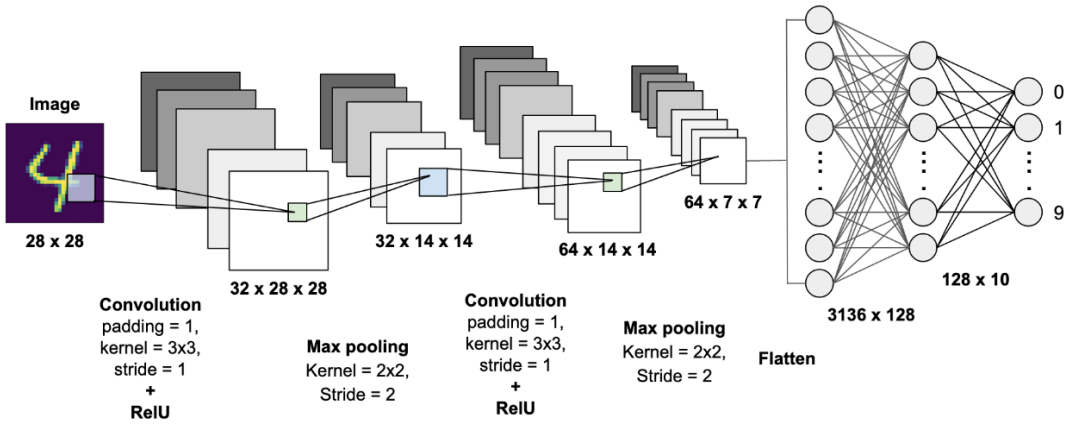


Abbildung 17: Aufbau eines Convolutional Neural Networks
Quelle: https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9, 22.03.2020

Durch diese Architektur haben CNNs nützliche Fähigkeiten. Bei der Bildverarbeitung können gelernte Muster an beliebig anderen Stellen des Inputs wiedererkannt werden. Und wie erwähnt, können sie den Bezug zu dem hierarchischen Aufbau eines Bildes darstellen, indem die erste Convolutions-Schicht beispielsweise die Kanten in dem eingegangen Bild herausfiltert, und die darauffolgenden Schichten aus den davor gefilterten Kanten weitere größere Muster zusammenschließt.[64]
Ein spezifischer Vorteil von CNNs im Hinblick auf Zeitreihen ist, dass sie in der Lage sind, automatisch die benötigten Features aus den Rohdaten, die von Nutzen für die Lösung des adressierten Problems sein könnten, zu erfassen.[65]

Da CNNs, wenn sie Abschnitte einer Zeitreihe als Input bekommen, nicht die zeitliche Reihenfolge der Beobachtungen bei der Verarbeitung beibehalten, wird nach den hintereinander geschachtelten Convolutions- und Pooling-Schichten üblicherweise noch ein Feedforward Network wie RNN oder LSTM eingesetzt. So wird das CNN in zwei Abschnitte eingeteilt. Der Feature Extraktion mit den Convolution- und Pooling-Schichten und dann der Regression mit einem Feedforward LSTM. Dadurch wird die zeitliche Reihenfolge der Beobachtungen einer Zeitreihe gewahrt und es können noch längere Sequenzen vorhergesagt werden als mit LSTM oder RNN allein.[66]

[64] vgl. Chollet, Deep learning with Python, 2018, S. 123.

[65] vgl. Brownlee Jason, Deep Learning for Time Series Forecasting, 2019, S. 5.

[66] vgl. Chollet, Deep learning with Python, 2018, S. 228–229.

3 Analyse und Konzeption

Wenn mit einem ausreichenden Horizont die Störungen oder Ausfälle in Rechenzentren zuverlässig vorhergesagt werden können, kann schneller reagiert und entsprechende Maßnahmen getroffen werden, um vermeintliche Kosten und Schäden zu reduzieren.

Um herauszufinden, wie gut die Modelle für die Vorhersage von Serverauslastungen geeignet sind, wird eine quantitative Forschung durchgeführt.
Die zuvor vorgestellten Modelle werden dafür implementiert, dies ist in Kapitel 4 beschrieben. Die von der Bundesagentur für Arbeit zur Verfügung gestellten Serverdaten werden zuerst analysiert, gegebenenfalls angepasst und für das Training der Machine Learning Modelle herangezogen.

Es wird ein quantitatives Vergleichsexperiment vollzogen, welches folgende Fragen beantworten soll:

- Kann die Serverauslastung mithilfe von Machine Learning zuverlässig vorhergesagt werden?
- Wie hoch ist die Genauigkeit der Vorhersage?
- Wie schnell wird ein Modell trainiert?
- Welches Modell ergibt die höchste Vorhersagegenauigkeit?
- Kann ein Modell für die Vorhersage von Serverauslastungen empfohlen werden?

In diesem Kapitel wird näher auf die Konzeption des Experiments eingegangen und die notwendigen Schritte bei der Ausführung beschrieben. Die Ergebnisse des Experiments sind in Kapitel 5 aufgezeigt.

3.1 Vorgehen beim Vergleichsexperiment der Modelle

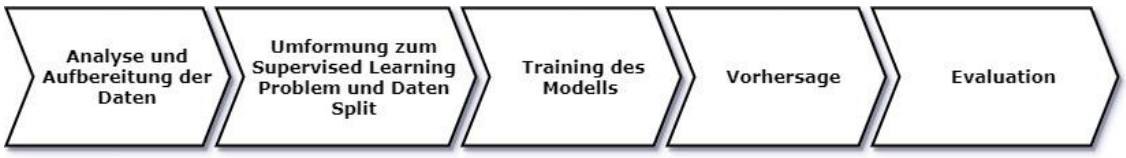


Abbildung 18: Vorgehen beim Vergleichsexperiment
Quelle: eigene Darstellung

Bei dem Vergleichsexperiment wird wie in Abbildung 18 vorgegangen.
Zuerst werden die zur Verfügung gestellten Daten (Zeitreihen) analysiert und gegebenenfalls bereinigt sowie angepasst. Danach folgt die Umformung des Datensatzes als Supervised Learning Problem. Bei den statistischen Modellen Prophet und ARIMA ist die Umformung zum Supervised Learning Problem nicht notwendig.

Im nächsten Schritt werden die Modelle trainiert, wonach anschließend eine Vorhersage getroffen und evaluiert wird. Hierfür wird ein Abschnitt der real stattgefundenen Zeitreihe extrahiert und mit der Vorhersage verglichen. Folglich soll auf die einzelnen Schritte detaillierter eingegangen werden.

3.2 Analyse und Aufbereitung der Daten

Zu Beginn des Vergleichsexperiments und jedes Machine Learning Problems ist zuerst die genaue Betrachtung und Analyse der vorhandenen Daten notwendig. Wie in Kapitel 2.1.2 erwähnt, kann die Beschaffenheit einer Zeitreihe bereits mit dem Plotten gut abgeschätzt werden.

3.2.1 Quelle und Zustand der Daten

Die Daten wurden von der Bundesagentur für Arbeit (BA) zur Verfügung gestellt. Dabei handelt es sich um tatsächlich eingetroffene Serverdaten, die aus mehreren Anwendungen der IT-Infrastruktur der Bundesagentur für Arbeit über einen Zeitraum von einem Monat gesammelt wurden.
Sie beinhalten die Auslastungen existenzkritischer Applikationen, die in den Einrichtungen der BA beispielsweise für die Berechnung und monatliche Auszahlung des Kindergeldes oder des Arbeitslosengeldes II (umgangssprachlich Hartz IV) dienen. Daher ist das Interesse groß, dass die Server dieser Anwendungen stets verfügbar sind.
Da diese Aufzeichnungen der Serverauslastung potenziell sicherheitsgefährdend und daher sensible Daten sind, wurden die jeweiligen Servernamen bzw. die Applikationszugehörigkeit der Messwerte anonymisiert. Es ist also nicht nachvollziehbar und nachbildbar, welche Serverauslastungen genau zu welcher Applikation der BA gehört.

Insgesamt sind über hundert Serverinstanzen in den bereitgestellten Daten verfügbar und für jede wurden bis zu zweihundert Auslastungswerte aufgezeichnet. Die Anzahl der gemessenen Metriken variiert tatsächlich zwischen den Serverinstanzen, da dies abhängig von der dahinter liegenden Applikation ist.

Für das Vergleichsexperiment wurde eine Serverinstanz zufällig ausgesucht und die dazugehörenden Auslastungswerte rausgefiltert. Anhand dieser wird das Experiment vollzogen, für die restlichen Serverinstanzen kann die Methodik analog durchgeführt werden.

Die für das Experiment herausgefilterte und verwendete Serverinstanz hatte 182 Messwerte die fortan als Features bezeichnet werden. Bei genauerer Betrachtung der

Features fiel auf, dass nicht alle im gleichen Zeitintervall gemessen worden sind. Die Messintervalle variierten von alle 10 Minuten bis 30 Minuten. Es war deshalb notwendig, alle Messwerte auf das gleiche Zeitintervall von 15 Minuten anzupassen.

Um eine Vorstellung für die Daten nach der Intervallanpassung zu bekommen, zeigt Abbildung 19 einen einwöchigen Auszug von sieben der 186 gemessenen Werte, jeweils im eigenen Graph geplottet.

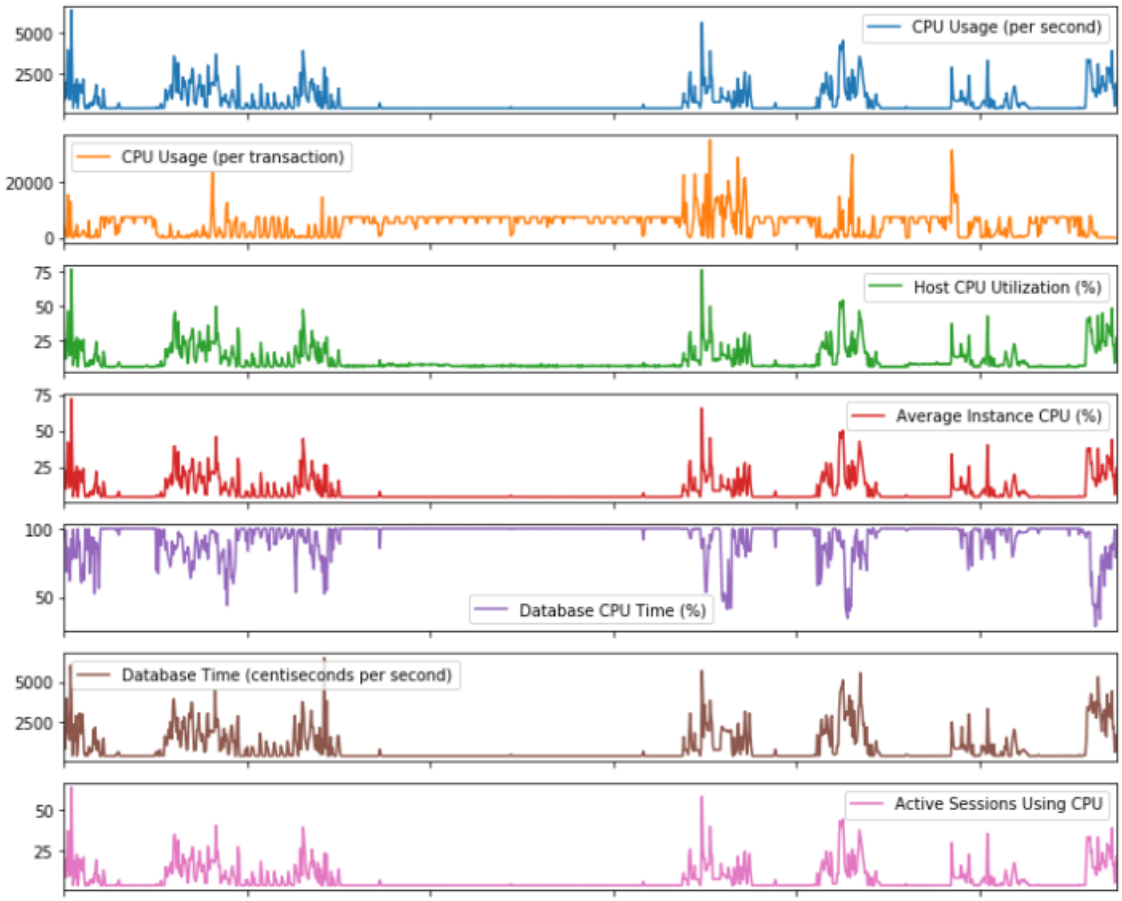


Abbildung 19: Einwöchiger Auszug der Daten
Quelle: eigene Darstellung

Bei den Deep Learning Modellen wurden die 182 Features oder einzelne Zeitreihen als multivariater Input für die Modelle herangezogen, um die CPU Usage per Second vorherzusagen. Die vorgestellten statistischen Modelle sind nicht in der Lage multivariaten Input zu verarbeiten, also mit Hilfe mehrerer in Bezug stehender Zeitreihen die Vorhersage für die anvisierte Zeitreihe zu tätigen. Deshalb wurde der Verlauf der CPU Usage per Second univariat in Abhängigkeit von nur seinem eigenen Verlauf vorhergesagt.

3.2.2 Feature Engineering

Der Datensatz aus Serverauslastungen mit den jeweiligen Messwerten als Features wurde zusätzlich manuell durch eigens entwickelte Features angereichert. Dies wird als Feature Engineering bezeichnet. Es ist der Prozess, bei dem aus den Rohdaten Features gewonnen werden, die als Input für Machine Learning Modelle verwendet werden. Die Wahl von qualitativen Features ist vom zu lösendem Problem abhängig und erfordert oftmals Fachwissen. Entsprechend gute Features helfen dabei, die Vorhersagegenauigkeit der Modelle zu verbessern.[67]

In diesem Fall wurden aus dem vorhandenen Zeitstempel weitere Features entwickelt. Dabei wurde aus dem Datum herausgefiltert, ob ein Wochenende vorliegt, um welchen genauen Wochentag, um welche Stunde und um welche Minute es sich handelt, und zu den bestehenden Daten angereichert. Zu den bereits vorhandenen 182 kommen also 4 hinzu.

Das Feature Engineering wird bei ARIMA und Prophet nicht vollzogen, da diese wie bereits gesagt, nur univariate Daten verarbeiten können. Sprich immer nur ein Messwert (Feature) als Input bekommen können.

3.2.3 Standardisierung und Differenzierung

Für das Prophet Modell ist keine weitere Verarbeitung der Daten notwendig. Dieses kann die Daten flexibel in der vorhandenen Form annehmen.

Für das ARIMA Modell wird es nötig sein, die Anzahl der Differenzierung zu bestimmen, da davon ausgegangen werden kann, dass die als Input dienenden Zeitreihen nicht stationär sind. Es gilt also den d Parameter zu bestimmen der gegen Ende des Kapitel 2.3.1 beschrieben worden ist.

Für die Deep Learning Modelle wird noch eine Standardisierung der Daten vollzogen. Dabei werden die Werte der Zeitreihen auf einen Bereich zwischen -1 und +1 abgebildet. Somit können sie leichter von den Modellen verarbeitet werden.

3.3 Umformung zum Supervised Learning Problem

Wie in Kapitel 2.2.1 erwähnt, kann die Vorhersage von Zeitreihen als ein Supervised Learning Regressionsproblem definiert werden. Dafür wird ein sogenannter gelabelter Datensatz für das Training der Modelle benötigt.

---

[67] vgl. Sarkar et al., Practical Machine Learning with Python, 2018, S. 182.

Dieser beinhaltet zu den entsprechenden Inputs den passenden richtigen Output. Aus Zeitreihen-Daten kann man diesen gelabelten oder gekennzeichneten Datensatz mithilfe des Sliding-Window-Verfahrens herstellen. Bei diesem Verfahren werden die Daten in Sequenzen umgeformt.
Abbildung 20 visualisiert das Verfahren in simpler Form anhand einer beispielhaften Zeitreihe. Es werden zwei Zeitfenster definiert. Das erste ist das Vergangenheitsfenster mit einer gewissen Anzahl an vergangenen Zeitschritten, die als Input für das Modell dienen. Das zweite ist das Vorhersagefenster mit einer gewissen Anzahl von Zeitschritten, die direkt nach dem Vergangenheitsfenster liegen.

Diese zwei Fenster werden über den gesamten Zeitverlauf der Zeitreihe geschoben. Mit jeder Verschiebung der beiden Fenster wird eine Sequenz extrahiert, mit Input und dazugehörenden Output. Am Ende entsteht ein gelabelter Datensatz, der zu seinem Input den entsprechenden Output aufweist.[68]

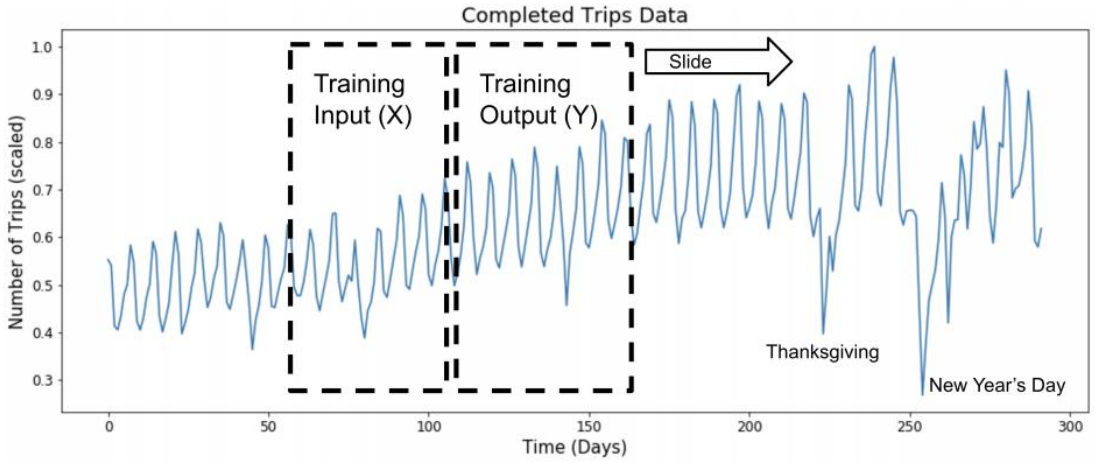


Abbildung 20: Das Sliding Window Verfahren
Quelle: https://machinelearningmastery.com/lstm-model-architecture-for-rare-event-time-series-forecasting/, 22.03.2020

[68] vgl. Brownlee Jason, Deep Learning for Time Series Forecasting, 2019, S. 25–26.

3.4 Train und Test Split der Daten

Die zuvor als Supervised Learning umformulierten Daten werden folglich noch weiter aufgeteilt in Training-, Validation- und Test-Split, verdeutlicht durch die Abbildung 21.

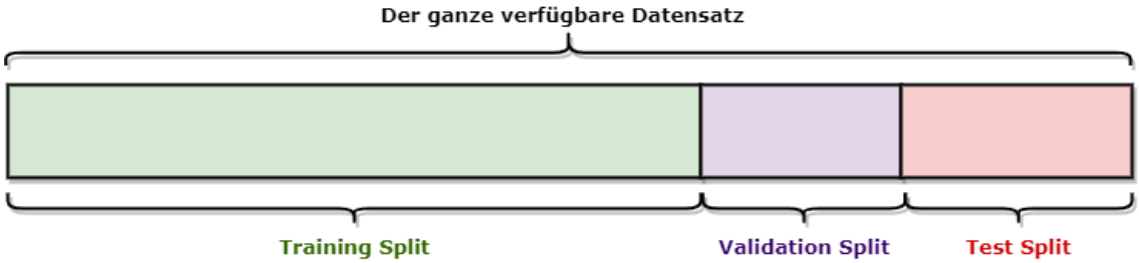


Abbildung 21: Aufteilung der Daten in Training-, Validation- und Test-Split
Quelle: eigene Darstellung

Der Trainings Split oder Satz beinhaltet dabei den größten Anteil der Daten. Dieser wird verwendet, um die Machine Learning Modelle zu trainieren. Für den Validation Split wurden 15% der Daten verwendet. Dieser ist notwendig, um die Genauigkeit der Modelle während des Trainings zu überprüfen und, falls notwendig, ein sogenanntes Early Stopping auszulösen. Das Early Stopping wird im nächsten Abschnitt erklärt. Eine Sequenz wird für den Test Split aufgehoben. Anhand dieser erfolgt dann die endgültige Evaluierung.

Bei den statistischen Modellen ist die Separierung des Validation Sets nicht notwendig. Der Training Split umfasst dabei auch den Validation Split. Der Test Split ist jedoch bei beiden Modellarten identisch.

3.5 Training

Die für das Training benötigte Zeit wird für jedes Modell gemessen, um im nachhinein diese miteinander vergleichen zu können.

Für das Training der Deep Learning Modelle wird die Anzahl der Trainingsepochen definiert. Während einer Epoche wir das Modell allen Sequenzen des Trainingssatzes einmal komplett ausgesetzt. Mittels Backpropagation werden die Parameter nach jeder Epoche nachjustiert. Wenn nach einer bestimmten Anzahl von Epochen keine Verbesserung mithilfe des Validation Sets bestimmt worden ist, wird das zuvor erwähnte Early Stopping ausgelöst und das Training abgebrochen, um keine weitere Zeit und Ressourcen zu vergeuden.

Die statistischen Modelle werden genaugenommen nicht trainiert. Sie *fitten* sich an den Input und approximieren diesen.

3.6 Vorhersage

Bei dem Experiment wird eine mehrschrittige Vorhersage getätigt. Vorhergesagt wird bei jedem der Modelle die Metrik "cpuusage_ps" (CPU usage per second).

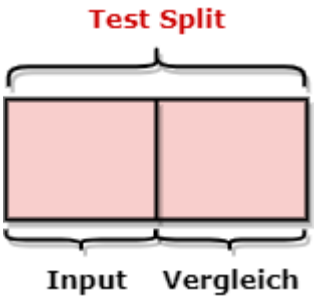


Abbildung 22: Test Split im Fokus
Quelle: eigene Darstellung

Im Endzustand soll das trainierte Modell einen Input von vergangenen Werten der Zeitreihe bekommen, dieses verarbeiten und eine Prognose erstellen. Für die Vorhersage wird der zuvor erstellte Test Split verwendet. Dieser ist nochmals gesondert in Abbildung 22 dargestellt. Aus den Test Split wird ein Ausschnitt mit 96 Schritten bzw. 24 Stunden als historische Daten angenommen und als Input für das Modell verwendet. Basierend auf diesen Daten ergibt das Modell eine Vorhersage für die nächsten vier Tage. Die übriggebliebenen Daten des Testsplit werden aufgehoben und dienen anschließend als Vergleichsdaten. Diese werden dann mit der ausgegebenen Vorhersage der Modelle verglichen und die Abweichung zwischen Ist und Vergleich mithilfe von Metriken evaluiert. Die Verläufe der Vorhersage und des tatsächlichen Verlaufs der Zeitreihe werden in Kapitel 5 bei der Präsentation der Ergebnisse nebeneinander gezeigt.

3.7 Evaluation der Vorhersagegenauigkeit

Für die Evaluation der Vorhersagegenauigkeit wird die vorhergesagte und die tatsächlich auftretende Sequenz (Vergleichsabschnitt des Test Splits) im selben Zeitintervall verwendet und in Metriken eingegeben.

Die Mittlere Quadratische Abweichung (engl. Mean Squared Error = MSE) kann zur Beurteilung der Güte der Vorhersage verwendet werden und ist definiert als:[69]

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

Formel 8: Mittlere quadratische Abweichung (MSE)

Dabei ist $\hat{Y}_i$ der durch das Modell vorhergesagte und Y_i der tatsächlich aufgetretene Wert in diesem Zeitschritt n. Je näher das MSE an dem Wert 0 ist, desto besser ist die Vorhersagegenauigkeit des Modells. Das MSE wird nie negativ.

[69] vgl. Fahrmeir et al., Statistik, 2016, S. 346.

Des Weiteren wird die Wurzel aus der mittleren quadratischen Abweichung (engl. Root Mean Squared Error = RMSE) verwendet. Diese lässt ebenfalls die Beurteilung der Vorhersagegenauigkeit zu und ist wie folgt definiert:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}$$

Formel 9: Wurzel der mittleren quadratischen Abweichung (RMSE)

Das Bestimmtheitsmaß (R^2) wird als weitere Metrik verwendet uns ist definiert als:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2}$$

Formel 10: Bestimmtheitsmaß (R^2)

Bei dieser Metrik dient die Streuung der tatsächlich aufgetretenen Sequenz $\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2$ um das Arithmetische Mittel $\bar{Y}$ als Grundbasis für die Entscheidung der Güte des Modells. Ein simples Modell zur Vorhersage wäre einfach die Annahme des arithmetischen Mittels. Die Streuung der Vorhersage $\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$ wird mit der Streuung der tatsächlichen Sequenz ins Verhältnis gesetzt. Bei der Berechnung der Metrik können Werte zwischen $-\infty$ und 1 auftrerten. Ist das Bestimmtheitsmaß nahe 1, so liegt ein Modell mit einer guten Vorhersagegenauigkeit vor. Ist das Bestimmtheitsmaß des Modells nahe 0, tätigt das Modell eine Vorhersage minimal besser als wenn immer nur das arithmetische Mittel für die Vorhersage angenommen wird. Für Bestimmtheitsmaß < 0 gilt, dass das Modell schlecht ist und sogar eine schlechtere Genauigkeit hat, als die Annahme des arithmetischen Mittels für die Zukunft.[70]

[70] vgl. Fahrmeir et al., Statistik, 2016, S. 149–151.

4 Implementierung

Im folgenden Kapitel soll die Implementierung der Modelle gezeigt werden. Als Programmiersprache wurde Python gewählt, da bereits Vorkenntnisse zu dieser Sprache vorhanden sind und sich Python allgemein großer Beliebtheit in Data Science und Machine Learning Kreisen erfreut.
Zunächst werden kurz die Bibliotheken, die bei der Entwicklung verwendet werden, und im Anschluss die Implementierung der Modelle, vorgestellt.

4.1 Python Bibliotheken im Machine Learning Ökosystem

Alle der hier vorgestellten Python Bibliotheken sind Open Source und können ohne Lizenzkosten verwendet werden.

4.1.1 Statsmodels

Statsmodels ist eine Bibliothek für statistische und ökonomische Analysen. Die Bibliothek beinhaltet Klassen und Funktionen für zahlreiche statistische Modelle, als auch für die Erstellung statistischer Tests sowie für die statistische Datenuntersuchung.[71] Die Bibliothek bietet speziell in dieser Arbeit Hilfsklassen für die Implementierung des Arima Modells.

4.1.2 NumPy

Die NumPy Bibliothek bietet viele Funktionen, die für wissenschaftliche Berechnungen verwendet werden können. Es ermöglicht die effiziente Berechnung mittels Vektoren, Matrizen und mehrdimensionalen Arrays. Die Effizienz ist dadurch zu verdanken, dass optimierter C Code von dem ebenfalls in der Programmiersprache C entwickelte Standard Python Interpreter verwendet werden kann.[72] In dieser Arbeit wird NumPy für die Manipulation der Daten verwendet, um sie beispielsweise in die richtige Form als Input für die Deep Learning Modelle zu bringen.

4.1.3 Pandas

Pandas bietet Funktionen die das einfache und schnelle Arbeiten mit Daten ermöglicht. Die Daten können dabei mithilfe sogenannter Dataframes als Tabellen strukturiert

[71] vgl. Seabold und Perktold, statsmodels: Econometric and statistical modeling with python, 2010, S. 57–58.

[72] vgl. What is NumPy? — NumPy v1.19.dev0 Manual, 2020, o.S.

werden und ähnlich wie relationale Datenbanken manipuliert werden. Mithilfe von Indexen können Aggregation, Umformungen aber auch Slicing und Dicing ausgeübt werden.[73]

Diese Bibliothek kommt bei der Implementierung sehr häufig zur Verwendung, beispielsweise für die Anpassung der Daten auf ein gemeinsames Zeitintervall oder auch bei dem in Absatz 3.2.2 erklärten Feature Engineering.

4.1.4 Keras

Keras wird für die Entwicklung aller Deep Learning Modelle in dieser Arbeit verwendet.

Die Deep Learning Bibliothek wurde als Open Source Projekt durch Francois Chollet gestartet. Ziel der Bibliothek ist es, möglichst schnell erlernbar und einfach benutzbar zu sein. Keras fungiert dabei als eine API mit höherem Abstraktionsgerad und ermöglicht dabei die Verwendung von Tensorflow oder Theano als backend.[74]

4.1.5 Matplotlib und Plotly

Matplotlib und Plotly sind beides Visualisierungsbibliotheken, die es ermöglichen, Liniendiagramme, Histogramme, Streudiagramme und einige weitere Plots mittels Python zu erstellen.[75,76] Matplotlib wurde beispielsweise für die Erstellung der Abbildung 19: Einwöchiger Auszug der Daten verwendet. Mit Plotly werden die in Kapitel 5 vorgestellten Ergebnisse dargestellt.

4.2 Implementierung des Arima Modells

Bei der Implementierung von ARIMA wurden zuerst mit einer Hilfsbibliothek namens pmdarima die in Kapitel 2.3.1 vorgestellten p, d und q Parameter herausgefunden. Da ersichtlich war, dass die eingegebene Zeitreihe „cpuusage_ps" Saisonalität beinhaltet, mussten weitere Parameter geschätzt werden. Dies erfolgte analog mit der Funktion:

```
pmdarima.arima.auto_arima(df['cpuusage_ps'], seasonal=True)
```

Diese Untersuchung ergab die Parameter p = 1, d = 3 und q = 0. Für die Saisonalität wurde P = 1, D = 1, Q = 1 und m = 96 definiert. Der Parameter m wurde manuell

[73] vgl. McKinney, Python for data analysis, 2017, S. 4–5.

[74] vgl. Why use Keras - Keras Documentation, 2019, o.S.

[75] vgl. Matplotlib: Python plotting — Matplotlib 3.2.0 documentation, 2020, o.S.

[76] vgl.Plotly Python Graphing Library, 2020, o.S.

eingegeben, da 96 Schritte je 15 min. Zeitintervall 24 Stunden ergeben (96 x 15 min/60 = 24 h), also eine tägliche Saisonalität.

Durch Verwendung der zuvor vorgestellten Bibliothek Statsmodels und den gerade bestimmten Parametern konnte das saisonale ARIMA Modell mit

```
model = SARIMAX(trainingData['cpuusage_ps'],order=(1, 3, 0),
seasonal_order=(1, 1, 1, 96))
model.fit()
```

definiert und gefittet werden.

4.3 Implementierung des Prophet Modells

Für die Implementierung des Prophet Modells kam die von Facebook eignes erstellte Prophet Library zum Einsatz.

Dafür mussten lediglich die „timestamp“ und „cpuusage_ps“ Spalte der Daten zu „ds“ und „y“ umbenannt und extrahiert werden:

```
trainingData = df.reset_index()[[timestamp,'cpuusage_ps']]
[:-n_history]
trainingData.columns = ["ds","y"]
```

Daraufhin konnte das Modell direkt gefittet werden. Für die wöchentliche Saisonalität wurde die Order der Fourier Reihe mit 50 angegeben.

```
m = Prophet(weekly_seasonality=50, yearly_seasonality=True)
m.fit(trainingData)
```

4.4 Implementierung des MLP Modells

Mit Keras kann ein Deep Learning Modell durch `model = Sequential()` instanziiert werden. Das implementierte MLP Modell beinhaltet insgesamt 6 Schichten. Die Architektur ist in Abbildung 23 auf der nächsten Seite aufgezeigt. Die Input Schicht bekommt ein dreidimensionales Array der Form (1,96,186) übergeben. Dabei handelt sich um eine Sequenz mit 96 Zeitschritten und jeweils 186 Features. Die erste Hidden Schicht mit MLP Neuronen und die Input Schicht wird mit Keras zum instanziierten Modell mit

```
model.add(Dense(186, activation='tanh', input_dim=n_input))
```

hinzugefügt.
Aus der Codezeile kann gesehen werden, dass die Tanh Funktion als Aktivierungsfunktion der Neuronen gewählt worden ist. Dies gilt auch für die anderen

Deep Learning Modelle. Jede weitere Schicht kann ähnlich hinzugefügt werden, wobei zu beachten ist, dass die zweite und dritte Hidden-Schicht jeweils 98 MLP Neuronen beinhalten, während die erste und vierte 186 beinhalten.

Die Output Schicht gibt ein zwei dimensionales Array der Form (1, 384) aus. Genauer gesagt eine Sequenz mit 384 Zeitschritten mit jeweils der vorhergesagten Metrik „cpuusage_ps“.

Wenn dem Modell keine weiteren Schichten hinzugefügt werden sollen, kann noch der Optimizer und die Gütemetrik gewählt und das Model kompiliert werden durch:

```
model.compile(loss='mse', optimizer='adam')
```

In dem Fall wurde dafür das MSE als Gütemetrik für die Trainingsgenauigkeit gewählt und adam als Optimizer.

Jedes Deep Learning Modell wurde maximal 400 Epochen trainiert, wobei ein Early Stopping mit einer Patience von 50 definiert worden ist. Das heißt, verbessert sich das Modell mit seiner Vorhersagegenauigkeit binnen 50 Epochen nicht, so wird das Training frühzeitig abgebrochen. Dies kann mithilfe von Keras wie folgt eingestellt werden:

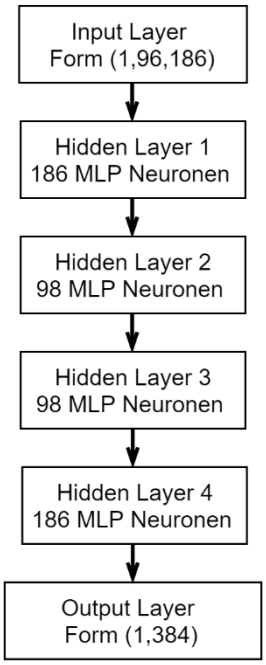


Abbildung 23: Architektur des Implementierten MLP Modells Quelle: eigene Darstellung

```
es= EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
```

Dem Modell wird der mit dem Sliding Window Verfahren, beschrieben in Kapitel 3.3, erstellte Trainingssatz (x_train, y_train) übergeben und es wird definiert, welcher Prozentsatz der Daten als Validation Split verwendet werden soll (hier 0.15 = 15%). Letztendlich kann das Training gestartet werden:

```
model.fit(x_train, y_train, validation_data=0.15, epochs=400,
callbacks=[es])
```

4.5 Implementierung des LSTM Modells

Die Implementierung des LSTM Modells erfolgt ähnlich wie die des MLP.
Das LSTM Modell hat insgesamt 7 Schichten und ist in Abbildung 24 zu sehen. Als Hidden Layer dienen 5 Schichten mit jeweils 75 LSTM Neuronen. Bei Verwendung von Keras kann eine solche Schicht wie folgt einem neu instanziierten Modell, mit bereits vorhandener Input Schicht, hinzugefügt werden:

```
model.add(LSTM(75, activation="tanh",
return_sequences=True))
```

Mit return_sequences = True wird die gesamte Ausgabesequenz an die nächste Schicht übergeben, ansonsten wenn dieser Wert = False wäre, würde nur das letzte Element weitergereicht.

Ähnlich wie bei dem MLP wurde auch in diesem Fall die Tanh Funktion als Aktivierungsfunktion der LSTM Neuronen verwendet. Ebenso wird mit maximal 400 Epochen trainiert, wobei das Early Stopping das Training nach 50 Epochen beenden soll, wenn keine Verbesserung festgestellt wird. Der Optimizer und die Gütemetrik sind ebenfalls identisch wie bei dem MLP Modell.

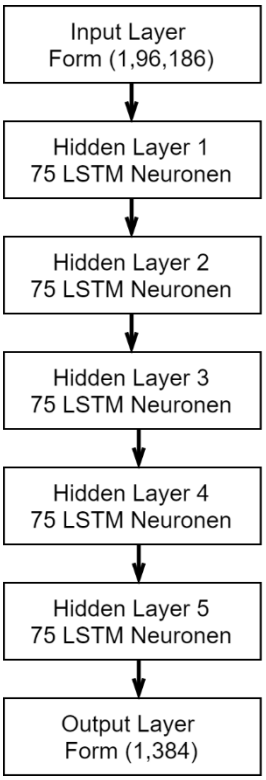


Abbildung 24: Architektur des Implementierten LSTM Modells Quelle: eigene Darstellung

4.6 Implementierung des CNN Modells

Auf der nächsten Seite zeigt die Abbildung 25 die Architektur des Implementierten Convolutional Neuronal Networks mit insgesamt 6 Schichten. Für die ersten drei Hidden Layer wurden eindimensionale Convolutions verwendet, mit 150 Filtern und einer Kernelgröße von 3 für die erste Hidden Schicht. Die Anzahl der Filter wurde für die zweite Hidden Schicht auf 80 mit einer Kernelgröße von 2, bei der dritten Hidden Schicht auf 60 Filter und ebenfalls einer Kernelgröße von 2 reduziert. Eine 1D Convolution kann zu einem Modell wie folgt hinzugefügt werden:

```
model.add(Conv1D(filters=80, kernel_size=2, padding="same",
activation='tanh'))
```

Auch hier wurden die üblichen Parameter für die Trainingsepochen, Early Stopping etc. verwendet.

Zu erwähnen ist, dass kein Pooling Layer verwendet worden ist, da herausgestellt wurde, dass diese die Vorhersagegenauigkeit in diesem Fall verminderten. Im Hidden Layer 4 wurde noch eine LSTM Schicht mit 75 Neuronen eingeführt, da diese sich positiv auswirkte.

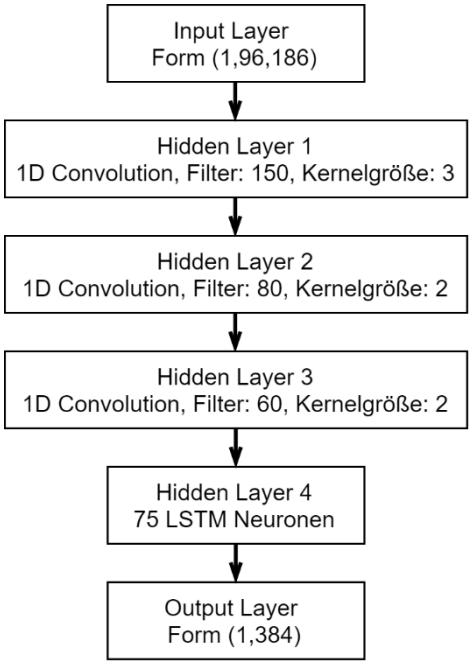


Abbildung 25: Architektur des Implementierten CNN Modells
Quelle: eigene Darstellung

5 Ergebnisse und Evaluation

In diesem Kapitel werden die Ergebnisse des im Abschnitt 3.1 vorgestellten Vergleichsexperiment gezeigt und evaluiert, um letztendlich eine Modelempfehlung für den Anwendungsfall der Vorhersage von Serverauslastungen zu geben. Zu jedem der zuvor vorgestellten Modelle wird ein Graph gezeigt in dem zu sehen ist, wie gut die im Abschnitt 3.6 beschriebene Test-Sequenz vorhergesagt wird.
Der Verlauf der Vorhersage wird dann mit dem wahrhaftigen Verlauf der Zeitreihe in diesen Zeitabschnitt verglichen. Zudem wird zu jedem Modell eine Tabelle mit den in Abschnitt 3.7 vorgestellten Metriken gezeigt.

Alle Modelle wurden mit dem gleichen Rechner trainiert, der mit einer Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz mit 4 Kernen und 8 logischen Prozessoren und 8 GB RAM ausgestattet ist. Die GPU wurde für das Training nicht verwendet. Die gemessene Trainingsdauer der Modelle kann je nach Hardware variieren.

5.1 Ergebnis des ARIMA Modells

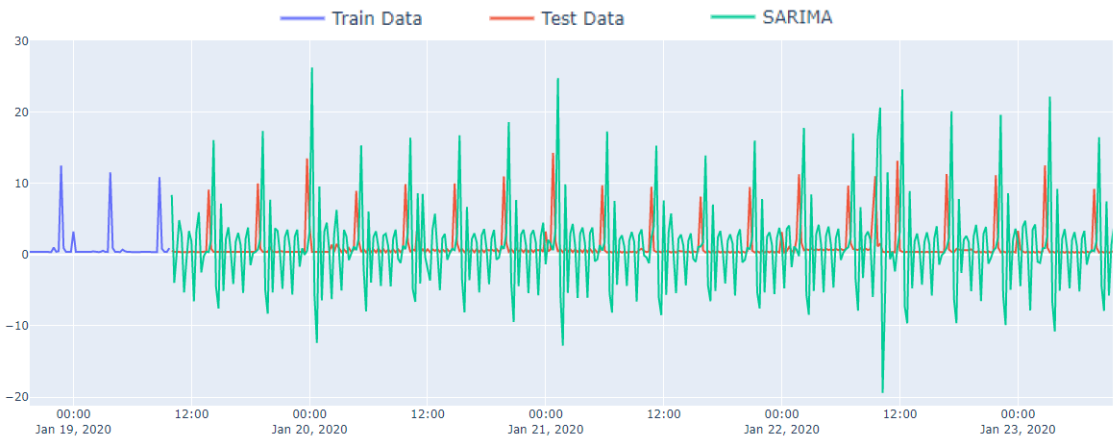


Abbildung 26: Vergleich der SARIMA Vorhersage
Quelle: eigene Darstellung

Abbildung 26 zeigt die Vorhersage (grüner Pfad) des implementierten SARIMA Modells. Die blaue Linie bildet die Trainingsdaten ab, die in Wahrheit noch um drei Wochen in die Vergangenheit verlaufen. Mithilfe der Plotly Bibliothek wurde aber bei jeder Vorhersage, die in diesem Kapitel vorgestellt wird, der Zeitraum zwischen dem 19. Januar 2020 und dem 23. Januar 2020 12:00 Uhr gewählt, um einen besseren Fokus auf die Vorhersage zu legen.
Der rote Pfad bildet die originalen Vergleichsdaten. Ein gutes Modell würde möglichst genau den Verlauf des roten Pfades nachfahren, was hier definitiv nicht der Fall ist. Aus dem Graphen kann bereits gesehen werden, dass das implementierte SARIMA Modell

nicht in der Lage ist, eine gute Vorhersage zu tätigen. Besonders interessant ist, dass negative Werte vorhergesagt werden bei einer Metrik (CPU Auslastung) die an sich nicht negativ sein kann.

Die Performance-Metriken die in Tabelle 1 zusammengefasst sind, bestätigen nur die schlechte Vorhersagegenauigkeit.

Modell	Trainingszeit	MSE	RMSE	R^2
SARIMA	23 min 27 s	41.202	6.419	-6.607

Tabelle 1: Performance-Metriken des SARIMA Modells
Quelle: eigene Tabelle

Mit einem deutlich negativen R^2 sind die Vorhersagen dieses Modells schlechter, als wenn immer nur das arithmetische Mittel als Vorhersage angenommen wurde. Der MSE und der RMSE sind ebenso schlecht. Wünschenswert ist ein Wert unweit von 0.

5.2 Ergebnis des Facebook Prophet Modells

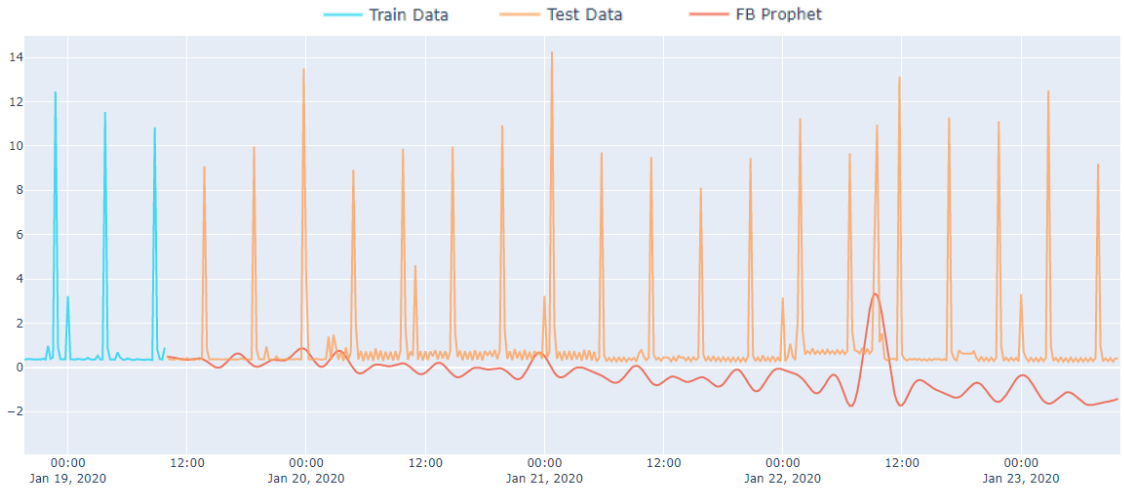


Abbildung 27: Vergleich der Facebook Prophet Vorhersage
Quelle: eigene Darstellung

Die Vorhersage des Prophet Modells dargestellt in Abbildung 27 sieht auf den ersten Blick besser aus als das SARIMA Modell, bringt jedoch auch keine ernstzunehmenden Ergebnisse, um das Modell für eine zuverlässige Prognose zu verwenden. Die Prophet Vorhersage (roter Pfad) schafft es nicht einmal annähernd, den tatsächlichen Verlauf nachzuahmen (orangener Pfad). Auch in diesem Fall werden negative Werte vorhergesagt, die in der Praxis gar nicht vorkommen können.

Die Performance Metriken für dieses Modell in Tabelle 2: Performance-Metriken des Facebook Prophet Modells bestätigen die Verbesserung gegenüber SARIMA, sind jedoch weiterhin weit entfernt von dem gewünschten Zustand. Auffällig ist dennoch die sehr schnelle Trainingszeit des Prophet Modells, welches nur vier Sekunden benötigte.

Modell	Trainingszeit	MSE	RMSE	R^2
Prophet	4 s	7.904	2.811	-0.459

Tabelle 2: Performance-Metriken des Facebook Prophet Modells
Quelle: eigene Tabelle

5.3 Ergebnis des MLP Modells

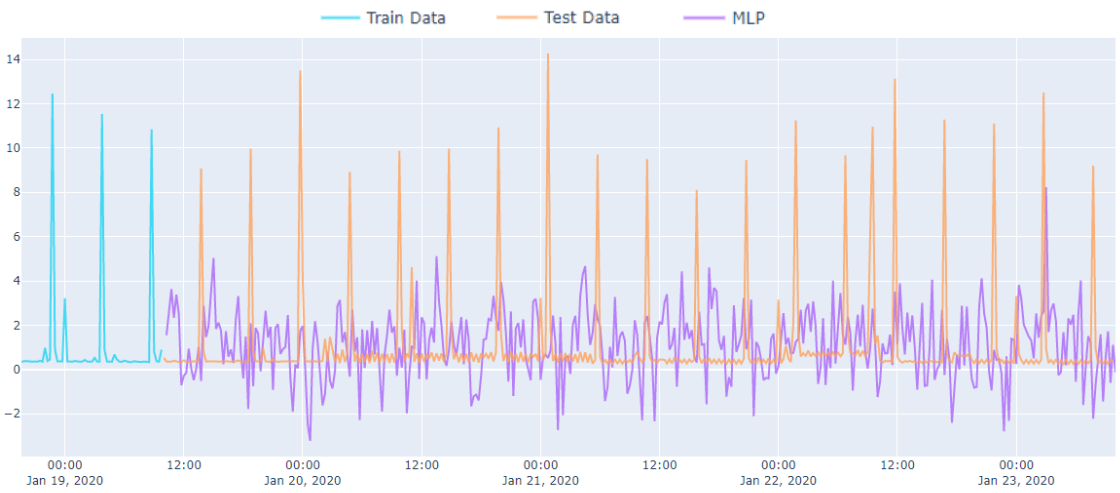


Abbildung 28: Vergleich der MLP Vorhersage
Quelle: eigene Darstellung

Etwas besser, jedoch immer noch nicht zufriedenstellend, sieht es mit der Vorhersage des bisher ersten verwendeten Deep Learning Modells MLP aus. Die Vorhersage in Abbildung 28 zeigt eine deutlich höhere Oszillation als das Prophet Modell. Auch in diesem Fall werden negative Werte vorhergesagt. Für eine präzise Prognose ist auch dieses Modell nicht wirklich brauchbar.

Die Metriken in der Tabelle 3: Performance-Metriken des MLP Modells zeigen immerhin eine Verbesserung gegenüber den zuvor vorgestellten Modellen. Für alle Deep Learning Modelle wurde zusätzlich die Epoche mit aufgenommen, in der das Modell fertig trainiert worden ist. Unteranderem wurde auch der MSE während des Trainings mit dem Trainings Split und der MSE der Validierung mit dem Validation Split aufgezeichnet.

Das Early Stopping wurde in Epoche 51 ausgelöst, welches das Training frühzeitig beendet hat. Negativ fällt die vergleichsweise lange Trainingszeit von knapp über drei Stunden auf, in Anbetracht, dass das Modell nur 51 Epochen von den in der Implementierung festgelegten 400 trainiert wurde.

Modell	Trainingszeit	Epoche	Training MSE	Validierung MSE	MSE	RMSE	R^2
MLP	3 h 2 min 3 s	51	2.371	0.646	6.944	2.635	-0.282

Tabelle 3: Performance-Metriken des MLP Modells
Quelle: eigene Tabelle

Der MSE des Validierungssatzes ist deutlich geringer als der MSE des Trainings, was ein Indiz für eine Überanpassung des Modells für die Abbildung des Inputs auf die zugehörigen Outputs des Validierungssatzes sein könnte.

5.4 Ergebnis des LSTM Modells

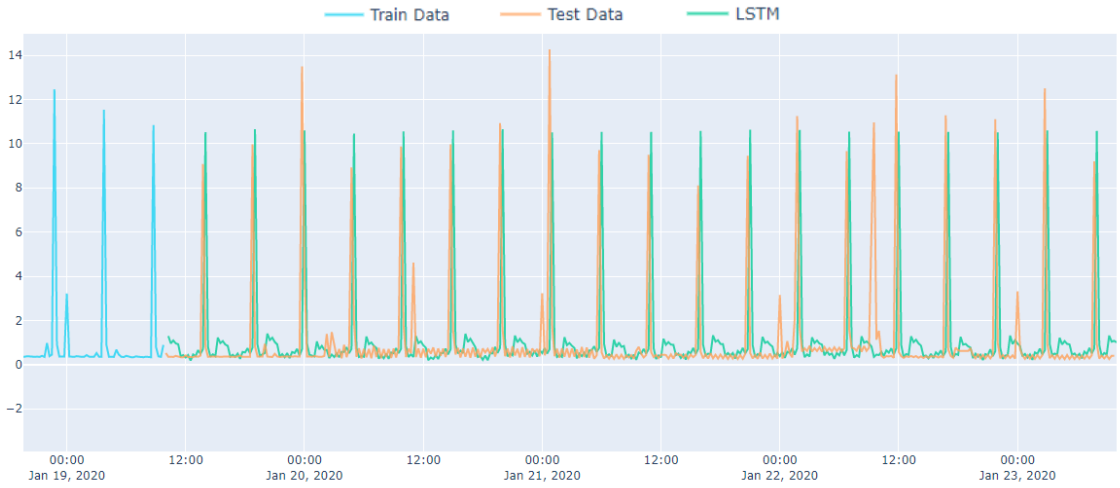


Abbildung 29: Vergleich der LSTM Vorhersage
Quelle: eigene Darstellung

Die Vorhersage mittels LSTM Modell in Abbildung 29 zeigt sich vielversprechend. Großteilig überdeckt die Vorhersage (grüner Pfad) den Verlauf der originalen Testdaten (gelber Pfad). Der Zyklus, in dem ein Anstieg kommt, wird konstant präzise getroffen. Lediglich die Spitzen werden häufig über- oder unterschätzt.

Bisher ist dies das Beste in dieser Arbeit untersuchte Modell, was auch die Metriken in Tabelle 4 bestätigen.

Modell	Trainingszeit	Epoche	Training MSE	Validierung MSE	MSE	RMSE	R^2
LSTM	1 h 0 min 37 s	136	0.824	0.038	0.743	0.862	0.862

Tabelle 4: Performance-Metriken des LSTM Modells
Quelle: eigene Tabelle

Der R^2 ist mit 0.863 nahe dem Wert 1, was für eine gute Vorhersagegenauigkeit spricht. Zur Erinnerung, ein R^2 von 1 würde auftreten, wenn die vom Modell getätigte Vorhersage eins zu eins mit den für das Vergleichsexperiment verwendeten Testdaten übereinstimmt, was praktisch nicht möglich ist. Mit diesem $R^2 > 0$ sagt das Modell die Sequenz deutlich besser voraus, als die einfache Annahme des arithmetischen Mittels für die Zukunft. Positiv ist auch die deutlich geringere Trainingszeit im Vergleich zum vorher vorgestelltem MLP Modell. Das Modell wurde 136 Epochen trainiert bevor das Early Stopping eintrat. Der MSE des Trainings und der Validierung tendieren Richtung 0, was positiv zu bewerten ist.

5.5 Ergebnis des CNN Modells

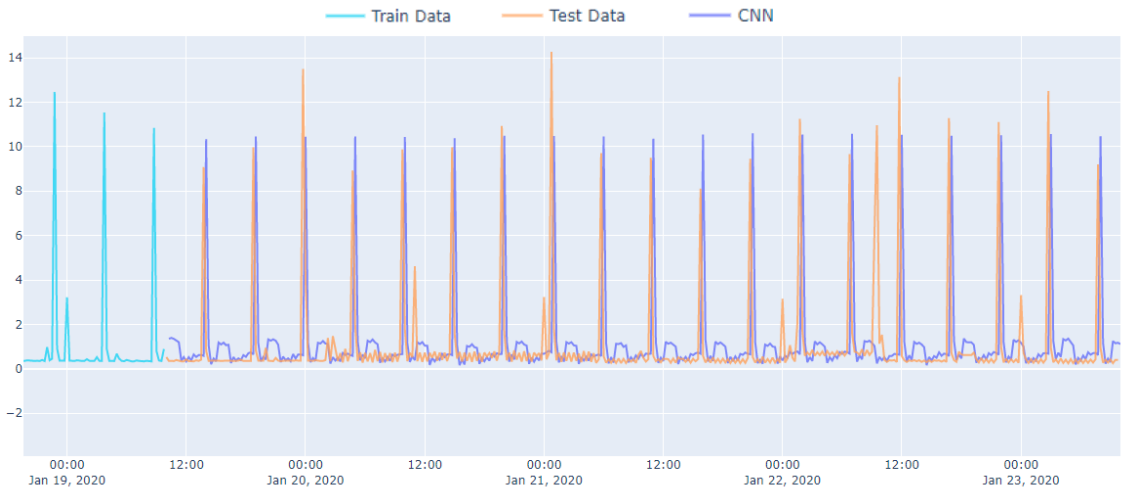


Abbildung 30: Vergleich der CNN Vorhersage
Quelle: eigene Darstellung

Ähnlich gut wie das LSTM Modell ist die Vorhersage des CNN Modells, dargestellt in Abbildung 30. Auch bei diesem Modell wird eine nutzbare Prognose getätigt. Der Pfad der Vorhersage (blau) folgt dicht dem Verlauf des Vergleichspfades (orange).

Die in Tabelle 5 aufgefassten Performance-Metriken zeigen ebenfalls, dass die Vorhersagegenauigkeit des in dieser Untersuchung implementierten CNN Modells präzise ist.

Modell	Trainingszeit	Epoche	Training MSE	Validierung MSE	MSE	RMSE	R^2
CNN	13 min 47 s	101	0.825	0.043	0.784	0.886	0.855

Tabelle 5: Performance-Metriken des CNN Modells
Quelle: eigene Tabelle

Die MSE und RMSE Werte nähern sich dem Wert 0 und auch das R^2 tendiert Richtung 1, was positiv ist. Hervorzuheben ist die schnellste Trainingszeit unter den Deep Learning Modellen von 13 Minuten und 47 Sekunden.

5.6 Gesamtevaluation und Modellempfehlung

Nachdem die Ergebnisse der untersuchten Modelle grafisch sowie metrisch präsentiert worden sind, soll schließlich eine Gesamtevaluation erfolgen und eine Modellempfehlung ausgesprochen werden.

Tabelle 6 zeigt dafür die Performance-Metriken aller bei dieser Untersuchung implementierten Modelle.

Modell	Trainingszeit	MSE	RMSE	R^2	Training MSE	Validierung MSE	Epoche
SARIMA	23 min 27 s	41.202	6.419	-6.607	-	-	-
Prophet	4 s	7.904	2.811	-0.459	-	-	-
MLP	3 h 2 min 3 s	6.944	2.635	-0.282	2.371	0.646	51
LSTM	1 h 0 min 37 s	0.743	0.862	0.862	0.824	0.038	136
CNN	13 min 47 s	0.784	0.886	0.855	0.825	0.043	101

Tabelle 6: Performance-Metriken aller implementierter Modelle
Quelle: eigene Tabelle

Das saisonale ARIMA enttäuscht als das schlechteste untersuchte Modell mit der geringsten Vorhersagegenauigkeit. Eine Erklärung für die geringe Präzision des Modells ist zum einem, dass nur eine univariate Zeitreihe als Input dienen kann und diese mit rund 2600 Zeitschritten zu lang ist. Die Vermutung ist, dass das Modell für Zeitreihen,

die weniger komplex und eine geringere Anzahl an Zeitschritten beinhalten, besser geeignet ist.

Auch das Prophet Modell erweist sich aus den gleichen Gründen als enttäuschend, bietet aber eine einfache Implementierung und schnelle Trainingsdauer.

Das erste Deep Learning Modell Multilayer Perceptron kann überraschenderweise auch nicht überzeugen und benötigte die längste Trainingszeit. Eine Erklärung für die schlechte Vorhersagegenauigkeit ist die einfache Architektur der MLP Neuronen, die keine spezifische Auslegung für Sequenzen oder Zeitreihen haben, wie die LSTM Neuronen.

Das Ergebnis des LSTM Modells belegt, dass die Vorhersage von Zeitreihen im Anwendungsfall von Serverauslastungen möglich ist. Von allen in dieser Arbeit vorgestellten Modellen hat dieses Modell die besten Performance-Metriken aufzuweisen. Das bezeugt, dass die auf Sequenzen und Zeitreihen ausgelegte Architektur der Neuronen vorgestellt im Kapitel 2.4.3 auch tatsächlich gut funktioniert.

Allerdings wird für den Anwendungsfall der Vorhersage von Serverauslastungen konkret das vorgestellte CNN Modell empfohlen. Es bietet eine minimal schlechtere Vorhersagegenauigkeit gegenüber dem LSTM, bringt dafür ein um vielfache schnellere Trainingszeit. Dieser Faktor ist vor allem ausschlagegebend, wenn betrachtet wird, dass mit der im Kapitel 1.1 vorgestellten Anwendung ein ganzes Datenzentrum abgebildet werden soll, welches aus mehreren Clustern besteht, mit wiederum tausenden von Serverinstanzen mit jeweils um die hundert mitgezeichneten Metriken.
Daraus wird ersichtlich, dass bei solchen Mengen eine kürzere Trainingszeit deutlich zum Vorteil wird und deshalb vorgezogen werden sollte.
In der Untersuchung wurde nur die Vorhersage eines Messwertes analysiert, aber in Hinblick auf die zahlreichen Metriken die in der Anwendung vorhergesagt werden sollen, rechnet sich die kürzere Trainingszeit deutlich. Anders betrachtet können mit dem CNN Modell längere Zeitreihen als Input herangezogen und längere Zeithorizonte mit der gleichen Trainingsdauer vorhergesagt werden, als bei dem LSTM bei vergleichsweise geringerem Input und Vorhersagehorizont.

6 Zusammenfassung, Fazit und Ausblick

In dieser Arbeit wurde ein Einblick in das Machine Learning gegeben und untersucht, welche Machine Learning Modelle für die Vorhersage von Zeitreihen eingesetzt werden können. Dies wurde mithilfe einer Literaturrecherche vollbracht. Als Ergebnis wurden die statistischen Modelle ARIMA und Facebook Prophet, sowie Deep Learning Modelle, Multilayer Perceptron, Long Short Term Memory und Convolutional Neural Network vorgestellt. Die Architektur und Funktionsweise jedes dieser Modelle wurde detailliert gezeigt und beschrieben.

Es wurde ein Konzept für ein Vergleichsexperiment vorgestellt, bei dem die Modelle auf ihre Vorhersagegenauigkeit analysiert und evaluiert werden sollen, um herauszufinden, welches von ihnen am besten für die Vorhersage von Serverauslastungen geeignet ist.

Um das Vergleichsexperiment zu vollziehen wurden die vorgestellten Modelle implementiert. Danach wurde das konzipierte Vergleichsexperiment an den von der Bundesagentur für Arbeit zur Verfügung gestellten Serverauslastungsdaten durchgeführt.

Aus den Ergebnissen des Vergleichsexperiments stellte sich heraus, dass Serverauslastungen durch Machine Learning zuverlässig, mit dem in der Arbeit vorgestellten Long Short Term Memory oder dem Convolutional Neural Network Modell, vier Tage in die Zukunft vorhergesagt werden können. Die zwei statistischen Modelle Arima und Prophet und das Deep Learning Modell Multilayer Perceptron stellten sich als ungeeignet heraus.

Für diesen konkreten Anwendungsfall wird das Convolutional Neural Network empfohlen, aufgrund von einer schnellen Trainingszeit und guter Vorhersagegenauigkeit mit einem Bestimmtheitsmaß von 0.855.

Für die Zukunft muss ein geeignetes Konzept für die Integration des empfohlenen Modells in die im Abschnitt 1.1 gezeigte Monitoring Anwendung erstellt werden. Dabei muss überlegt werden wie eine geeignete Architektur aussehen soll, die es ermöglicht, eine Vorhersage für mehrere tausende von Servern skalierbar tätigen zu können.

Literaturverzeichnis

Bontempi, Gianluca (2008): Long term time series prediction with multi-input multi-output local learning. In: *Proceedings of the 2nd European Symposium on Time Series Prediction (TSP), ESTSP08*.

Bontempi, Gianluca; Ben Taieb, Souhaib; Le Borgne, Yann-Aël (2013): Machine Learning Strategies for Time Series Forecasting. In: Marie-Aude Aufaure und Esteban Zimányi (Hg.): Business Intelligence. Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures, Bd. 138. Berlin/Heidelberg: Springer Berlin Heidelberg.

Brockwell, Peter J.; Davis, Richard A. (2016): Introduction to Time Series and Forecasting. Cham: Springer International Publishing.

Brownlee, Jason (2019): Deep Learning for Time Series Forecasting. Predict the Future with MLPs, CNNs ans LSTMs in Python.

Chircu, Alina; Sultanow, Eldar; Baum, David; Koch, Christian; Seßler, Matthias (2019): Visualization and Machine Learning for Data Center Management. In: INFORMATIK 2019: 50 Jahre Gesellschaft für Informatik - Informatik für Gesellschaft (Workshop-Beiträge). Bonn, S. 23–35.

Chollet, François (2018): Deep learning with Python. Shelter Island, NY: Manning.

Contreras, J.; Espinola, R.; Nogales, F. J.; Conejo, A. J. (2003): ARIMA models to predict next-day electricity prices. In: *IEEE Trans. Power Syst.* 18 (3), S. 1014–1020. DOI: 10.1109/TPWRS.2002.804943.

Fahrmeir, Ludwig; Heumann, Christian; Künstler, Rita; Pigeot, Iris; Tutz, Gerhard (2016): Statistik. Der Weg zur Datenanalyse. 8., überarbeitete und ergänzte Auflage. Berlin, Heidelberg: Springer Spektrum (Springer-Lehrbuch). Online verfügbar unter http://dx.doi.org/10.1007/978-3-662-50372-0.

Fayyad, Usama; Piatetsky-Shapiro, Gregory; Smyth, Padhraic (1996): From data mining to knowledge discovery in databases. In: *AI magazine* 17 (3), S. 37.

Geron, Aurelien (2019): Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. : Concepts, Tools, and Techniques to Build Intelligent Systems. 2. Aufl.: O'Reilly Media, Incorporated.

Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016): Deep learning. Cambridge, Massachusetts, London, England: MIT press. Online verfügbar unter http://www.deeplearningbook.org/.

Heinrich, Kai; Fleißner, Vera (2018): Deep Intelligent Systems for Time Series Prediction. Champion or Lame Duck? Evidence from Crude Oil Price Prediction. In: AMCIS 2018. New Orleans.

Hupez, Martin; Toubeau, Jean-François; Grève, Zacharie de; Vallée, François (2017): SARMA Time Series for Microscopic Electrical Load Modeling. In: Ignacio Rojas, Héctor Pomares und Olga Valenzuela (Hg.): Advances in Time Series Analysis and Forecasting. Selected Contributions from ITISE 2016, Bd. 50. Cham: Springer International Publishing (Contributions to Statistics), S. 133–145.

Koller, Wolfgang (2014): Prognose makroökonomischer Zeitreihen. Ein Vergleich linearer Modelle mit neuronalen Netzen. Forschungsergebnisse der Wirtschaftsuniversität Wien. Frankfurt a. M.: Peter Lang International Academics Publishers (63).

Konar, Amit; Bhattacharya, Diptendu (2017): Time-Series Prediction and Applications. A Machine Intelligence Approach. Cham: Springer International Publishing (Intelligent Systems Reference Library, 127).

Louridas, Panos; Ebert, Christof (2016): Machine Learning. In: *IEEE Softw.* 33 (5), S. 110–115. DOI: 10.1109/MS.2016.114.

Matplotlib: Python plotting — Matplotlib 3.2.0 documentation (2020). Online verfügbar unter https://matplotlib.org/, zuletzt aktualisiert am 05.03.2020, zuletzt geprüft am 18.03.2020.

Mayring, Philipp; Fenzl, Thomas (2019): Qualitative Inhaltsanalyse. In: Nina Baur und Jörg Blasius (Hg.): Handbuch Methoden der empirischen Sozialforschung. Wiesbaden: Springer Fachmedien Wiesbaden, S. 633–648. Online verfügbar unter https://doi.org/10.1007/978-3-658-21308-4_42.

McKinney, Wes (2017): Python for data analysis. Data wrangling with Pandas, NumPy, and IPython. Second edition. Sebastopol, CA: O'Reilly Media. Online verfügbar unter http://proquest.tech.safaribooksonline.de/9781491957653.

Montgomery, Douglas C.; Jennings, Cheryl L.; Kulahci, Murat (2015): Introduction to Time Series Analysis and Forecasting. Hoboken, New Jersey.: Wiley (Wiley Series in Probability and Statistics).

Nguyen, Michael (2018): Illustrated Guide to LSTM’s and GRU’s: A step by step explanation. towards data science. Online verfügbar unter https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, zuletzt aktualisiert am 24.09.2018, zuletzt geprüft am 03.03.2020.

Plotly Python Graphing Library (2020). Online verfügbar unter https://plot.ly/python/, zuletzt aktualisiert am 16.03.2020, zuletzt geprüft am 18.03.2020.

Sarkar, Dipanjan; Bali, Raghav; Sharma, Tushar (2018): Practical Machine Learning with Python. A Problem-Solver's Guide to Building Real-World Intelligent Systems. Berkeley, CA: Apress.

Seabold, Skipper; Perktold, Josef (2010): statsmodels: Econometric and statistical modeling with python. In: 9th Python in Science Conference.

Selvin, Sreelekshmy; Vinayakumar, R.; Gopalakrishnan, E. A.; Menon, Vijay Krishna; Soman, K. P. (2017): Stock price prediction using LSTM, RNN and CNN-sliding window model. In: nternational Conference on Advances in Computing, Communications and Informatics (ICACCI). Udupi, S. 1643–1647.

Sibbertsen, Philipp (2011): Zeitreihenanalyse. In: Heinz Grohmann, Walter Krämer und Almut Steger (Hg.): Statistik in Deutschland. 100 Jahre Deutsche Statistische Gesellschaft. Heidelberg: Springer, S. 165–176.

Sorjamaa, Antti; Hao, Jin; Reyhani, Nima; Ji, Yongnan; Lendasse, Amaury (2007): Methodology for long-term prediction of time series. In: *Neurocomputing* 70 (16-18), S. 2861–2869. DOI: 10.1016/j.neucom.2006.06.015.

Taylor, Sean J.; Letham, Benjamin (2017): Forecasting at scale.

Vogel, Jürgen (2015): Prognose von Zeitreihen. Eine Einführung für Wirtschaftswissenschaftler. Wiesbaden: Springer Gabler.

Vom Brocke, Jan; Simons, Alexander; Niehaves, Bjoern; Niehaves, Bjorn; Reimer, Kai; Plattfaut, Ralf; Cleven, Anne (2009): RECONSTRUCTING THE GIANT: ON THE IMPORTANCE OF RIGOUR IN DOCUMENTING THE LITERATURE SEARCH.

Webster, Jane; Watson, Richard T. (2002): Analyzing the past to prepare for the future: Writing a literature review. In: *MIS Quarterly* 26 (2).

What is NumPy? — NumPy v1.19.dev0 Manual (2020). Online verfügbar unter https://numpy.org/devdocs/user/whatisnumpy.html, zuletzt aktualisiert am 17.03.2020, zuletzt geprüft am 17.03.2020.

Why use Keras - Keras Documentation (2019). Online verfügbar unter https://keras.io/why-use-keras/, zuletzt aktualisiert am 19.09.2019, zuletzt geprüft am 17.03.2020.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Hilfsmittel benutzt und wörtliche sowie sinngemäße Zitate als solche gekennzeichnet habe.

Hawangen, den 22.03.2020

Unterschrift