Multivariate Zeitreihenprognose mittels neuronaler Netze

Ein Vergleich verschiedener Netzstrukturen

Kurzfassung

Der kontinuierliche Fortschritt auf dem Forschungsgebiet künstliche Intelligenz, speziell im Bereich der künstlichen neuronalen Netze, steigert deren Relevanz in Unternehmen. Kombiniert mit der zunehmenden Verfügbarkeit von Daten jeglicher Art, bieten diese Technologien vielversprechende Chancen. Hierbei können neuronale Netze u. a. für die Analyse und Prognose wirtschaftlicher Zeitreihen genutzt werden. Deren effizienter Einsatz in der Zeitreihenprognose erfordert jedoch einige Vorbereitungsschritte. Neben der Wahl einer guten Modellstruktur ist insbesondere die Vorbereitung der zu analysierenden Daten ein entscheidender Faktor für den Erfolg. Am Beispiel eines Datensatzes eines Fahrradverleihsystems werden die Schritte für eine erfolgreiche Definition von Netzstrukturen, der richtigen Vorbereitung des Datensatzes und der korrekten Festlegung von Untersuchungsmerkmalen im Datensatz anhand von verschiedenen Versuchen aufgezeigt. Die einzelnen Versuche vergleichen jeweils die Performance von unterschiedlichen Netzstrukturen und über die Versuche hinweg wird den Datensatz auf verschiedene Weisen modifiziert. Die Auswertung der Versuche ermöglicht das Treffen einer Aussage über die Eignung der genutzten Netzstrukturen, sowie der bestmöglichen Zusammenstellung des Datensatzes.

Abstract

The continuous advance in the research field of artificial intelligence, especially in the area of artificial neural networks, leads to an increased relevance for enterprises. In combination with increasing availability of differing data sets, these technologies enable promising chances. Hereby, neural networks can be used to analyze and predict economic time series. The efficient use of neural networks for time series predictions demands several preparation steps. Beside the choice of the right model architecture, the prearrangement of the data set is a crucial criterion for success. On the example of a data set from a bike sharing system, the steps for a successful definition of a neural network, the accurate preparation of the dataset and the precise determination of examination attributes are illustrated by different experiments. Within the single experiments, different network structures are compared to each other and a comparison of the different experiments indicates the influence of altered data sets. At last, the experiments are evaluated and a proposition about the adequacy of different network architectures, as well as the aggregation of the data set, is consummated.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

FC	fully connected
FNN	Feedforward neural network
GRU	Gated recurrent Unit
KNN	künstliche neuronale Netze
LSTM	long short-term Memory
RNN	recurrent neural network
tanh	tangens hyperbolicus
ReLU	Rectified Linear Unit
BGD	Batch Gradient Descent
SGD	Stochastic Gradient Descent
ÖPVN	Öffentlicher Personen Nahverkehr

1 Einleitung

Die Prognose von Zeitreihen spielt in der Wirtschaft eine immer entscheidendere Rolle. In den verschiedensten unternehmerischen Bereichen führt dies zu einem signifikanten Vorteil gegenüber Mitbewerbern. Hierbei können bspw. Absatzprognosen dazu beitragen eigene Ressourcen effizienter zu nutzen. Marktprognosen dienen der Optimierung der eigenen Geschäftsmodelle, während die Antizipation von Kundenverhalten die Erschließung neuer Märkte eröffnet. Neben immer schneller werdenden Rechenleistungen, der stetig wachsenden Verfügbarkeit von Speicherkapazitäten und einer weltweiten Vernetzung, hat insbesondere die gestiegene Bereitschaft von Kunden, Konkurrenten oder Kooperationspartnern Daten zur Verfügung zu stellen, dazu geführt, dass diese in wirtschaftlichen Erfolg transformiert werden. Einen entscheidenden Beitrag leisten hierbei Methoden der künstlichen Intelligenz, welche dazu fähig sind, eine schnelle und akkurate Analyse von verschiedenen Daten zu realisieren. Insbesondere die Verwendung neuronaler Netze hat in den vergangenen Jahren zu einem stetigen Fortschritt im Bereich der Datenverarbeitung und -analyse geführt. Diese zeichnen sich neben der Anwendung für verschiedene Problemstellungen, wie bspw. der Analyse von Zeitreihen, der Erkennung und Klassifikation von visuellen Daten oder der Verarbeitung von menschlicher Sprache, vor allem durch ihre Lern- und Generalisierungsfähigkeit, aus. Hierbei werden neuronale Netze für unterschiedliche Problemstellungen mit verschiedenen Lernverfahren trainiert und ermöglichen dabei eine zuverlässige Performance in realen Anwendungsdatensätzen.

Trotz der vielversprechenden Möglichkeiten begründen die Netze durch falsches Training, oder der falschen Definition der Problemstellung, wirtschaftliche Risiken. Neben der schwierigen Nachvollziehbarkeit der Modellprognosen, sind Überbeziehungsweise Unteranpassungen des Lernproblems eine häufige Ursache. Das Ziel dieser Arbeit besteht in der Ermittlung eines Vorgehens zur effizienten Nutzung neuronaler Netze in der Zeitreihenprognose. Hierfür werden unterschiedliche Netzstrukturen zur Prognose von unterschiedlichen Aufgabenstellungen am Beispiel eines Datensatzes des Fahrradverleihsystems VRNnextbike in Kaiserslautern

untersucht. Zusätzlich erfolgt eine Illustration von Methoden, wie Datensätze für das Training mit neuronalen Netzen erfolgreich vorbereitet werden können und inwiefern die Definition der Modellarchitektur einen Einfluss auf die Güte der Modellperformance nehmen kann.

Die Struktur dieser Arbeit ist folgendermaßen aufgebaut. Das anschließende Kapitel beschreibt zunächst die grundsätzliche Funktionsweise von neuronalen Netzen, indem neben dem Kernelement neuronaler Netze, dem künstlichen Neuron, der Aufbau von Netzstrukturen, das Vorgehen beim Training von Netzen, sowie die Herausforderungen rekurrenter Netze, die zeitlichen Abhängigkeiten der Zeitreihen zu modellieren. Insbesondere die Verwendung von *long short-term memory* Netzwerken, sowie *Gated recurrent Units* bietet Möglichkeiten solche Problemstellungen zu umgehen. Das dritte Kapitel beschreibt den Aufbau, die Struktur und die Parameter, welche das Lernverfahren innerhalb der Netze definieren. Zudem wird das Trainingsverfahren spezifiziert, sowie Möglichkeiten zur Evaluation der Modelle betrachtet. Die Vorstellung des Datensatzes, bestehend aus Zeit-, Wetter- und Fahrraddaten, sowie vorbereitende Maßnahmen für ein erfolgreiches Training der Netze erfolgt im nachfolgenden Kapitel. Neben der Vervollständigung und Reparatur des Datensatzes erfolgt dessen Transformation in ein für das Netz lesbares Format. Das Bilden eines Test-, Validierungs- und Trainingsdatensatzes schließt die Bearbeitung des Datensatzes ab. Anschließend wird dieser in verschiedenen Versuchen verwendet. Der erste Versuch befasst sich mit der Prognose von Fahrrädern an den verschiedenen Stationen des Fahrradverleihsystems. Die Verfügbarkeit an Fahrrädern ist auf 11 Klassen beschränkt und alle Merkmale des Datensatzes sind ins Training der Netze einbezogen. Für den zweiten Versuch wird der Datensatz stark reduziert, sodass die Netze mit 23 anstelle von 45 Merkmalen trainiert werden. Der folgenden Versuch untersucht die Erhöhung der Anzahl an Klassen auf 15. Der letzte Versuch befasst sich mit der Prognose über mehrere Zeitschritte in die Zukunft. Abschließend werden die Versuchsergebnisse der verschiedenen Netze miteinander verglichen und eine Aussage über die Eignung der Netze für die in dieser Arbeit betrachteten Problemstellungen getroffen.

2 Neuronale Netze

Neuronale Netze sind Lernalgorithmen, die dem Gebiet des maschinellen Lernens angehören und somit ein Teilgebiet der künstlichen Intelligenz bilden. Der Begriff maschinelles Lernen beschreibt Methoden, die, nach Beendigung eines Lernprozesses, Zusammenhänge in Datensätzen erkennen und auf der Basis dieser Fähigkeit, Vorhersagen treffen. Selbstlernende Algorithmen ermöglichen die Erkennung und Analyse zukunftsrelevanter Rückschlüsse in bestehenden Daten. (Welsch et al. (2018)) Die Lernverfahren des maschinellen Lernens beschreiben Computerprogramme, die aus der Erfahrung E lernen, bei der Ausführung bestimmter Aufgaben T unter Verwendung eines Leistungsmaßes P. Das Ziel ist eine Leistung, die in den Aufgaben T erbracht, anhand des Leistungsmaßes P gemessen und durch die Erfahrung E verbessert wird. (Mitchell (1997))

2.1 Grundlagen neuronaler Netze

Künstliche neuronale Netze (KNN) sind informationsverarbeitende Systeme, die in Computern simuliert werden. Ihre Struktur und Funktionsweise ist dem neuronen Netz in menschlichen oder tierischen Gehirnen nachempfunden (Kruse et al. (2011)). Obwohl sich Netze hinsichtlich Anwendungsgebiet und Aufbau unterscheiden, bleibt die Funktionsweise der KNN immer gleich (Abbildung 1). Das Netz erhält eine Menge an Eingaben, deren spezifische Eigenschaften ein M dimensionalen Eingabevektor X_M beschreibt, und wandelt diese in eine Menge an Ausgaben um, die wiederum ein N dimensionalen Ausgabevektor Y_N darstellt. (Scherer (2013))

Die Kernelemente des KNN bilden einzelne künstliche Neuronen (Abbildung 2). Diese erhalten Eingangssignale $X(t)$, welche ein Gewichtungsfaktor W_i modifiziert. Der Gewichtungsfaktor beschreibt die Stärke der Verbindung und unterliegt Veränderungen während des Lernprozesses (Traeger et al. (2003)). Nach der Gewichtung der Eingangssignale summiert eine Übertragungsfunktion, wie

Abbildung 1 – Prinzip der Datenverarbeitung neuronaler Netze

bspw. $u(t) = \sum_{i=1}^{n} X_i W_i$, diese auf. Unter Verwendung einer Aktivierungfunktion bestimmt der kumulierte Eingangswert den Aktivierungszustand des Neurons. Sofern der Wert der Funktion $u(t)$ einen Schwellenwert δ überschreitet, ist das Neuron aktiviert und bekommt den Ausgabewert $Y(t)$ zugeordnet. Falls δ nicht überschritten wird, bleibt das Neuron im inaktiven Zustand und gibt ein abweichendes Ausgangssignal aus. (Styczynski et al. (2017))

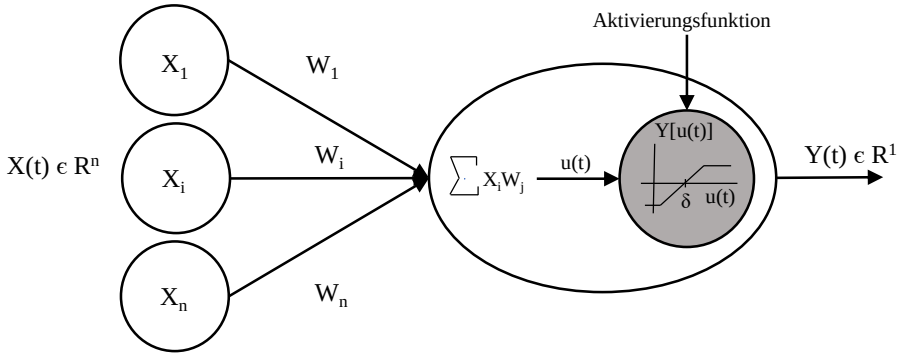

Abbildung 2 – Aufbau eines einzelnen Neurons, nach Styczynski et al. (2017)

2.1.1 Struktur neuronaler Netze

KNN bestehen aus einer Vielzahl an Neuronen, sowie deren Verbindungen untereinander, die in drei Arten von Schichten (*eng. layer*) vorliegen (Abbildung 3). Jede einzelne Schicht enthält beliebig viele Neuronen. Während es jeweils nur eine Eingabe- und Ausgabeschicht gibt, besitzt ein KNN beliebig viele versteckte Schichten. Entsprechend der Schichten werden die jeweiligen Neuronen benannt. Eingabe-Neuronen erhalten externe Reize oder Signale, die verarbeitet werden sollen. Die Ausgabe-Neuronen geben den durch das Netz bearbeiteten Input aus und die Neuronen der versteckten Schichten empfangen, verarbeiten und leiten interne Impulse weiter. (Baetge and Henning (2008))

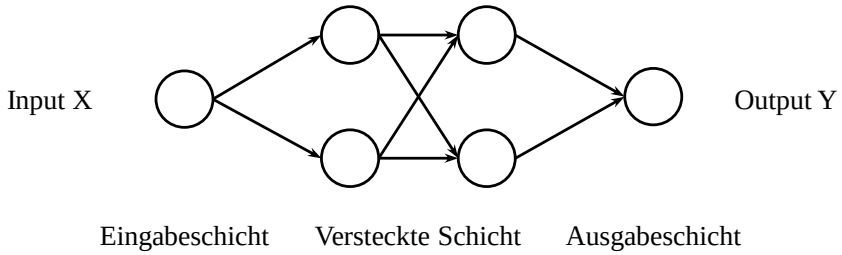

Abbildung 3 – Schematischer Aufbau eines KNN mit 2 versteckten Schichten

Ein entscheidendes Merkmal bei der Charakterisierung von Netzstrukturen ist die Rückkopplung. Netze ohne Rückkopplung werden als vorwärts gerichtete Netze (*eng. Feedforward Network*, FFN) bezeichnet und Netze bei denen die Rückkopplung möglich ist, heißen rekurrente Netze (*eng. Recurrent Network*, RNN). RNN ermöglichen die mehrfache Passierung von Neuronen durch ein Signal, indem die Netzstruktur über rückwärtsgerichtete Verbindungen verfügt (Frankenberger (2007)). Die Rückkopplung ermöglicht Netzen dynamisches Verhalten zu antizipieren und Signale zeitlich verzögert zu betrachten. RNN können verschiedenen Arten der Rückkopplungen besitzen (Abbildung 4). Bei einer direkten Rückkopplung ist ein Neuron durch eine Schleife mit sich selbst verbunden, wodurch sich der Aktivierungszustand des Neurons verstärkt oder mindert. Verbindungen zur vorherigen Schicht liegen bei der indirekten Rückkopplung vor. Hierdurch beeinflusst ein Neuron j die eigene Aktivierung. Die letzte Rückkopplungsart ist die laterale Rückkopplung. Bei dieser sind Neuronen mit benachbarten Neuronen der gleichen Schicht verknüpft, mit einer Verstärkung oder Hemmung einzelner Neuronen als Ergebnis. (Strecker (1997))

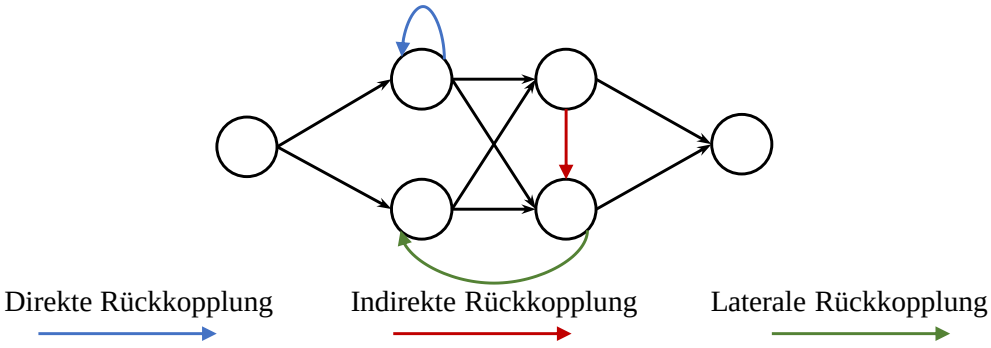

Abbildung 4 – Schematischer Aufbau eines RNN, nach Dörn (2016)

2.1.2 Lernverhalten in neuronalen Netzen

Der Erstellungsprozess von KNN umfasst zwei Phasen, das Definieren und das Ausführen (Abbildungen 5,6). In der ersten Phase, dem Definieren, wird ein Berechnungsgraph konstruiert. Dabei spezifiziert ein Model die Anzahl der Neuronen in den einzelnen Schichten, die Verbindungen zwischen den Neuronen, die Gewichtungsfaktoren und die Aktivierungsfunktionen der einzelnen Neuronen. Nach der Definition des Modells wird der Berechnungsgraph erstellt und durch die Modellparameter festgelegt. Dadurch ist das Netz in der Lage Informationen vorwärtsgerichtet, von der Eingabeschicht hin zur Ausgabeschicht zu berechnen. Zusätzlich definiert eine Gradientenfunktion die Basis für das Training der Netze in der zweiten Phase, welche in der ersten Phase jedoch nur durch eine automatische Gradientenfunktionalität dargestellt ist. (Tokui et al. (2015))

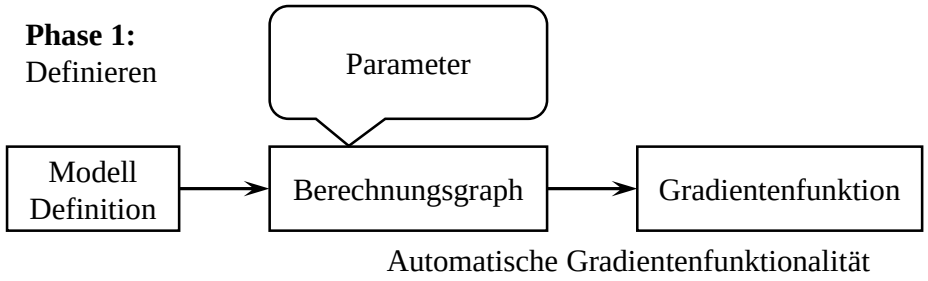

Abbildung 5 – **Phase 1:** Definieren, nach Tokui et al. (2015)

In der zweiten Phase (Abbildung 6) erfolgt das Training des Netzes für ein gegebenes Problem auf der Basis von Trainingsdaten (Tokui et al. (2015)). Ermöglicht wird dies durch den Einsatz unterschiedlicher Lernverfahren, wie bspw. der *Backpropagation Algorithmus*. Bei diesem sind die Parameter des Netzes zu Beginn des Trainings zufällig gewählt, wodurch bei der Ausgabe zunächst große Fehler, zwischen den Vorhersagen des Netzes und den tatsächlichen Ergebnissen, auftreten. Die Beziehung zwischen den Parametern und den Fehlern beschreibt eine Fehlerfunktion. Basierend auf der gewählten Gradientenfunktion wird durch die Anpassung der Netzparameter die Fehlerfunktion minimiert. (Rojas (2001))

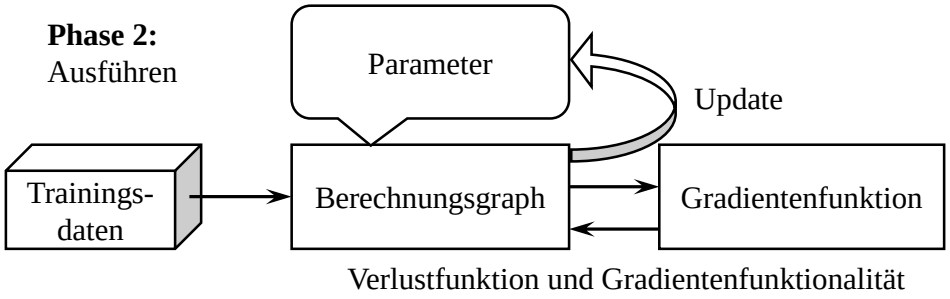

Abbildung 6 – **Phase 2:** Ausführen, nach Tokui et al. (2015)

Formal ist ein KNN ein Netzwerk mit M Input und N Output Neuronen, sowie L versteckten Schichten. Zum Training werden P Paare (X, T) verwendet, wobei X ein M dimensionaler Inputvektor und T ein N dimensionaler Outputvektor mit bekanntem Output sind. Eine Berechnungsfunktion wie bspw. $E = \frac{1}{2} + \sum_{i=1}^{P} || X_i - T_i^2 ||$ beschreibt das Lernproblem. Der Lernvorgang des Netzes umfasst drei Schritte (Abbildung 7). Zunächst rechnet das Netz vorwärts und berechnet für den gegebenen Input X den N dimensionalen Output Y. Der zweite Schritt vergleicht die berechneten Ergebnissen Y mit den tatsächlichen Ergebnissen T, um mittels einer Verlustfunktion (*eng. loss function*) einen Distanzwert auszumachen. Basierend auf dem Distanzwert passt im letzten Schritt ein Optimierer die Netzparameter an. Das Update der Parameter basiert auf der gewählten Gradientenfunktionalität und stellt eine Funktion $W_{i,j}^k(t+1) = W_{i,j}^k(t) - \alpha \frac{\delta E}{\delta W_{i,j}^k}$ für das Verbindungsgewicht zwischen Neuron i, in der versteckten Schicht k, und Neuron j, bzw. $b_i^k(t+1) = W_i^k(t) - \alpha \frac{\delta E}{\delta b_i^k}$ für die Aktivierungsfunktion des Neurons i in der versteckten Schicht k, dar. Hierbei ist E die Berechnungsfunktion des Lernproblems und α eine positive Lernrate. Die Wahl von α beeinflusst ob und wie schnell der Gradient der Abweichung konvergiert. Da die Anpassung der Netzparameter von der letzten hin zur ersten Schicht erfolgt, wird der letzte Schritt oftmals als Rückwärtsrechnung bezeichnet. (Chollet (2018), Awad and Khanna (2015))

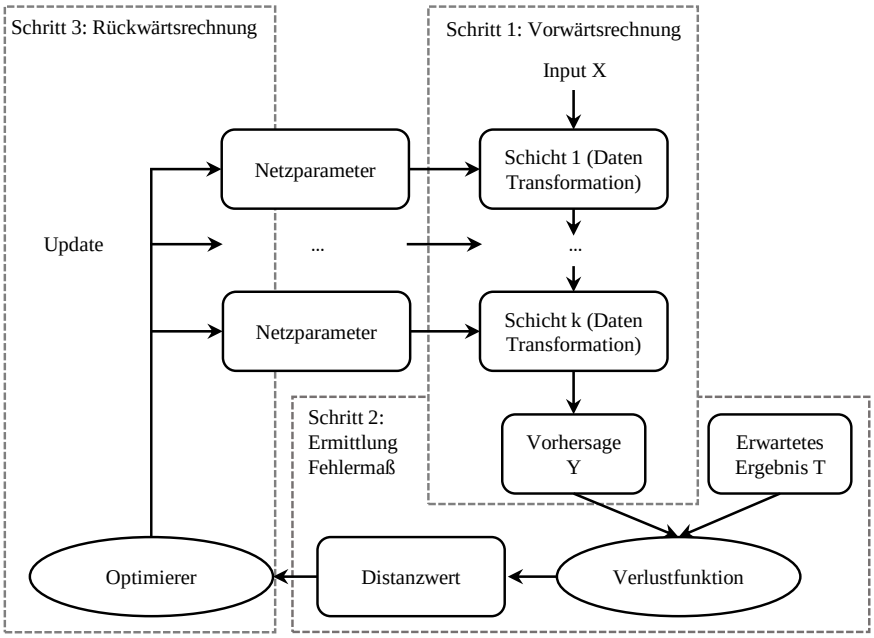

Abbildung 7 – Lernproblem neuronaler Netze, nach Chollet (2018)

2.1.3 Verschwindende und explodierende Gradienten

Der Lernvorgang von RNN ist eine Herausforderung, besonders wenn Langzeit-Abhängigkeiten innerhalb der Daten vorliegen (Hochreiter et al. (2001), Bengio et al. (2003)). Hierbei tritt das Problem der verschwindenden oder explodierenden Gradienten auf. Dieses Phänomen resultiert aus der Anpassung der rekurrenten Verbindungsgewichte, die sich über mehrere Zeitschritte ausbreiten. Demonstriert wird dieses Problem anhand des folgenden Beispiels eines RNN mit einem Input-, einem Output- und einem versteckten Neuron (Abbildung 8).

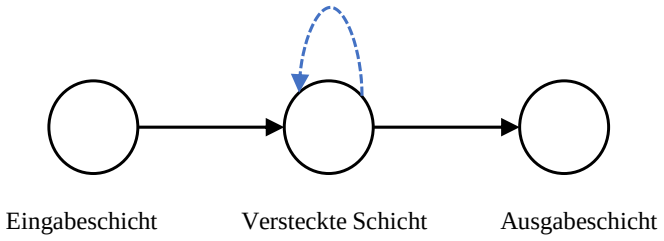

Abbildung 8 – Einfaches RNN

Zu der Zeit $t-1$ erhält das Netz eine Eingabe U_{t-1} und berechnet anhand der

Ausgabe Y_{t+1} den Fehler E_{t+1} zu dem Zeitpunkt $t+1$. Zwischen den Zeitschritten $t-1$ und $t+1$ erhält das Netz keine weiteren Eingaben. Aufgrund des Speichers der RNN hängt das Fehlermaß des Zeitpunktes t nicht nur von den Berechnungen des aktuellen Zeitpunktes ab, sondern wird zudem von den Berechnungen des Netzes in vorherigen Zeitschritten beeinflusst (siehe Abbildung 9). Diese Verknüpfung heißt *Backpropagation through time*. (Lipton et al. (2015))

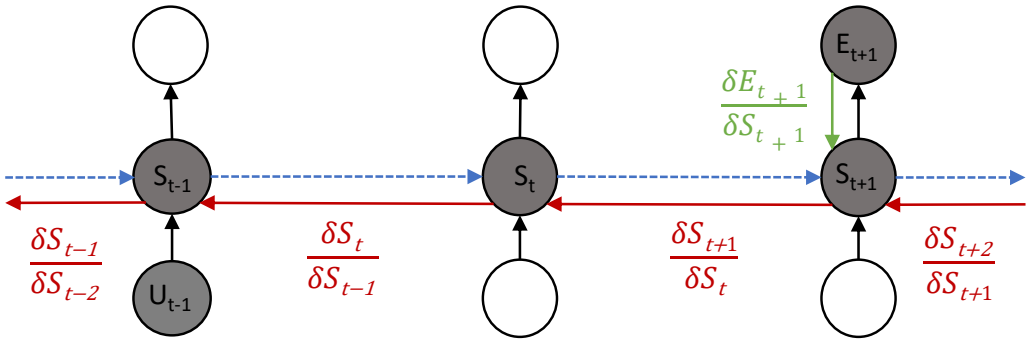

Abbildung 9 – Backpropagation durch die Zeit, nach Pascanu et al. (2013)

Die Variable X_t beschreibt den Zustand des Netzes zum Zeitpunkt t (2.1) . Hierbei beschreibt θ die Modellparameter, bestehend aus den Gewichtsmatrizen der rekurrenten und vorwärtsgerichteten Verbindungen, sowie die jeweilige Aktivierungsfunktion der einzelnen Neuronen. Gemäß des Gradienten der Verlustfunktion werden auf der Basis des Fehlermaßes nach jeder Vorwärtsrechnung die Netzparameter angepasst.

$$X_t = F(X_{t-1}, U_t, \theta) \tag{2.1}$$

$$\frac{\delta E_{t+1}}{\delta \theta} = \sum_{k=t-1}^{t+1} \frac{\delta E_{t+1}}{\delta S_{t+1}} \frac{\delta S_{t+1}}{\delta S_k} \frac{\delta S_k}{\delta \theta} \tag{2.2}$$

$$\frac{\delta S_{t+1}}{\delta S_k} = \prod_{i>(k)}^{i\leq(t+1)} \frac{\delta S_i}{\delta S_{i-1}} \tag{2.3}$$

Bedingung (2.2) stellt den Anpassungsfaktor der Netzparameter zum Zeitpunkt $t+1$ dar. Die Summe auf der rechten Seite der Gleichung beschreibt die einzelnen zeitlichen Komponenten der Modifikation und zeigt den Einfluss der Parameter

auf das Fehlermaß zum Zeitpunkt k auf. Die Netzanpassungen (2.3) transportieren den Fehler über einzelne Zeitschritte hinweg. (Pascanu et al. (2013)) Je nach Ausprägung des Terms $\frac{\delta S_i}{\delta S_{i-1}}$ treten explodierende oder verschwindende Gradienten auf. Sofern $\frac{\delta S_i}{\delta S_{i-1}} > 1$ ist, vergrößert sich sowohl der Wert $\frac{\delta S_{t+1}}{\delta S_k}$, als auch der Anpassungsfaktor mit jedem zusätzlichen Zeitschritt. Dadurch erfolgt eine zu starke Anpassung der Netzparameter und das Netz wird übertrainiert. Hierdurch steigt die Fehlerzahl in der Ausgabe an. Gegensätzlich verhält sich der Anpassungsfaktor für $\frac{\delta S_i}{\delta S_{i-1}} < 1$. Aufgrund der Multiplikation von Werten kleiner 1, nähert sich das Produkt mit zunehmender Anzahl an Faktoren dem Wert 0. Daher verkleinert sich der Wert $\frac{\delta S_{t+1}}{\delta S_k}$ mit jedem zusätzlichen Zeitschritt und der Anpassungsfaktor hat nur noch einen marginalen Einfluss auf die Netzparameter. Dadurch steigt die Fehlerzahl zwar nicht zwangsläufig an, jedoch ist es mit zunehmenden Zeitschritten immer schwieriger, die Netzparameter so anzupassen, dass Fehler aus den Prognosen verschwinden. (Lipton et al. (2015))

2.2 Neuronale Netze in der Zeitreihenprognose

Eine Zeitreihe ist eine Menge geordneter Beobachtungen X_t mit $t = 1, ..., T$ Ausprägungen, die entsprechend eines Zeitindexes geordnet sind. Dabei ist davon auszugehen, dass das Betrachtungsobjekt im Zeitverlauf variiert und daher in irgendeiner Art und Weise Zufallseinflüssen unterworfen ist. Zeitreihen können in univariate und multivariate Zeitreihen untergliedert werden. Hierbei ist das Unterscheidungskriterium die Anzahl der simultan betrachteten Zusammenhänge in der Zeitreihe. Während univariate Zeitreihen lediglich eine einzige Zeitreihe als Betrachtungsobjekt untersuchen, beschreibt die multivariate Zeitreihe die Zusammenhänge zwischen mehreren Zeitreihen und untersucht dadurch mehrere Variablen zeitgleich.(Leiner (2018)) Bedingt durch die Annahme, dass eine Zeitreihe als Wirkung von vier Ursachengruppen anzusehen ist, lässt sich diese in die folgenden vier Bewegungskomponenten untergliedern: (Wald (1936), Crone (2010), Leiner (2018))

- **Trend:** Ist die allgemeine Grundrichtung der Zeitreihe über längere Zeitabschnitte und beschreibt somit den Hauptverlauf der Zeitreihe
- **Zyklus:** Überlagert den Trend und beschreibt die kurz- und mittelfristigen Schwankungen der Zeitreihe

- **Saisonalität:** Beschreibt Schwankungen die über ein längeres Zeitintervall zu bestimmten Zeitpunkten wiederkehren
- **Irreguläre Komponente:** Steht für den Einfluss von Störfaktoren mit Zufallscharakter auf die Zeitreihe

Zeitreihenprognosen beschäftigen sich mit der Vorhersage einer Zeitreihe über einen oder mehrere Zeitschritte in der Zukunft, basierend auf den beobachteten Ausprägungen in den Daten (Runkler (2010)).

Die Verfügbarkeit großer Datenbestände, die nicht lineare Muster enthalten, führt ein Bedürfnis nach nicht linearen Prognosemodellen mit sich. RNN eignen sich besonders für die Modellierung dynamischer Systeme. Dies resultiert zum einen aus der Fähigkeit, auf der Basis von Inputdaten zu arbeiten, und zum anderen aus der Möglichkeit, unter der Verwendung rückwärtsgerichteter Kanten, Informationen zeitlich verzögert zu betrachten. Hierbei ist eine direkte Verarbeitung zeitlicher Abhängigkeiten möglich. (Hsu (2017)) Unter allen rekurrenten Netzstrukturen stechen bei der Charakterisierung von Langzeiterinnerungen besonders das *long short-term memory network* (LSTM) und die *Gated Recurrent Unit* (GRU) hervor. Beide lernen das Verarbeiten von Kurz- und Langzeiterinnerungen, indem sie einen konstanten Fehlerfluss durch einen vorgefertigten Zustand einzelner Neuronen erzwingen. Der nächste Zeitschritt erhält den unveränderten internen Zustand der Neuronen, welcher ausschließlich über *Gates* verändert wird, wodurch sich das Problem der verschwindenden und explodierenden Gradienten umgehen lässt. (Hsu (2017), Brinkmann (2019))

2.2.1 Long short-term memory network

Die LSTM Zelle wurde erstmals von Hochreiter and Schmidhuber (1997) eingeführt und unterscheidet sich grundsätzlich von anderen Neuronen durch die Aufteilung des Zellzustandes in den Kurzzeit Zustand $h_{(t)}$ und den Langzeit Zustand $c_{(t)}$. Die Idee der LSTM Zelle beruht auf der Annahme, dass das neuronale Netz lernt, Informationen im Langzeit Zustand zu speichern und Informationen daraus zu löschen. Der Langzeit Zustand $c_{(t-1)}$ durchläuft die Zelle von links nach rechts. Dabei stößt es zunächst auf das *forget gate* (Abbildung 10). Dieses dient dem Löschen von Erinnerungen aus dem Langzeit Speicher. Der nachfolgenden Additionsoperator erweitert den Langzeit Speicher der Zelle. Das daraus resul-

tierende Ergebnis $c_{(t)}$ wird ohne weitere Transformationen weitergeleitet. Durch diesen Prozess erhält oder verliert der Langzeit Speicher in jedem Zeitschritt Informationen. Nach dem Additionsoperator werden die Informationen des Langzeit Gedächtnisses zusätzlich dem Kurzzeit Gedächtnis hinzugefügt. Dafür muss der Langzeit Zustand eine *tanh* Aktivierungsfunktion passieren, dessen Ergebnis am *Output gate* gefiltert wird und somit die Kurzzeit Erinnerung $h_{(t)}$ anpasst. Diese entspricht zudem dem Output der Zelle $y_{(t)}$ zur Zeit t.

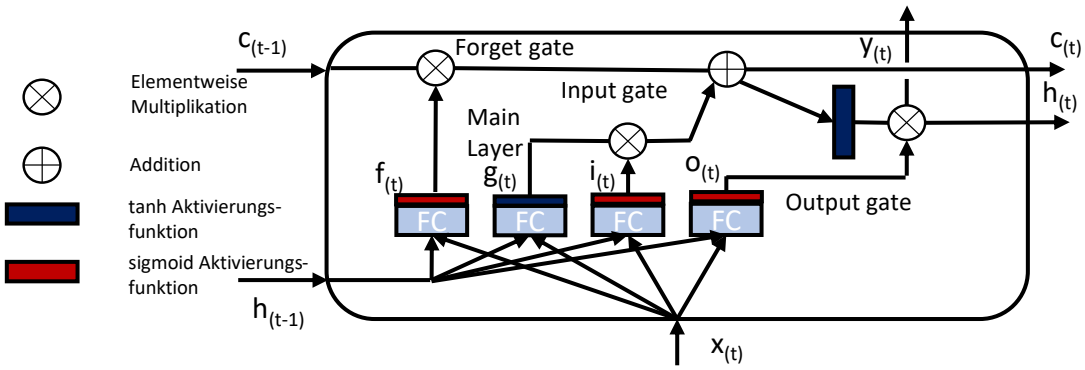

Abbildung 10 – Aufbau LSTM Zelle, nach Géron (2019)

Der Input $X_{(t)}$ sowie die Kurzzeit Erinnerung $h_{(t-1)}$ speisen vier verschiedene, komplett verbundene (*eng. fully connected*, FC) Schichten, die jeweils anderen Funktionen dienen:

- **Main Layer :** Produziert den Output $g_{(t)}$ und verarbeitet den Input $X_{(t)}$ sowie die Informationen des Kurzzeit Speicher $h_{(t-1)}$
- **Forget gate:** Produziert den Output $f_{(t)}$ und kontrolliert das Löschen von Informationen
- **Input gate :** Produziert den Output $i_{(t)}$ und kontrolliert welche Bestandteile des Outputs $g_{(t)}$ in den Langzeit Speicher aufgenommen werden
- **Output gate:** Produziert den Output $o_{(t)}$ und kontrolliert welche Bestandteile des Langzeit Speichers $c_{(t-1)}$ in diesem Zeitschritt berücksichtigt werden und welche Informationen ausgegeben werden, sowohl für $h_{(t)}$ als auch für $y_{(t)}$

2.2.2 Gated Recurrent Unit

Chung et al. (2014) haben die GRU Zelle erstmals vorgestellt. Ihre innere Struktur ist der LSTM Zelle nachempfunden, da die Steuerung des Zustandes der Zelle ebenfalls mit der Hilfe von *Gates* erfolgt. Im Gegensatz zur LSTM Zelle ist der Zellzustand jedoch nicht in Lang- und Kurzzeit aufgeteilt, sondern der Zustand $h_{(t)}$ beschreibt beide Zustände. Zusätzlich ist die Anzahl der *Gates* reduziert (Abbildung 11).

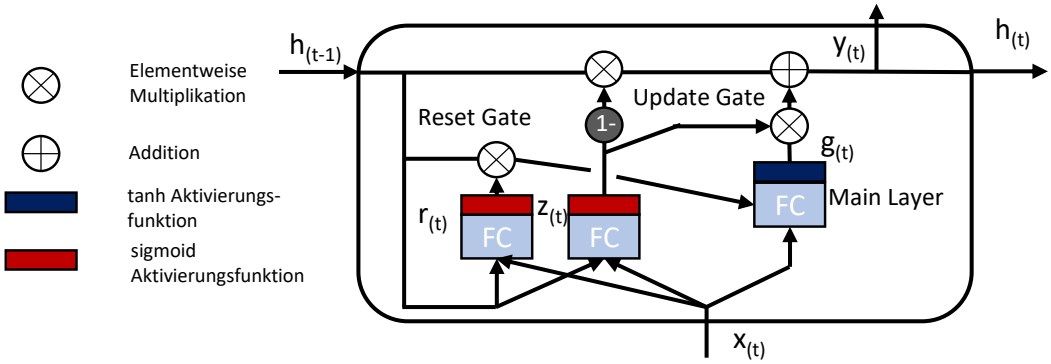

Abbildung 11 – Aufbau GRU Zelle, nach Géron (2019)

Das *Reset Gate* produziert den Output $r_{(t)}$ und entscheidet über den Einbezug vorheriger Ausgaben $h_{(t-1)}$ für die Berechnung des neuen Zellzustandes $g_{(t)}$. Dadurch übernimmt das *Gate* die Funktionen des *Forget Gates* und des *Input Gates* der LSTM Zelle, indem es als Schaltung fungiert. Sofern das *Reset Gate* aktiviert ist, wird das *Input Gate* geöffnet und das *Forget Gate* geschlossen. Sofern das *Reset Gate* nicht aktiviert ist, wird die Funktion des *Forget Gate* aktiviert. Durch diesen Prozess ist die GRU Zelle in der Lage, Informationen aus dem Gedächtnis zu löschen und neue Erinnerungen aufzunehmen. Da die GRU Zelle über kein *Output Gate* verfügt, besteht der Output in jedem Zeitschritt aus dem gesamten Zellzustand. Das zweite *Gate* in der GRU Zelle, das *Update Gate*, produziert den Output $z_{(t)}$ und steuert den Anteil des Informationsflusses, der aus dem Gedächtnis $h_{(t-1)}$ bzw. dem aktuellen Zellzustand $g_{(t)}$, in das Gedächtnis aufgenommen und als Output $y_{(t)}$ ausgegeben wird.

3 Modelldefinition

Ein Begriff der häufig mit neuronalen Netzen in Verbindung gebracht wird, ist die Generalisierbarkeit. Diese beschreibt die Fähigkeit eines Netzes, mit einem Datensatz zu trainieren und auf der Grundlage dieses Lernprozesses richtige Resultate in unabhängigen Testmengen zu realisieren (Rosin and Fierens (1995)). Die Generalisierbarkeit ist von zwei Wirkursachen geprägt, die sich gegenläufig zueinander verhalten, weshalb beide möglichst klein gehalten werden sollten (Yu et al. (2005)). Zum einen können Verzerrungen durch systematische Fehler in den Daten entstehen, oder es sind zu wenige Parameter im Modell vorhanden, um alle Verzerrungen in den Daten zu erfassen. Die zweite Wirkursache beschäftigt sich mit Varianzen, die durch eine zu genaue Beschreibung von Abhängigkeiten in den Daten entstehen können. Der Zusammenhang zwischen den beiden Wirkursachen lässt sich besonders durch die Modellkomplexität sowie die Anzahl an Vorhersagefehlern verdeutlichen (Abbildung 12).

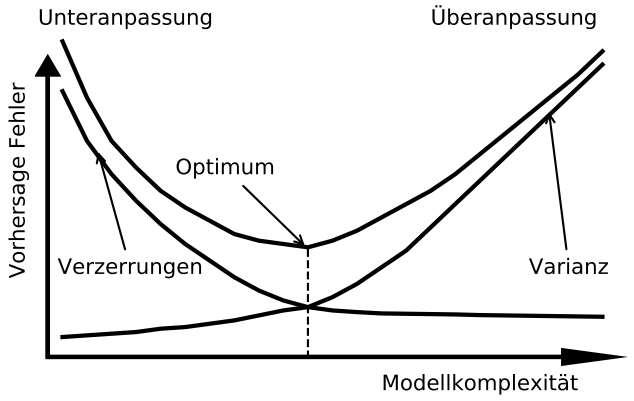

Abbildung 12 – Zusammenhang zwischen Vorhersagefehlern und Modellkomplexität, nach Broch (2017)

In Modellen mit einer einfachen Architektur und wenigen Parametern entsteht eine Vielzahl an Vorhersagefehlern, da eine hohe Anzahl an Varianzen nicht

beschrieben werden. Zusätzlich beeinflussen viele Verzerrungen die Vorhersagen, da die einfache Modellkomplexität diese nicht erfasst. Bei einer extremen Ausprägung des Sachverhalts kann das Problem der Unteranpassung (*eng. Underfitting*, Abbildung 13) entstehen. Bei dieser beschreibt das Modell die Zusammenhänge im Datensatz nicht genau genug, wodurch Vorhersagefehler entstehen und die Generalisierbarkeit gering ist. Auf der anderen Seite tritt eine Überanpassung (*eng. Overfitting*) auf, sofern die Verzerrungen niedrig und die Varianz hoch ist. Hierbei bildet das Modell die Zusammenhänge in den Trainingsdaten sehr gut ab, jedoch entstehen Ungenauigkeiten bei der Inter- und Extrapolation der Modellfunktion, wodurch die Generalisierbarkeit wiederum abnimmt. (Broch (2017))

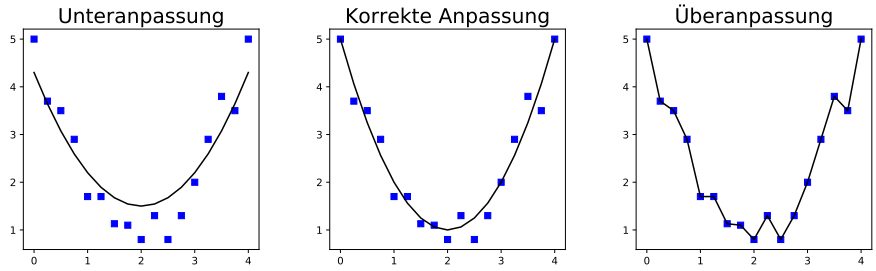

Abbildung 13 – Unteranpassung (links), korrekte Anpassung (mitte), Überanpassung (rechts)

Für die in Kapitel 5 durchgeführten Versuche werden drei unterschiedliche Netze benutzt. Einmal wird ein normales FFN verwendet, das zweite Netz besitzt lediglich LSTM Zellen in der versteckten Schicht und die versteckte Schicht des letzten Netzes besteht aus GRU Zellen. Die Definition der Netzstruktur sowie die Festlegung der Modellparameter erfolgt zunächst allgemeingültig für alle Netze. Sofern der Aufbau eines Netzes sich von den anderen unterscheidet, wird dieses explizit aufgezeigt. Weiterhin sind die hier definierten Modelle der Ausgangspunkt für die später durchgeführten Versuche. Für den Fall einer Änderung von einzelnen Netzstrukturen innerhalb der Versuche, wird dies im Versuchsaufbau näher erläutert.

3.1 Modellarchitektur

Neuronale Netze sind dazu fähig, mit einer gegebenen Anzahl an nicht linearen Neuronen aus Erfahrungen zu lernen und somit jeden komplexen funktionalen Zusammenhang mit hoher Akkuratheit darzustellen. Die richtige Architektur dafür zu finden ist keine einfache Aufgabe. Obwohl in der Literatur eine Vielzahl an Richtlinien zur Bestimmung der optimalen Netzstruktur, wie bspw. der *pruning algorithm* (Renda et al. (2020)), der *polynominal time algorithm* (Zhang et al. (2016)), oder die *canonical decomposition technique* (Atamanyuk and Kondratenko (2016)) diskutiert werden, kann keiner dieser Ansätze garantieren, die beste Architektur für alle Vorhersagesituationen zu beschreiben.

3.1.1 Anzahl und Art der Schichten

Jedes Netz verfügt über genau eine Input- und eine Output Schicht. Schwieriger gestaltet sich die Festlegung einer Anzahl an versteckten Schichten. Da Zhang and Qi (2005) gezeigt haben, dass bereits normale FFN mit 2 versteckten Schichten die Fähigkeit besitzen, sich bei der Prognose von Zeitreihen jeder beliebigen Funktion anzunähern, wird die Anzahl an versteckten Schichten auf 2 festgelegt. Wie auch bei der Input- und Outputschicht ist die Art der jeweiligen Schicht zu bestimmen.

Dabei erfolgt zunächst eine Betrachtung des Inputs, den das Netz erhält, sowie des Outputs, den das Netz produziert. Die Menge $(X_1, X_2, \ldots, X_N)$ beschreibt die Ausprägungen einer Zeitreihe X, die durch eine Reihe an Vektoren $X_t = (X_t, X_{t+\tau}, X_{t+2*\tau}, \ldots, X_{t+(m-1)*\tau}) \in R^m$ dargestellt ist. Hierbei ist τ die Verzögerungszeit der einzelnen Ausprägungen, ein Vielfaches der betrachteten Periode und m die Dimension der Zeitreihe. Die Ausprägung der Zeitreihe zum Zeitpunkt T verdeutlicht der Term $X_{t+T+(m-1)*\tau}$. Diese ist abhängig von den verfügbaren historischen Ausprägungen der Zeitreihe X_t mit $t = (1, \ldots, n)$, wobei n als $n = T + (m-1) * \tau$ definiert ist. Infolge dessen kann der Output zum Zeitpunkt $t + T + (m-1) * \tau$ über die Gleichung $X^F_{t+T+(m-1)*\tau} = f(X_t) + e_t$ berechnet werden. Hierbei ist e_t ein Zufallseinfluss und die Funktion $f(.)$ bildet den historischen Verlauf des Inputs ab. (Wu et al. (2010)) Somit berechnet das Netz einen spezifischen Output zum Zeitpunkt t anhand des Verlaufs der Inputmerkmale des Zeitintervalls $(t - z)$. Die Variable z stellt die Anzahl an Zeitschritten dar, die dem Netz zur Berechnung des Outputs zusätzlich zur Verfügung stehen.

Im Folgenden wird das Zeitintervall $(t - z)$ als Sequenzlänge bezeichnet.

Da das Netz in jedem Rechenvorgang lediglich einen Outputvektor berechnet, ist eine normale Schicht als Outputschicht ausreichend. Hierfür eignen sich insbesondere dicht vernetzte Schichten (*dense layers*) (Keras (a)). Problematischer ist die Inputschicht. Bei dieser erhält das FFN ein *flatten layer*. Dieses reduziert die Dimensionalität der Inputdaten, wodurch das FFN dazu fähig ist, ohne rekurrente Verbindungen mit einer Sequenzlänge zu arbeiten. Im LSTM bzw. GRU Netz wird jeweils eine LSTM bzw. GRU Schicht für den Input genutzt. Für die versteckten Schichten stehen beim FFN Netz *dense layers* und im LSTM bzw. GRU Netz entsprechende LSTM bzw. GRU Schichten zur Verfügung. Tabelle 1 beschreibt die einzelnen Schichten der Netze.

Tabelle 1 – Übersicht Modellarchitektur - Art der Schicht

Modell	**Inputschicht**	**1. versteckte Schicht**	**2. versteckte Schicht**	**Output Schicht**
FFN	Flatten Layer	Dense Layer	Dense layer	Dense Layer
LSTM	LSTM Layer	LSTM Layer	LSTM Layer	Dense Layer
GRU	GRU Layer	GRU Layer	GRU Layer	Dense Layer

3.1.2 Neuronenzahl

Die Anzahl der Inputneuronen ist einer der wichtigsten Parameter und deren Festlegung erfolgt entsprechend der Anzahl an Merkmalen, die die im Datensatz vorhandenen Muster und Korrelationen beschreiben (Zhang et al. (2001)). Lachtermacher and Fuller (1995) haben den Effekt von zusätzlichen Inputneuronen untersucht und dabei festgestellt, dass das Vorhandensein zusätzlicher Neuronen bei der Prognose eines einzelnen Zeitschritts unerwünschte Effekte in den Vorhersagen verursacht. Jedoch ändert sich die Wirkung bei der Vorhersage über mehrerer Zeitschritte. Da die Menge der Inputneuronen einen entscheidenden Effekt auf die Lern- und Vorhersagefähigkeit bewirken, wobei zu wenig Neuronen zu einem Unterlernen und zu viele Inputneuronen zu einer Überspezifikation führen (Zhang et al. (2001)), wird die Anzahl der Inputneuronen entsprechend der genutzten Merkmale (Kapitel 4.4) auf 45 festgelegt.

Die Menge an versteckten Neuronen pro Schicht ist ebenso von großer Bedeutung. Die versteckten Schichten und deren Neuronen heben die definierten Eigenschaften des Datensatzes hervor und stellen die nicht lineare Beziehung zwischen Input

und Output dar (Hamzaçebi et al. (2009)). Ein häufig verwendeter Ansatz zur Bestimmung der Anzahl an versteckten Neuronen basiert auf Experimenten oder dem Ausprobieren mit einer Fehler basierten Anpassung (Wang et al. (2015)). Hierbei ist das Finden einer optimalen Lösung jedoch nicht gewährleistet. Einen anderen Ansatz verfolgen Hunter et al. (2012), indem die Menge an versteckten Neuronen pro Schicht in Abhängigkeit von der Anzahl der Inputneuronen bestimmt wird. Verschiedene Formeln bestimmen die Anzahl der Neuronen, je nach Art des Netzes (Bedingung 3.1, 3.2). Dabei beschreibt N_i die Menge der Inputneuronen und N_h die Anzahl der versteckten Neuronen pro Schicht.

$$N_h = N_i + 1 \tag{3.1}$$

$$N_h = 2 * N_i + 1 \tag{3.2}$$

Die Festlegung der Menge an versteckten Neuronen erfolgt beim FFN durch die Bedingung (3.2), durch welche die Neuronenzahl der versteckten Schichten auf $N_h = 2 * 45 + 1 = 91$ bestimmt wird. Da die beiden rekurrenten Netzstrukturen über deutlich mehr Verbindungen zwischen Neuronen verfügen, und somit mehr Netzparametern beinhalten, dient Bedingung (3.1) der Berechnung der Neuronenzahl. Hierdurch entspricht die Menge der versteckten Neuronen der rekurrenten Netze $N_h = 45 + 1 = 46$.

Die Anzahl der Outputneuronen ist von der jeweiligen Lernaufgabe abhängig. Bei Regressionsaufgaben kommt lediglich ein Outputneuron zum Einsatz. Dieses gibt einen präzisen Wert als Ergebnis der Vorhersage aus. Bei Klassifikationsaufgaben weicht die Menge an Neuronen in der letzten Schicht ab. Hierbei kann bspw. die $1 - of - N^{(c)}$-*encoding* Methode verwendet werden, bei welcher die Menge der Outputneuronen der Anzahl an Klassen $N^{(c)}$, in welche die Ausprägungen des Datensatzes einzuordnen sind, entspricht. Bei der Vorhersage berechnet das Netz verschiedene Wahrscheinlichkeiten für die einzelnen Outputneuronen. Anschließend gibt die Klasse des Neurons mit dem höchsten berechnetem Wert das Ergebnis aus (Hüsken and Stagge (2003)). Neben der Art der Lernaufgabe kann die Menge der Outputneuronen entsprechend verschiedener Vorhersageaufgaben variieren. Entscheidend ist insbesondere die Anzahl der Vorhersageperioden. Sofern der Vorhersagenhorizont lediglich eine Periode umfasst, ist ein Output Neuron ausreichend. Umspannt die Vorhersage mehrere Zeitschritte im Vorhersagehorizont, ändert sich die Anzahl der Outputneuronen. Sofern die Vorhersage iterativ

erfolgt, wobei nacheinander die einzelnen Zeitschritte auf der Basis der vorherigen Vorhersage ermittelt werden, ist ein Outputneuron ausreichend. Im Gegensatz dazu entspricht bei einer direkten Vorhersagemethode, bei welcher eine Berechnung die Vorhersagen für alle Zeitschritte ermittelt, die Anzahl der Outputneuronen der Anzahl an vorhergesagten Zeitschritten. (Hamzaçebi et al. (2009))

Der Ausgangsversuch sagt lediglich einen Zeitschritt in der Zukunft vorher und die Lernaufgabe ist als Klassifikation definiert. Somit erfolgt die Festlegung der Anzahl der Outputneuronen, entsprechend der Anzahl an Klassenwerten der Fahrräder, auf 11. Tabelle 2 fasst abschließend die einzelnen Modellarchitekturen zusammen.

Tabelle 2 – Übersicht Modellarchitektur - Anzahl der Neuronen

	Modell	**FFN**	**LSTM**	**GRU**
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	45	45	45
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	11	11	11

3.1.3 Aktivierungsfunktionen

Wie bereits in Kapitel 2 erwähnt, erhalten Neuronen Eingangssignale, die unter der Verwendung einer Übertragungsfunktion an eine Aktivierungsfunktion geleitet werden. Diese überträgt das Neuron in einen aktiven oder inaktiven Zustand. Basierend auf dem Zustand des Neurons gibt es einen Ausgabewert aus, der über die Aktivierungsfunktion bestimmt wird. Die Werte liegen oftmals innerhalb des Intervalls $[0 : 1]$, bzw. $[-1 : 1]$. Die in Abbildung 14 dargestellte Stufenfunktion soll die Funktionsweise der Aktivierungsfunktion anhand eines einfachen Beispiels verdeutlichen. Sofern die Summe der Eingangssignale den Schwellenwert δ überschreitet, wird das Neuron aktiviert und die Aktivierungsfunktion gibt den Wert 1 aus. Sofern der Schwellenwert nicht überschritten wird bleibt das Neuron inaktiv und gibt somit den Wert 0 aus (Sharma (2017)).

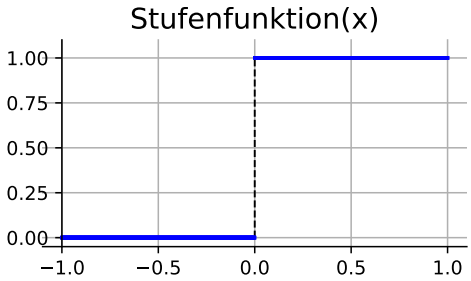

Abbildung 14 – Beispiel Stufenfunktion

Wie in Kapitel 2.2 beschrieben besitzt die LSTM bzw. die GRU Zelle im Inneren eigene Schichten die jeweils mit *tanh* oder *sigmoid* Aktivierungsfunktionen ausgestattet sind. Die *sigmoid* Aktivierungsfunktion ist eine nicht lineare Aktivierungsfunktion, die insbesondere in FFN benutzt wird.

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Der Wertebereich liegt im Intervall $[0, 1]$. Dabei bildet die Funktion eine s-förmige Kurve ab (Abbildung 15). Problematisch bei der Verwendung dieser Aktivierungsfunktion ist jedoch, dass deren Ausgaben nicht auf 0 zentriert sind. Dies führt dazu, dass die Aktualisierung des Gradienten zu weit in verschiedene Richtungen auseinander geht und somit die Optimierung des Netzes erschwert. Die Tangens hyperbolicus (*tanh*) Aktivierungsfunktion hingegen ist auf 0 zentriert und kann einen größeren Wertebereich im Intervall $[-1 : 1]$ annehmen (Abbildung 15). Die Funktion verläuft ebenfalls s-förmig und ist definiert als: (Redaktion AI-United)

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.4}$$

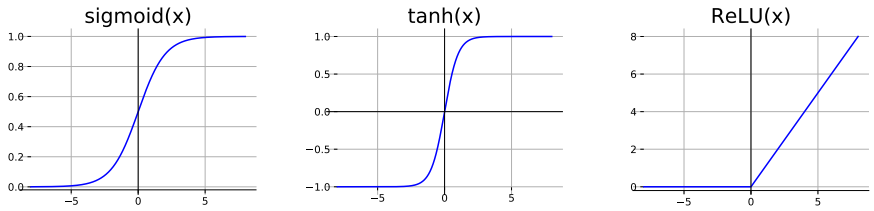

Abbildung 15 – Sigmoid tanh und ReLU Aktivierungsfunktionen

Beide Aktivierungsfunktionen haben das Problem einer schnellen Sättigung. Infolge dessen nimmt die Aktivierungsfunktion für große Werte den Wert 1 und

für kleine Werte den Wert 0 bzw. -1 an. Dadurch reagieren beide Aktivierungsfunktionen lediglich im mittleren Wertebereich empfindlich auf Änderungen der Eingangssignale. Das schnelle Erreichen des Sättigungspunktes und die unzureichende Empfindlichkeit führen dazu, dass Gewichtsanpassungen schwer festzulegen sind und somit die Leistung des Modells, ab einem gewissen Punkt, nicht weiter verbessert werden kann. (Brownlee (2019c))

Nair and Hinton (2010) haben im Jahr 2010 die *rectified linear unit* (ReLU) Aktivierungsfunktion vorgestellt. Diese ist eine schnell lernende Aktivierungsfunktion (Abbildung 15, rechts) und wird immer häufiger in KNN verwendet. Im Vergleich mit der *sigmoid* und *tanh* Aktivierungsfunktion liefert die $ReLU$ eine bessere Performance und somit eine bessere Generalisierbarkeit des Modells. Die Funktion setzt alle Werte kleiner 0 auf 0 und ist definiert als:

$$f(x) = ReLU(x) = max(0, x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.5}$$

Da die rekurrenten Netze über sogenannte gestapelte Schichten (*eng. stacked layers*, siehe S. 26 f) verfügen, leitet die erste versteckte Schicht neben dem Output weiterhin den internen Zustand der einzelnen Zellen an die folgende Schicht weiter. Damit die $ReLU$ Aktivierungsfunktion den internen Zellzustand bei der Weitergabe nicht beeinflusst, erfolgt die Verwendung von *sigmoid* und *tanh* Aktivierungsfunktionen in der ersten versteckten Schicht der rekurrenten Netze. In der zweiten versteckten Schicht kommt die $ReLU$ Aktivierungsfunktion zusätzlich für den Output der Zelle zum Einsatz. Hierbei berechnen innerhalb der Zelle *sigmoid* und *tanh* Aktivierungsfunktionen ein Zwischenergebnis, das im Folgenden an die $ReLU$ Aktivierungsfunktion übertragen wird. Die Weitergabe deren Outputs erfolgt letztlich an die Neuronen der Ausgabeschicht. Da das FFN über keinen internen Zellzustand verfügt, erhält dieses in beiden versteckten Schichten die $ReLU$ Aktivierungsfunktion.

Die $ReLU$ Aktivierungsunktion eignet sich insbesondere zur Verwendung in versteckten Schichten. Deshalb kommt eine andere Aktivierungsfunktion für die Ausgabeschicht zum Einsatz. Hier empfiehlt sich die *softmax* Aktivierungsfunktion. Diese wird insbesondere bei multivariaten Klassifikationsaufgaben eingesetzt und ist definiert als:

$$f(x) = softmax(x) = \frac{exp(x_i)}{\sum_j exp(x_i)} \tag{3.6}$$

Die Inputwerte werden auf einen Vektor mit Wahrscheinlichkeitsverteilung normalisiert (Nwankpa et al. (2018)). Die normalisierten Werte beschreiben jeweils die Wahrscheinlichkeit, dass das Ergebnis in die entsprechende Klasse eingeordnet wird. Somit ergibt die Summe der normalisierten Werte den Wert 1 und die Klasse mit der höchsten Wahrscheinlichkeit ist das Ergebnis (Mahmood (2018)). Die folgende Tabelle soll einen abschließenden Überblick über die einzelnen Modellarchitekturen geben.

Tabelle 3 – Übersicht Modellarchitektur - Aktivierungsfunktionen

	Modell	**FFN**	**LSTM**	**GRU**
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	45	45	45
	Aktivierungsfunktion	-	-	-
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
	Aktivierungsfunktion	ReLU	tanh	tanh
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
	Aktivierungsfunktion	ReLU	ReLU	ReLU
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	11	11	11
	Aktivierungsfunktion	Softmax	Softmax	Softmax

3.2 Modellparameter

Neben der Modellarchitektur spielen die Modellparameter eine ebenso entscheidende Rolle. Diese definieren das Verhalten des Berechnungsgraphen während des Lernprozesses und nehmen somit wesentlichen Einfluss auf einen erfolgreichen Lernvorgang von Netzen.

3.2.1 Verlustfunktion

Die Verlustfunktion berechnet den Fehler zwischen dem von Netz berechneten Output und dem tatsächlichen Ergebnis. Dabei berechnen unterschiedliche Verlustfunktionen für die gleiche Problemstellungen verschiedene Fehlermaße. Da-

durch nimmt die Wahl der Verlustfunktion einen Einfluss auf die Modellperformance (Agrawal (2017)). Die Verlustfunktion *categorical crossentropy* eignet sich insbesondere für Klassifikationsaufgaben mit mehreren Klassen, indem sie den Unterschied zwischen zwei Wahrscheinlichkeitsverteilungen berechnen und somit den durch die *softmax* Aktivierungsfunktion berechneten Output verarbeiten kann. (Brownlee (2019b))

Die *categorical crossentropy* Verlustfunktion arbeitet dabei auf der Basis des logarithmischen Verlustes (*eng. logarithmic loss*). Dessen Maß wird durch die Bestrafung falscher Vorhersagen bestimmt. Für einen Datensatz mit N Ausprägungen, der M Klassen beschreibt, ist der logarithmische Verlust definiert als:

$$\text{Logarithmischer Verlust } = \frac{-1}{N} * \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * log(p_{ij}) \tag{3.7}$$

Hierbei gibt y_{ij} an, ob ein Beispiel i zur Klasse j gehört und p_{ij} beschreibt die vom Netz berechnete Wahrscheinlichkeit, mit der das Beispiel i zur Klasse j gehört. Der Wertebereich des logarithmischen Verlustes kann durch das Intervall $[0, \infty]$ beschrieben werden. Dabei ist die Performance des Netzes umso besser, je stärker der logarithmische Verlust sich dem Wert 0 annähert. (Mishra (2018))

3.2.2 Optimierer

Optimierer passen Verbindungsgewichte und andere Parameter im Netz an und verringern somit das Fehlermaß innerhalb der Vorhersagen. Grundsätzlich können zwei verschiedene Arten von Optimierern ausgemacht werden, die sich hinsichtlich der Lernrate unterscheiden.

Optimierer mit einer konstanten Lernrate, wie beispielsweise der stochastische Gradientenabstieg (*eng. stochastic gradient descent*, (*SGD*)), berechnen den Gradienten für den gesamten Datensatz und aktualisieren bei der Rückwärtsrechnung alle Netzparameter mit einer konstanten Lernrate bis ein Optimum gefunden ist. Zusätzlich können weitere Parameter wie bspw. die Lernrate α, welche das Maß bestimmt, mit dem alle Netzparameter verändert werden, eingestellt werden. Die Wahl der Lernrate kann schwierig sein, da eine kleine Lernrate zu einem langsamen Konvergieren der Verlustfunktion führt. Im Gegensatz dazu kann eine hohe Lernrate das Konvergieren der Verlustfunktion verhindern. (Agrawal (2017))

Im Gegensatz zu den Optimierern mit einer konstanter Lernrate, passen die adaptiven Optimierer nicht alle Netzparameter mit einer definierten Lernrate an, sondern variieren individuell die Veränderung je nach Bedeutung des Parameters. Ein Vergleich zwischen zwei Optimierern, die auf dem aktuellen Stand der Forschung sind, der *SGD* Optimierer und der adaptiven Lernalgorithmus *ADAM* (*eng. adaptive moment estimation*), hat gezeigt, dass der *SGD* deutlich schneller ein Ergebnis gefunden hat. Jedoch war die Güte dieses Ergebnisses deutlich vom Optimum entfernt. *ADAM* hingegen hat ein wesentlich besseres Optimum gefunden, dafür aber auch deutlich mehr Zeit benötigt. Die Wahl des Optimierers ist somit ein Kompromiss zwischen einer höheren erforderlichen Rechenleistung und dem optimalen Ergebnis. (Agrawal (2017))

Das FFN arbeitet mit dem *SDG* Optimierer. Dieser ermöglicht das Einstellen zusätzliche Parameter, wie die Lernrate, das Momentum oder das Nesterov Momentum (Keras (d)). Da die Aktivierungsfunktion über keine obere Schranke verfügt, besitzt diese einen großen Wertebereich. Infolge dessen fällt die Wahl auf eine niedrige Lernrate. Der SDG Optimierer mit Momentum ist eine Methode, die hilft, die Gradientenvektoren in die richtige Richtung zu beschleunigen (Abbildung 16), woraus eine schnellere Konvergenz resultiert. Hierbei erfolgt die Ermittlung des Momentums aus den Gradientenanpassungen vergangener Schritte, um die Richtung der Gradientenänderung beim aktuellen Schritt zu optimieren (Kathuria (2018)). Der Parameter w für den berechneten Gradienten g erfährt die folgende Angepassung (Keras (2020)):

$$\text{velocity} = \text{momentum} * \text{velocity} - \text{Lernrate} * g \tag{3.8}$$

$$w = w + \text{momentum} * \text{velocity} - \text{Lernrate} * g \tag{3.9}$$

Das Nesterov Momentum ist ein anderes Verfahren zur Ermittlung dieser Richtung (Abbildung 16, rechts) (Bushaev (2017)). Während die Bestimmung der *velocity* entsprechend Bedingung (3.9) erfolgt, weicht die Berechnung von w ab und entspricht Bedingung (3.10) (Keras (2020)).

$$w = w * \text{velocity} \tag{3.10}$$

Das FFN arbeitet mit dem *SGD* Optimierer mit Nesterov Momentum und die beiden rekurrenten Netze mit dem *ADAM* Optimierer.

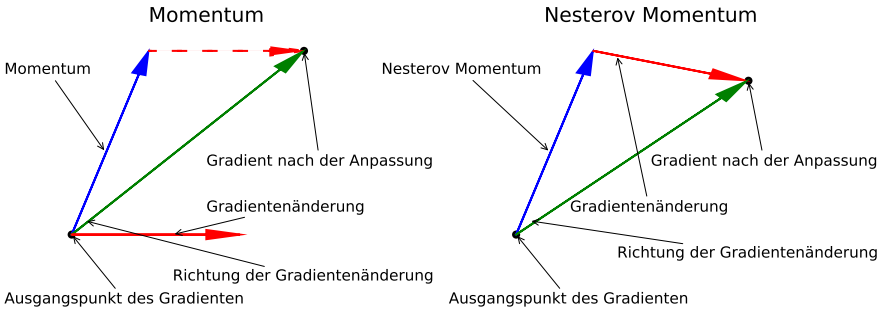

Abbildung 16 – Unterschied zwischen Momentum und Nesterov Momentum, nach Bushaev (2017)

3.2.3 Weitere Parameter

Die Netze verfügen über weitere Parameter, die teilweise nur für die beiden rekurrenten Netzstrukturen von Bedeutung sind:

Dropout

Der *Dropout* bezeichnet ein Verfahren, das versteckte und sichtbare Neuronen aus dem Netz entfernt. Der Begriff *Entfernen* beschreibt hierbei das temporäre Entnehmen einzelner Neuronen und deren Verbindungsgewichte aus der Netzstruktur (Abbildung 17). Die Wahl der entfernten Neuronen erfolgt dabei zufällig. Im einfachsten Fall definiert eine Wahrscheinlichkeit den *Dropout*. Diese wird bei den beiden rekurrenten Netzen auf 0.1 festgelegt. Srivastava et al. (2014) haben nachgewiesen, dass die Verwendung eines *Dropouts* einen entscheidenden Vorteil, unabhängig von der Netzstruktur und den Netzparametern mit sich bringt, durch die Reduktion des Fehlermaßes beim Training.

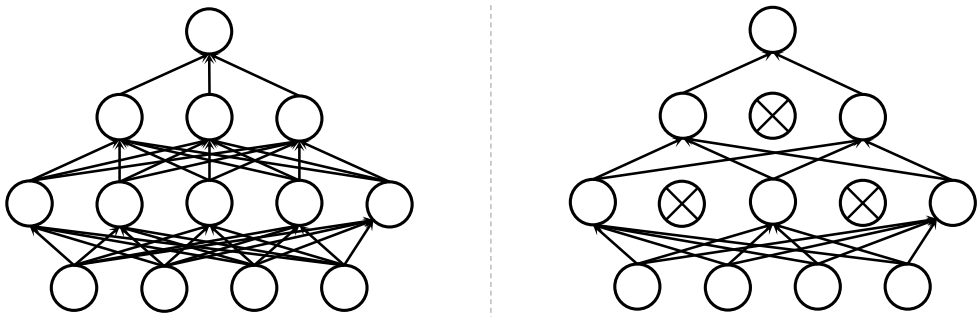

Abbildung 17 – Neuronales Netz vor (links) und nach (rechts) dem Dropout, nach Srivastava et al. (2014)

Rekurrenter Ausfall und Rückgabesequenz

Der *rekurrente Dropout* (*eng. recurrent dropout*) besitzt die gleiche Funktion wie der *Dropout*, wendet dies jedoch hinsichtlich der rekurrente Neuronen und Verbindungen an. Die Wahrscheinlichkeit des *rekurrenten Dropouts* beträgt 0.5. Die *Rückgabesequenz* (*eng. return sequence*) ist ein boolscher Operator, der bei Aktivierung die komplette Output Sequenz an die nächste Schicht weiterleitet. Sofern der Operator nicht aktiviert ist, erhält die nachfolgende Schicht lediglich den letzten Output der Outputsequenz (Keras (c)). Bei beiden rekurrenten Netzen ergibt sich dadurch eine Struktur, die als gestapelte Schichten (*eng. stacked layers*) bekannt ist. Die entstehende tiefere Netzstruktur ermöglicht es dem Modell, zusätzliche Level der Abstraktion darzustellen und somit die Eingangssignale im Laufe der Zeit zu separieren, oder die Problemstellung auf verschiedenen Zeitskalen abzubilden. (Brownlee (2019a))

Frühzeitiger Stopp und Modellcheckpoints

Die Funktion des frühzeitigen Stopps (*eng. early stopping*) beendet das Training, sofern ein überwachter Parameter sich über eine definierte Anzahl an Lernschritten nicht weiter verbessert. Innerhalb der Funktion können dafür verschiedene Parameter eingestellt werden. Hierbei sind insbesondere die Überwachung, der Modus und die Verzögerung relevant. Die Überwachung des Wertes der Validierungsverlustfunktion beendet das Training, sofern der Wert über eine Verzögerungsdauer nicht weiter minimiert wird.(Keras (b)) Entscheidend ist hierbei das min

delta, welches eine Minimierungsvorschrift für den Verlustwert festlegt, um das Training fortzusetzen. Die Verzögerung definiert die Anzahl der Epochs zur Einhaltung dieser Vorschrift. Ersteres entspricht dem Wert 0.001. Die Verzögerung im ersten Versuch beträgt 30 Epochs im FFN und GRU Netz und im LSTM Netz 100 Epochs.

Modell Checkpoints ermöglichen das Vervielfältigen von Modellen, durch die Speicherung aller Modellparameter in einer Datei. Hierbei speichert die Funktion die beste Modellstruktur, entsprechen dem Wert der Validierungsverlustfunktion.

3.3 Modelltraining

Neuronale Netze trainieren mit verschiedenen Verfahren. Grundsätzlich unterscheiden sich das stochastische und das Batch Lernen. Vor der genauen Erläuterung beider Lernverfahren werden zunächst einige Begrifflichkeiten geklärt.

Ein Batch ist ein Hyperparameter, der definiert, wie viele Beispiele der Trainingsdaten ein neuronales Netz bearbeitet, vor der Änderung der internen Modellparameter. Ein Epoch hingegen beschreibt die Bearbeitung des gesamten Trainingsdatensatzes durch das Netz und kann somit aus unterschiedlich vielen Batches bestehen. Infolge dessen definiert die Anzahl der Epochs die Trainingsdurchgänge mit dem gesamten Trainingsdatensatz und die Anzahl der Batches pro Epoch, die Häufigkeit der Anpassungen der Netzparameter, vor der Bestimmung der Netzperformance anhand der Validierungsdaten.

3.3.1 Lernverfahren

Der Trainingssatz besteht aus einem oder mehreren Batches. Abhängig von der Batchgröße verändert sich der Lernalgorithmus. Sofern die Größe des Batches der Größe des gesamten Datensatzes entspricht, heißt das Lernverfahren Batch Gradienten Abstieg (*eng. Batch Gradient Descent*, *BGD*). Sofern die Batchgröße 1 beträgt, ist das Lernverfahren der stochastische Gradienten Abstieg (*eng. stochastic gradient descent*, *SGD*). Die letzte Möglichkeit, der Mini-Batch Gradienten Abstieg (*eng. Mini-Batch Gradient descent*), erfüllt Bedingung (3.11). (Brownlee (2018))

$$1 < \text{Batchgröße} < \text{Größe des Trainingsdatensatzes} \tag{3.11}$$

Jedes der Verfahren besitzt einige Vorteile gegenüber den anderen. So ist der Ansatz des *SGD* normalerweise deutlich schneller beim Lernen und liefert darüber hinaus auch bessere Ergebnisse. Dafür ermöglicht das Batch Lernen eine bessere Nachvollziehbarkeit der Bedingungen der Konvergenz und die theoretische Analyse der Dynamik der Verbindungsgewichte ist einfacher. Unabhängig vom Lernverfahren existieren weitere Optimierungsmöglichkeiten. Hierbei ist zu berücksichtigen, dass die Lernfähigkeit neuronaler Netze durch unerwartete Beispiele des Trainingsdatensatzes steigt. Dadurch nimmt die Zusammenstellung der Batches Einfluss auf deren Informationsgehalt, indem Beispiele, die ein hohes Fehlermaß produzieren, mit einer erhöhten Sequenz während des Trainings genutzt werden. Ein weiterer Ansatz ist das Mischen von Beispielen. Der Zweck dieses Vorgehens besteht darin, das Trainingsset so zu verändern, dass aufeinanderfolgende Trainingsbeispiele möglichst selten der gleichen Klasse angehören. (LeCun et al. (2012))

Bei der Verwendung von gemischten Trainingssätzen ist in der Zeitreihenprognose jedoch zu berücksichtigen, dass das Mischen der einzelnen Datenausprägungen möglich ist, jedoch ist es zwingend notwendig, die Sequenzlänge entsprechend der ausgewählten Ausprägung und nicht zufällig zu bestimmen.

Bei der Wahl des Lernverfahrens sollte berücksichtigt werden, dass in Machine Leraning (2018) festgestellt haben, dass der stochastische Gradienten Abstieg eine bessere Performance bei weniger als 1000 Trainingsiterationen liefert. Bei zusätzlichen Iterationen übertrifft die Performance des *BGA* die des *SGD*. Darüber hinaus haben sie gezeigt, dass das Verfahren des Mini-Batch Gradienten Abstiegs beide anderen Verfahren hinsichtlich des Fehlermaßes und der Anzahl an Iterationsschritten, die für die Konvergenz der Verlustfunktion erforderlich sind, übertrifft. Für die in dieser Arbeit betrachteten Netze eignet sich das Verfahren des Mini-Batch Lernens. Die Aufteilung des Trainingsdatensatzes erfolgt in drei Teile, die unterschiedliche Funktionen besitzen (Kapitel 4.5). Das Trainingsset dient dem Lernen des Modells. Am Ende eines jeden Epochs bestimmen die Validierungsdaten die Güte der Modellvorhersagen und der letzte Teil des Datensatzes, das Testset, prüft die Vorhersagen in einem vom Netz noch nicht bearbeiteten Datensatz nach Abschluss des Trainings.

3.3.2 Definition der Batchgröße

Kleine Batchgrößen führen dazu, dass das Netz eine große Anzahl an Parameteranpassungen innerhalb eines Epochs tätigt. Im Kontrast dazu stehen Batches mit einem großen Umfang an Trainingsdaten. Bei diesen werden nur wenige Parameteranpassungen in einem Epoch vollzogen. Dadurch nimmt die Wahl der Batchgröße Einfluss auf das Konvergieren der Verlustfunktion. Bei einer großen Anzahl an Parameteranpassungen in einem Epoch wird die Konvergenz beschleunigt, jedoch konvergiert die Verlustfunktion dabei häufig in einem lokalen Optimum. Dies resultiert daraus, dass der Validierungsdatensatz in einer sehr geringen Frequenz, verglichen mit den Netzanpassungen, getestet wird. Auf der anderen Seite hingegen führt eine geringe Batchgröße dazu, dass innerhalb eines Epochs nur wenige Netzparameter angepasst werden, wodurch das Netz insgesamt eine höhere Anzahl an Epochs benötigt, bis die Verlustfunktion konvergiert. Somit gestaltet sich die Wahl der Batchgröße zu einem Kompromiss zwischen einer besseren Modellperformance und einer höheren Rechenleistung.

Die Aufteilung des Trainingsdatensatzes erfolgt in 10 Batches mit jeweils 262144 Ausprägungen. Dadurch vollzieht das Netz 10 Parameteranpassungen vor dem Performancetest anhand der Validierungsdaten. Zur Beschleunigung des Trainingsprozesses, werden zufällige Ausprägungen aus den Trainingsdaten, zur Füllung der Batches genutzt. Die anderen beiden Datensätze dienen jeweils dem Testen der Netzperformance. Hierfür bekommten die Netze den gesamte Datensatz in einem Batch. Auf das Mischen wird bei diesen beiden Datensätzen verzichtet, sodass das Netz alle Ausprägungen der Datensätze genau einmal, entsprechend des Index, bearbeitet.

Durch die Wahl des Lernverfahrens Mini-Batch Gradienten Abstieg erhält das Netz die Trainingsdaten in definierten Batches. Infolge dessen nimmt die Dimensionalität der Inputdaten weiter zu. Abbildung 18 beschreibt das Format der Input- und Outputdaten, die das Netz vor der Anpassung der jeweiligen Parameter bearbeitet.

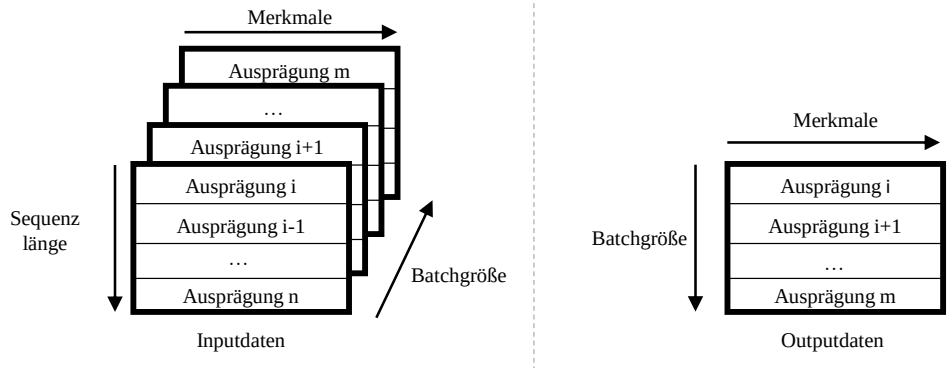

Abbildung 18 – Format Inputdaten

3.4 Modellperformance

Die Netzperformance wird anhand verschiedener Maße getestet. Hierbei kann zwischen zwei Arten der Performance unterschieden werden. Zum einen der Rechenaufwand, welchen das Netz für das Training benötigt, und auf der anderen Seite die Güte der Vorhersagen. Für ersteres bestehen zwei Möglichkeiten. Einmal ist es möglich, die Performance der Netze nach einer definierten Anzahl an Epochs zu vergleichen, oder die Anzahl an Epochs zu zählen, nach denen der Gradient der Verlustfunktion nicht weiter konvergiert. Die hier betrachteten Netze verfügen über die Funktion des frühzeitigen Stopps, weshalb sich das zweite Verfahren anbietet.

3.4.1 Begrifflichkeiten

Zunächst erfolgt die Definition einiger Begrifflichkeiten, die für das Verständnis der Performancemaße erforderlich sind. Einfachheitshalber werden die Begriffe im Kontext einer binären Klassifizierung erklärt. Hierbei gibt es einmal die Klasse *Positiv* und einmal die Klasse *Negativ*. Für Klassifikationsaufgaben mit mehreren Klassen sind manche Begriffe zusätzlich spezifiziert (Foss (2018)).

- **Wahr Positiv:** Ausprägungen deren Klasse *Positiv* ist und die korrekt in die Klasse *Positiv* eingeordnet wurden
- **Falsch Positiv:** Ausprägungen deren Klasse *Negativ* ist und die falsch in die Klasse *Positiv* eingeordnet wurden. Bei mehreren Klassen werden

die Ausprägungen jeweils der entsprechenden falsch eingeordneten Klasse zugeordnet

- **Wahr Negativ:** Ausprägungen deren Klasse *Negativ* ist und die korrekt in die Klasse *Negativ* eingeordnet wurden
- **Falsch Negativ:** Ausprägungen deren Klasse *Positiv* ist und die falsch in die Klasse *Negativ* eingeordnet wurden. Bei mehreren Klassen werden die Ausprägungen jeweils der entsprechenden falsch eingeordneten Klasse zugeordnet

3.4.2 Konfusionsmatrix

Die Konfusiosnmatrix bzw. Fehlermatrix zeigt die Performance eines Modells auf, indem die Vorhersagen mit den tatsächlichen Ergebnissen verglichen werden. Jede Reihe der Konfusionsmatrix stellt die Instanz der vorhergesagten Klasse dar und jede Spalte der Matrix die Instanz der richtigen Klasse. (Minaee (2019)) Die nachfolgende Tabelle soll den Aufbau der Konfusionsmatrix anhand einer Klassifizierungsaufgabe mit 3 Klassen verdeutlichen.

Tabelle 4 – Beispielhafter Aufbau der Konfusionsmatrix

		Tatsächliche Klasse		
		Klasse 1	Klasse 2	Klasse 3
Vorhergesagte Klasse	Klasse 1	60 (Wahr Klasse 1)	5 (Falsch Klasse 2)	10 (Falsch Klasse 3)
	Klasse 2	5 (Falsch Klasse 1)	70 (Wahr Klasse 2)	0 (Falsch Klasse 3)
	Klasse 3	15 (Falsch Klasse 1)	5 (Falsch Klasse 2)	55 (Wahr Klasse 3)

3.4.3 Akkuratheit der Klassifikation

Die Akkuratheit ist ein Maß für die Anzahl an korrekt klassifizierten Beispielen verglichen mit der gesamten Anzahl an Beispielen. Bedingung (3.13) beschreibt die mathematische Berechnung der Akkuratheit. (Foss (2018))

$$\text{Akkuratheit der Klassifizierung} = \frac{\text{Wahr Positiv} + \text{Wahr Negativ}}{\text{Gesamtzahl der Vorhersagen}} \tag{3.12}$$

3.4.4 F1-Score

Das letzte Performancemaß ist der *F1-Score*, der über zwei andere Performancemaße berechnet wird, der *Precision* und dem *Recall*. Die *Precision* misst die richtig gekennzeichneten Beispiele einer Klasse, im Verhältnis zu der Gesamtanzahl an Beispielen, die dieser Klasse zugeordnet wurden.

$$\text{Precision} = \frac{\text{Wahr Positiv}}{\text{Wahr Positiv } + \text{Falsch Positiv}} \tag{3.13}$$

Der *Recall* hingegen beschreibt das Verhältnis der korrekten Vorhersagen einer Klasse und der Gesamtanzahl an vorliegenden Beispielen dieser Klasse.

$$\text{Recall} = \frac{\text{Wahr Positiv}}{\text{Wahr Positiv } + \text{Falsch Negativ}} \tag{3.14}$$

Der *F1-Score* ist das harmonische Mittel aus beiden Größen und berechnet sich über die folgende Formel:

$$\text{F1-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.15}$$

Der Wertebereich des *F1-Score* ist auf das Intervall $[0, 1]$ beschränkt und je höher dessen Wert ist, desto besser ist die Performance des Modells. Da der *F1-Score* versucht, eine Balance zwischen Präzision und Rückruf zu finden, muss immer ein Kompromiss zwischen einer hohen Präzision und einem hohen Rückruf gefunden werden. (Mishra (2018), Minaee (2019))

4 Datenvorbereitung

Wie in Kapitel 2.1.2 beschrieben, basiert der Lernprozess von neuronalen Netzen auf dem Training mit Daten. Deren Vorbereitung ist ein wichtiger und entscheidender Schritt für die Analyse, da deren Qualität einen starken Einfluss auf die Resultate des Netzes besitzt. Richtig vorbereitete Daten sind für das Netz einfacher zu bearbeiten, wodurch die Komplexität der Analyseaufgabe reduziert wird. Im Gegensatz dazu steigern falsch vorbereitete Daten die Komplexität der Aufgaben oder machen diese gar unlösbar. Die Datenvorbereitung ist ein wichtiges Instrument für alle Analyseverfahren. Nichtsdestotrotz profitieren KNN von dieser besonders, da eine gute Datenvorbereitung mehrere positive Auswirkungen auf das Netz hat. Zum einen unterliegt der Analyseprozess einer Beschleunigung, weiterhin reduziert sich die Modellkomplexität und zuletzt nimmt die Generalisierbarkeit zu. (Yu et al. (2005)) Abbildung 19 verdeutlicht diese Effekte, durch die Erweiterung von Abbildung 12 aus Kapitel 3, mit dem Einfluss der Qualität von Trainingsdaten auf die Modellkomplexität und Vorhersagefehler.

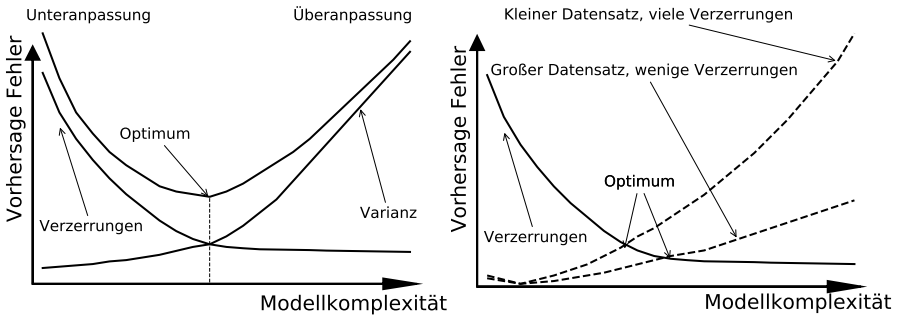

Abbildung 19 – Einfluss von Verzerrungen im Datensatz auf Modellkomplexität und Vorhersagefehler, nach Yu et al. (2005)

Hierfür werden zwei Extremfälle von vorliegenden Daten betrachtet. Zum einen kann der Trainingsdatensatz einen begrenzten Umfang und viele Verzerrungen besitzen. Für solche Datensätze ist eine einfache Modellkomplexität erforderlich

zur Vertmeidung einer Überanpassung. Sofern komplexe Modelle mit solchen Datensätzen trainieren, lernen diese, nicht wie gewünscht ein generelles Modell des Sachverhaltes zu analysieren, vielmehr lernt das Modell die Verzerrungen in den Daten abzubilden. Beim zweiten Extremfall liegt ein umfangreicher Datensatz mit wenig verzerrten Daten vor. Für solche Datensätze sind komplexere Modelle mit vielen Parametern besser geeignet, da diese zu einem geringeren Vorhersagefehler führen. Nicht alle Datensätze entsprechen einer dieser Klassen, sondern sind vielmehr irgendwo zwischen den Extremfällen einzuordnen. Dies erfordert eine individuelle Suche nach der optimalen Modellkomplexität für jeden Datensatz.

Die Datenvorbereitung für die in dieser Arbeit betrachteten Problemstellung der Zeitreihenprognose umfasst die in Abbildung 20 dargestellten Schritte. Die Erläuterung dieser Schritte erfolgt in den nachfolgenden Kapiteln.

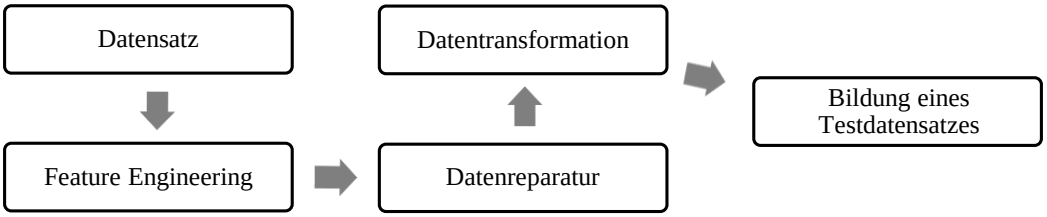

Abbildung 20 – Schritte der Datenvorbereitung

4.1 Datensatz

Der Datensatz wird aus zwei Datenreihen, die jeweils verschiedene Merkmale beinhalten, aufgebaut. Beide Datenreihen verfügen für jede Ausprägung der einzelnen Merkmale über einen Zeitstempel (*eng. timestamp*), welcher die Vereinigung der Merkmale beider Datenreihen ermöglicht.

4.1.1 Fahrraddaten

Die Fahrraddaten beschreiben die Anzahl an Fahrrädern, die an den einzelnen Stationen des VRNnextbike Fahrradvermietsystems in Kaiserslautern vorhanden sind. Eröffnet hat das Vermietungssystem im Juni 2017 mit einem Umfang von

10 Stationen. Nach einer kurzen Zeitspanne ist die Anzahl der Stationen auf 21 gestiegen und im Mai 2019 hat der Betrieb der 22. Station gestartet. Die Stationen sind innerhalb von Kaiserslautern verteilt und unterscheiden freie und feste Stationen (nextbike GmbH). Während freie Stationen das einfache Abstellen von Fahrräder ermöglichen, befinden sich in festen Stationen Fahrradgestelle, zur Fixierung der Räder. Die Anzahl der Fahrradgestellen der festen Stationen variiert (Abbildung 21). Sofern alle Fahrradgestelle einer Station besetzt sind, ist das Abstellen von Fahrrädern an der Station trotzdem noch möglich. In diesem Fall funktioniert die Fahrradabgabe gleich dem Prinzip der freien Stationen. Eine Übersicht über alle Stationen befindet sich im Anhang (Kapitel 7). Zur Zuordnung des Fahrradbestandes innerhalb der Daten zu den einzelnen Stationen, verfügen die Fahrraddaten weiterhin über Längen- und Breitengrade, die den Koordinaten der einzelnen Stationen entsprechen.

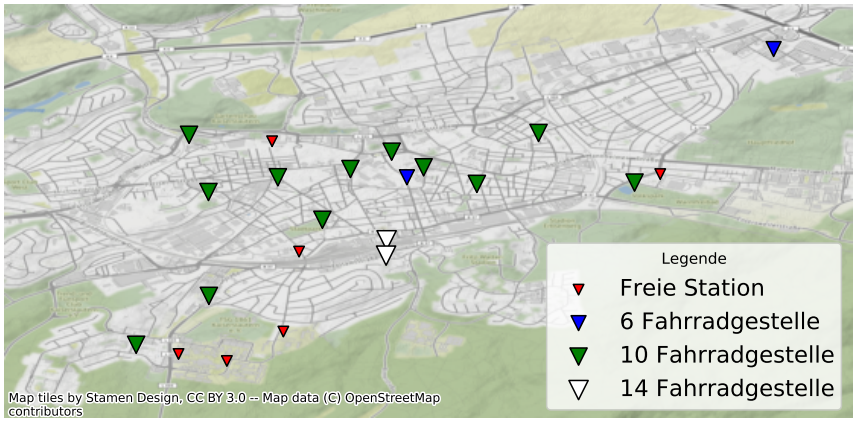

Abbildung 21 – Verteilung Fahrradstationen in Kaiserslautern

Alle Fahrräder innerhalb des Vermietungssystems sind mit GPS ausgestattet, wodurch eine eindeutige Zuordnung zu einer Station stattfindet. Der Bestandes an Fahrrädern, der in jeder Station vorliegt, liegt in fünf minütigen Intervallen vor. Zwar wurde das Fahrradvermietsystem schon im Juni 2017 eröffnet, jedoch liegt die Anzahl der Ausleihen im ersten halben Betriebsjahr bei lediglich 6000. Da die Anzahl der Ausleihen im zweiten halben Betriebsjahr auf über 19.000 gestiegen ist (PressestelleStadtKaiserslautern (2019)), sind die Daten aus dem ersten

halben Jahr wenig repräsentativ und eignen sich nicht als Betrachtungsobjekt. Der früheste Zeitstempel der betrachteten Daten wird auf Anfang Februar 2018 festgelegt und der Datensatz endet Ende November 2019. Eine Ausnahme bildet hierbei die 22. Station, welche im Mai 2019 den Betrieb gestartet hat. Für diese Station liegen nur Daten vom Eröffnungszeitpunkt bis Ende November 2019 vor.

4.1.2 Wetterdaten

Die Wetterdaten stammen von der Internetseite *Openweathermap.org* und beschreiben den Zeitraum vom 01.02.2018 - 31.11.2019. Die Wetterdatenreihe besitzt die folgenden Merkmale, deren Ausprägungen stündlich vorliegen:

- Temperatur [°C]
- Windrichtung [%]
- Windgeschwindigkeit [$\frac{m}{s}$]
- Luftdruck [bar]
- Anzahl Wolken [%]
- Luftfeuchtigkeit [%]

Die Windrichtung beschreibt die meteorologische Windrichtung, bei welcher das Winkelmaß repräsentativ für die Richtung des Windes verwendet wird. Hierbei entspricht 0°/360° der Windrichtung Norden, 90° Osten, 180° Süden und 270° Westen. Die Anzahl der Wolken gibt an, wie bewölkt der Himmel ist. Hierbei entsprechen 100 % einem vollständig bewölkten Himmel und 0 % einem vollständig wolkenfreien Himmel. (Openweathermap)

4.2 Feature Engineering

Trends oder Saisonalitäten in Trainingsdaten können unerwünschte Effekte in der Vorhersageperformance von neuronalen Netzen verursachen. Problematisch ist hierbei, dass Zeitreihen häufig stark von Trends und Saisonalitäten geprägt sind (Yu et al. (2005)).

Eine Möglichkeit die Effekte von Trends und Saisonalitäten zu umgehen, ist deren Kennzeichnung innerhalb der Daten. Dies optimiert den Lernprozess des Netzes, wodurch die Generalisierbarkeit der Netze steigt. (Zhang and Qi (2005)) Hierfür eignet sich das Feature Engineering, welches sich mit der Erweiterung von Trainingsdaten durch zusätzliche Merkmale beschäftigt. Die Zielsetzung besteht in einer effektiveren Gestaltung des maschinellen Lernalgorithmuses (Puget (2017)). Die zusätzlichen Merkmale kennzeichnen die in den Daten vorliegenden Trends

und Saisonalitäten explizit und bilden diese somit innerhalb des Datensatzes ab. Witschel (2014) hat in seiner Arbeit gezeigt, dass Prognosemodelle für Fahrradverleihsysteme neben den Stations- und Wetterdaten häufig noch zeitliche Daten berücksichtigen. Kaltenbrunner et al. (2010) und Froehlich et al. (2009) haben in ihren jeweiligen Arbeiten nachgewiesen, dass innerhalb der betrachteten Fahrradverleihsysteme zeitliche Muster hinsichtlich der Ausleihe von Fahrrädern vorliegen. Begründet ist dies anhand einer veränderten Verfügbarkeit am Wochenende im Vergleich zu den Werktagen.

Aus diesem Grund erhalten die vorliegenden Daten ein zusätzliches Merkmal, die Art des Tages. Hierbei unterscheiden sich Wochenende, Feiertage und Wochentage. Da der Sonntag von den charakteristischen Merkmalen einem Feiertag ähnlicher ist, als einem Samstag, z.B. indem Geschäfte sonntags und an Feiertagen geschlossen sind, samstags jedoch nicht, erfolgt dessen Kennzeichnung ebenfalls als Feiertag. Ein weiteres Argument, weshalb der Sonntag als Feiertag zu kennzeichnen ist, begründet der öffentliche Personennahverkehr (ÖPNV). Fahrradverleihsysteme dienen oftmals als Erweiterung des ÖPNV. Dieser nutzt für Sonn- und Feiertage die gleichen Fahrpläne, die sich oftmals von denen an Samstagen unterscheiden. Zusätzlich zur Art des Tages wird der Tag der Woche als eigenes Merkmal einbezogen.

Eine Kennzeichnung der saisonalen Schwankungen innerhalb der Wetterdaten erfolgtg ebenfalls. Hierfür erhält der Datensatz die beiden Merkmalen Jahreszeiten und Monate. Letzteres beruht auf der Annahme, dass innerhalb der einzelnen Monate einer Jahreszeit unterschiedliche Aktivitäten hinsichtlich der Nutzung des Fahrradverleihsystems vorliegen. Diese verursachen bspw. Semester- oder Schulferien, innerhalb derer die Anzahl der Ausleihen zu- oder abnimmt.

Zeitliche Schwankungen können jedoch nicht nur auf der Ebene von Monaten oder Jahreszeiten auftreten, sondern auch zu unterschiedlichen Tageszeiten. Eine typische Nutzergruppe eines Fahrradverleihsystems sind Berufstätige. Der Fahrtzweck dieser Nutzergruppe beschränkt sich insbesondere darauf, den Weg von und zu dem Arbeitsplatz zu bewältigen. Dies führt zu einer entsprechend höheren Nachfrage in den Morgen- und Abendstunden an Arbeitstagen. (Witschel (2014)) Aus diesem Grund wird die Tageszeit als letztes Merkmal in den Datensatz aufgenommen. Die nachfolgende Tabelle 5 stellt abschließend alle zusätzlichen Merkmale dar, welche im Rahmen des Feature Engineerings in den Datensatz aufge-

nommen werden.

Tabelle 5 – Zusätzliche Merkmale des Feature Engineerings

Datum	Tageszeit	Monat	Jahreszeit	Tag der Woche	Art des Tages
31.05.2018 12:30	12:30	Mai	Frühling	Donnerstag	Feiertag
30.06.2018 12:30	12:30	Juni	Sommer	Samstag	Wochenende
31.07.2018 12:30	12:30	Juli	Sommer	Dienstag	Wochentag
31.08.2018 12:30	12:30	August	Sommer	Freitag	Wochentag
30.09.2018 12:30	12:30	September	Herbst	Sonntag	Feiertag

Der Datensatz ist nun vollständig und umfasst 15 Merkmale, darunter vier Merkmale über die einzelnen Stationen, fünf Merkmale zur Darstellung der Zeit und sechs Merkmale bezüglich des Wetters. Tabelle 6 fasst die einzelnen Merkmale abschließend zusammen.

Tabelle 6 – Übersicht Merkmale

Wetterdaten	Fahrraddaten	Zeitdaten
Temperatur	Stationsart	Jahreszeit
Windgeschwindigkeit	Längengrad	Monat
Luftfeuchtigkeit	Breitengrad	Tag der Woche
Luftdruck	Anzahl Fahrräder	Art des Tages
Wolken		Tageszeit
Windrichtung		

4.3 Datenreparatur

Bevor die Daten im nächsten Schritt in ein für das KNN lesbares Format transformiert werden, wird der Datensatz auf fehlende und fehlerhafte Daten überprüft. Unvollständige oder verzerrte Daten können einen negativen Einfluss auf die Performance von KNN ausüben, indem sie Effekte wie *Overfitting* verursachen und somit die Generalisierbarkeit des Netzes niedrig halten (Muzhou and Lee (2013)). Die Identifikation der fehlenden und fehlerhaften Daten, ist der erste Schritt zur Korrektur des Datensatzes. Zerhari (2016) erläutert in seiner Arbeit verschiedene Ansätze zur Identifikation verzerrter Daten. Zhang et al. (2012) nennen weitere

Verfahren und nutzen eine 3 Schritte umfassende Methode zur Identifikation von Verzerrungen in den Rohdaten eines Verkehrsdatensatzes. Dabei werden zunächst negative Daten gefiltert, durch das Entfernen von Datenpunkten aus dem Datensatz, die aufgrund ihrer Einheit falsch sind. Der zweiten Schritt überprüft den Datensatz auf fehlende Daten und der letzte Schritt beschäftigt sich mit der Identifikation anormaler Daten. Verfahren zur Behebung von Verzerrungen beschreiben u.A. Yu et al. (2005), Zhang et al. (2012), Srinivasan et al. (1992) und Gamberger et al. (2000).

Zunächst wird der Datensatz auf anormale Daten untersucht. Daten sind anormal, sofern deren Ausprägung stark vom Zentrum der Verteilung abweicht (Zhang et al. (2012)). Im Fahrraddatensatz ist dies bei den Ausprägungen des Zeitstempels vorhanden.

Abschließend erfolgt eine Überprüfung hinsichtlich fehlender Daten. Hierbei befinden sich fehlende Werte innerhalb aller Merkmale der Wetterdaten. Diese liegen lediglich zu stündlichen Ausprägungen vor, ungleich den fünfminütigen Intervallen der Fahrraddaten. Die verbleibenden Merkmale sind vollständig. Abschnitt 4.3.2 beschreibt die Handhabung der fehlenden Wetterdaten.

4.3.1 Behandlung verzerrter Daten

In Daten sind zwei verschiedene Arten von Verzerrungen auszumachen. Zum einen können diese inmitten einzelner Merkmale auftreten und zum anderen innerhalb von Klassen. Beim Erstgenannten treten die Verzerrungen bei der Erhebung von Ausprägungen einzelner Merkmale auf. Typische Ursachen bei dieser Art der Verzerrung sind bspw. redundante Daten oder Variablen mit fehlenden Werten. Ursachen für Verzerrungen innerhalb von Klassen liegen häufig in der falschen Zuordnung von Merkmalen zu Klassen, oder der fehlerhaften Transformation von Merkmalsausprägungen zu Klassen. (Gamberger et al. (2000))

Die Ausprägungen der Fahrraddaten liegen zur vollen Stunde und in exakten fünf Minuten Schritten vor, weshalb in den Ausprägungen keine einzelnen Sekunden, oder Minuten außerhalb dieser Intervalle vorliegen dürfen. Da dies nicht der Fall ist (Tabelle 7), liegen Verzerrungen vor. Die zeitlichen Verschiebungen sind durch die Erhebung der Fahrraddaten begründet. Bei dieser wurde der Zeitstempel nach dem Upload der Daten in eine Datenbank hinzugefügt. Hierbei

sind Verzögerungen durch eine schlechte Internetverbindung, oder einen hohen Datenverkehr entstanden. Die Behebung der Verzerrungen erfolgt in zwei Schritten. Der erste überprüft die Sekundenausprägungen des Zeitstempels, stellt die Abweichung fest und subtrahiert diese abschließend vom Zeitstempel. Der zweite Schritt wiederholt diesen Vorgang auf der Basis von Minutenwerten.

Tabelle 7 – Bereinigung der Fahrraddaten

Zeitstempel	Verzerrtes Datum	Modulo_Sekunde	Modulo_Minute	Richtiges Datum	Fahrräder
1525008241	2018-04-29 13:24:01	1	240	2018-04-29 13:20:00	4
1525008541	2018-04-29 13:29:01	1	240	2018-04-29 13:25:00	4
1525008841	2018-04-29 13:34:01	1	240	2018-04-29 13:30:00	4
1525009141	2018-04-29 13:39:01	1	240	2018-04-29 13:35:00	4

4.3.2 Behandlug fehlender Werte

Um den Datensatz in ein vollständiges Datenformat zu übertragen, werden Daten häufig entfernt oder imputiert. Betreffen die fehlenden Daten lediglich einen kleinen Teil, kann das Entfernen dieser Daten eine gute Lösung sein. Sofern die fehlenden Daten größere Teile des Datensatzes und ggf. mehrere Merkmale umfassen, bilden diese häufig einen essentiellen Teil der Daten und das Entfernen würde folglich einen großen Informationsverlust bedeuten. Die Alternative zum Entfernen, die Datenimputation, ist nicht weniger problematisch. Der Begriff der Imputation beschreibt das Beheben von fehlenden Werten durch die Generierung von plausiblen Werten bspw. durch statistische Methoden. Zwar werden die Daten nach der Erzeugung häufig als echte Werte behandelt, jedoch können die erzeugten Werte die im Datensatz vorliegenden Korrelationen beeinflussen und somit die Daten verzerren. (Schafer (1997))

Der einfachste Weg, die fehlenden Werte zu beheben, besteht in dem Ignorieren dieser und der alleinigen Nutzung der Fahrraddaten zur vollen Stunde. Die Konsequenz daraus wäre jedoch der Verlust von über 90% der Fahrraddaten. Zur Vermeidung des Verlustes erfolgt eine Imputation der Wetterdaten. Fraglich ist hierbei welche Methode der Imputation am besten geeignet ist. Aufgrund der stündlich vorliegenden Werte sind der Trend, sowie die Saisonalität der Zeitreihen innerhalb der Wetterdatenreihe grob beschrieben und die Imputation verzerrt diese nicht. Da der Zyklus einiger der Wetterdaten mehr auf Tages- als auf

Stunden-ebene variiert, z.B. die Temperatur steigt morgens an, bleibt über Mittag auf einem hohen Niveau, sinkt abends ab und bleibt in der Nacht auf einem niedrigen Niveau, wird die Verzerrung dieser Bewegungskomponente ebenfalls niedrig gehalten. Aus diesem Grund eignet sich für die Imputation der Wetterdaten die statistische Methode der linearen Regression. Dafür werden die fehlenden Daten, basierend auf den Werten der vorherigen und nachfolgenden vollen Stunde erzeugt. Zwar ist diese Methode nicht in der Lage die irreguläre Komponente der Wetterdatenreihe abzubilden, jedoch ist generell fraglich, inwiefern andere Methoden der Imputation diese, aufgrund ihres Zufallscharakters, berücksichtigen können. Die folgende Abbildung fasst das Problem der Imputation der Wetterdaten anhand des Merkmals Windgeschwindigkeit abschließend zusammen.

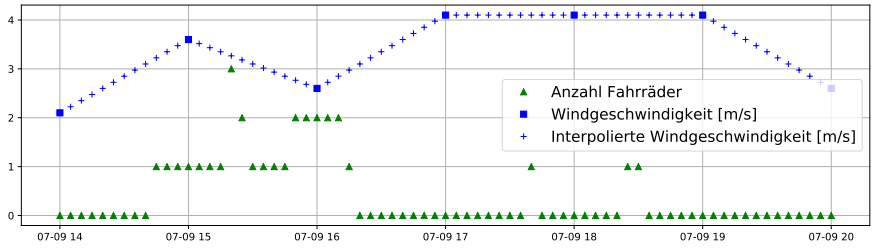

Abbildung 22 – Reparatur der Wetterdaten am Beispiel Windgeschwindigkeit

4.4 Datentransformation

Die Datentransformation beschäftigt sich mit der Normalisierung der Werte der Rohdaten auf Werte in einem Intervall $[0, 1]$ bzw. $[-1, 1]$. Dies erhöht die Performance der Netze, durch die Transformation kontinuierlicher numerischer Eingaben und der Anpassung deren Verteilung dahingegen, damit diese zu der Verteilung der Outputvariablen passt (Shi (2000)). Die Problemstellung der Datentransformation bezieht sich nicht ausschließlich auf neuronale Netze, sondern findet zudem in der Statistik bei der Analyse und der Untersuchung von Daten Anwendung. Die Datentransformation nimmt einen wichtigen Bestandteil der Datenvorbereitung ein und wirkt sich dabei positiv auf die Lernfähigkeit der Netze aus. Dies kann ein entscheidendes Charakteristikum bei der Lokalisierung von Mustern in den Daten sein. (Lacroix et al. (1997)) Verschiedene Autoren haben in Studien nachgewiesen, dass die Performance von Netzen mit transformierten

Trainingsdaten, der Performance von Netzen ohne normalisierte Daten überlegen ist (Vgl. Lacroix et al. (1997), Cannas et al. (2006), Shanker et al. (1996)).

Die Datentransformation ist ein mehrere Schritte umfassender Prozess (Yale (1997)). Hierbei erfolgt eine Vorverarbeitung der Inputdaten auf eine Art und Weise, damit die Inputvariablen eine gleichermaßen verteilte Bedeutung erhalten. Entscheidend ist, dass Inputvariablen mit einem höheren absoluten Wert die gleiche Bedeutung beschreiben, wie die entsprechenden Inputvariablen mit kleineren absoluten Werten. Erreicht wird dies, indem die Verteilung eines Inputmerkmals nach der Transformation der Verteilung von zuvor entspricht (Prasad and Beg (2009)). Abhängig von der Art der Daten erfolgt die Transformation auf verschiedene Arten. Unterschieden wird hierbei zwischen der Transformation kategorischer und numerischer Daten (Lacroix et al. (1997)).

Die Existenz kategorischer Werte beeinflusst das Lernproblem dahingegend, dass die Zielvariable nicht ausschließlich von numerischen Werten abhängig ist, sondern einem zusätzlichen Einfluss von Variablen mit anderen Verteilungen unterliegt. Da neuronale Netze ausschließlich auf der Basis numerischer Inputvektoren arbeiten, ist eine Umwandlung anders verteilter Werte zu numerischen Werten erforderlich. (Potdar et al. (2017))

4.4.1 Behandlung kategorischer Werte

Die Literatur betrachtet zahlreiche Verfahren zur Transformation kategorischer Werte (Vgl. Potdar et al. (2017), Cerda et al. (2018), Ru et al. (2019)). Ein mögliches Verfahren ist die *one-hot encoding* Methode. Diese transformiert kategorische Merkmale in eine *one-hot encoding* Verteilung, welche eine Variable mit N Ausprägungen in eine N dimensionale Matrix, mit lediglich einem Wert ungleich Null pro Zeile, überträgt (Ru et al. (2019)). Die Problematik des Verfahrens besteht in der starken Erhöhung der Dimension des Datensatzes, wodurch der Rechenaufwand der Netze steigt. Ein Verfahren, das diese Problemstellung umgeht, ist das *hash encoding*. Dieses stellt die einzelnen Klassen, anders als beim *one-hot encoding*, nicht als einen eigenen N dimensionalen Vektor dar, sondern verwendet eine *hash Funktion* zur Abbildung eines Klassenvektors durch eine kleinere Zielmenge. (Cerda et al. (2018))

One-hot encoding

Im Datensatz liegen verschiedene Merkmale vor, die kategorische Werte aufweisen. Tabelle 8 soll die Vorgehensweise des *one-hot encodings* anhand des Merkmals Jahreszeit verdeutlichen. Die nachfolgende Tabelle 9 stellt alle Merkmale des Datensatzes dar, die eine solche Transformation erfahren, sowie die neuen Dimensionen im Datensatz, die hierbei entstehen.

Tabelle 8 – Funktionsweise des one-hot encodings

Jahreszeit	**Frühling**	**Sommer**	**Herbst**	**Winter**
Frühling	1	0	0	0
Sommer	0	1	0	0
Herbst	0	0	1	0
Winter	0	0	0	1

Tabelle 9 – Übersicht der mit der one-hot encoding Methode transformierten Merkmale

Merkmal	**Neue Dimensionen nach der Transformation**			
Jahreszeit	Frühling	Sommer	Herbst	Winter
Monat	Januar	Februar	März	April
	Mai	Juni	Juli	August
	September	Oktober	November	Dezember
Wochentag	Montag	Dienstag	Mittwoch	Donnerstag
	Freitag	Samstag	Sonntag	
Art des Tages	Wochentag	Feiertag	Wochenende	
Windrichtung	Nord	Nordwest	West	Südwest
	Süd	Südost	Ost	Nordost

Ein weiteres kategorisches Merkmal im Datensatz ist die Art der Station. Da dieses Merkmal lediglich zwei Klassen beschreibt, ist eine Transformation durch das *one-hot encoding* nicht erforderlich. Stattdessen werden die unterschiedlichen Klassen innerhalb des Merkmals beschrieben. Hierbei werden die freien Stationen mit einer 0 gekennzeichnet und feste Stationen durch eine 1. Nach der Transformation der kategorischen Merkmale ist die Dimension des ursprünglichen Datensatzes von 15 Spalten auf 44 Spalten angestiegen.

Hash encoding

Das letzte kategorische Merkmal im Datensatz ist die Tageszeit. Problematisch bei der Verwendung des *one-hot encodings* ist jedoch die starke Erhöhung der Dimension des Datensatzes. Die Tageszeit weist zum einen Minutenwerte in 5 minütigen Abständen auf und zum anderen jede der 24 Stunden des Tages. Infolge dessen erzeugt die Abbildung der Minutenzeit 12 Dimensionen und die Abbildung der Stunden 24 weitere Merkmale. Stattdessen wird eine *hash Funktion* zur Transformation verwendet. Zeit ist ein zyklisches Maß, weshalb die Transformation mittels einer linearen Funktion, welche die Sekunden des Tages über den Tagesverlauf aufsummiert, das folgende Problem verursacht. Das Netz erhält bei der Uhrzeit 23 : 00 Uhr einen hohen Wert und für die Uhrzeit 01 : 00 einen niedrigen Wert, woraus es schließt, dass sich die erste Uhrzeit stark von der zweiten unterscheidet. Jedoch liegen beide Uhrzeiten lediglich eine Stunde von Mitternacht entfernt. Aus diesem Grund muss die Wahl der *hash Funktion* auf eine Funktion fallen, die die zyklische Komponente der Zeit darstellt.

London (2016) empfiehlt die Verwendung einer Sinus und Cosinus Funktion, da diese ebenfalls einen zyklischen Charakter besitzen. Zur Transformation erfolgt die Umwandlung der Tageszeit in Minuten nach Mitternacht. Diese Werte werden anschließend mittels der *hash Funktion* transformiert.

Hierfür kommt sowohl eine Sinus als auch eine Cosinus Funktion zum Einsatz (Abbildung 23). Bei der Verwendung von nur einer der Funktionen, ergibt sich die Problemstellung, dass einzelne, identische Werte verschiedene Uhrzeiten beschreiben (Abbildung 23, links). Durch die zweite Funktion ist diesen doppeldeutigen Werten ein zweiter Wert zugeordnet, der jede Tageszeit eindeutig kennzeichnet (Abbildung 23, Mitte). Die Abbildung der Sinus- und Cosinuswerte der transformierten Tageszeit in einem Koordinatensystem erzeugt eine kreisförmige Funktion (Abbildung 23, rechts), die auf der einen Seite den zyklischen Charakter der Zeit darstellt und auf der anderen Seite jede Tageszeit eindeutig beschreibt. Durch die Transformation des Merkmals Tageszeit hat der Datensatz nochmals eine zusätzliche Dimension erhalten und umfasst endgültig 45 Merkmale.

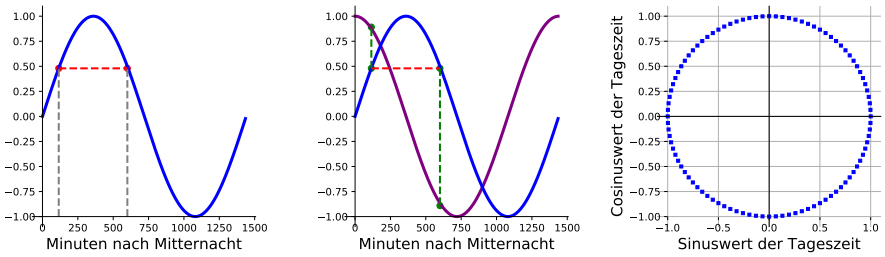

Abbildung 23 – Transformation des Merkmals Zeit des Tages

4.4.2 Datennormalisierung

Der Nutzen der Datennormalisierung liegt insbesondere in der Beschleunigung der Konvergenz der Verlustfunktion. Langsames Konvergieren resultiert allem voran aus einer schnellen Sättigung der Aktivierungsfunktion bei hohen Eingabewerten, wodurch lediglich kleine Gradientenänderungen bei der Rückwärts-rechnung möglich sind. Die Normalisierung der Daten auf kleine Werte beschleunigt das Training durch größere Gradientenände-rungen. Die Normalisierung der Werte erfolgt auf ein Intervall $[0, 1]$ bzw. $[-1, 1]$. Bei der Wahl des Wertebereichs nach der Normalisierung ist zu berücksichtigen, dass dieser dem Wertebereich der Aktivierungsfunktion entspricht. (Amiruddin et al. (2018)) Darüber hinaus beschreiben LeCun et al. (2012) einen weiteren Ansatz der Datennormalisierung, der das Konvergieren der Verlustfunktion weiter beschleunigt. Über mehrere Schritte transformiert er die Ausprägungen der einzelnen Merkmale auf einen Wert, der möglichst nah an dem Wert 0 liegt. Die Normalisierung numerischer Inputwerte ist eine Standardprozedur bei der Datenvorbereitung. Die Literatur diskutiert hierfür eine Vielzahl an Verfahren, die oftmals weniger aufwendig sind, als das Verfahren von LeCun et al. (2012). Ein häufig verwendetes Verfahren ist die *lineare Transformation*. Bei dieser erfolgt die Transformation der Inputwerte eines Merkmals, auf der Basis der minimalen und maximalen Ausprägung, und bildet diese auf dem Intervall $[0, 1]$ ab (Bowden et al. (2003); Jolai and Ghanbari (2010)). Weitere Verfahren sind bspw. statistische Verfahren wie der *Z-Score* (Jolai and Ghanbari (2010)), logarithmische Verfahren (Bowden et al. (2003)), oder dezimal basierte Verfahren (Patro and Sahu (2015)).

Die Transformation der numerischen Werte im vorliegenden Datensatz basiert auf dem Ansatz von LeCun et al. (2012), jedoch weicht die Bearbeitung ab (Ab-

bildung 24). Nach der Transformation des Erwartungswertes der Datenreihe auf den Wert 0, unter Berücksichtigung, die in den Daten vorliegende Werteverteilung nicht zu verändern, erfolgt im zweiten Schritt die Transformation des Wertebereiches auf das Intervall $[-1, 1]$, sodass der Erwartungswert der Ausprägungen weiterhin dem Wert 0 entspricht. Die folgenden Merkmale erfahren eine Normalisierung nach diesem Prinzip:

- Temperatur [°C]
- Wolken [%]
- Längengrad[°]
- Breitengrad[°]
- Luftfeuchtigkeit [%]
- Windgeschwindigkeit $[\frac{m}{s}]$
- Luftdruck [bar]

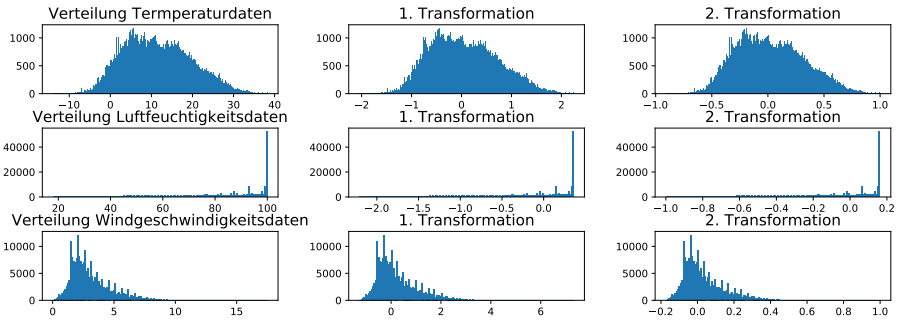

Abbildung 24 – Prinzip der Transformation numerischer Werte

Nun sind alle Merkmale, abgesehen von der Anzahl an Fahrrädern normalisiert. Die Anzahl an Fahrrädern ist das entscheidende Merkmal dieser multivariaten Zeitreihe, aus dessen Ausprägung das Netz den Bestand einer bestimmten Station prognostizieren soll. Damit das Netz hierbei die bestmöglichen Informationen nutzen kann, wird dieses Merkmal bewusst nicht transformiert. Je nach Aktivierungsfunktion kann dies zu dem Problem führen, dass die Werteverteilung des Merkmals nicht dem Wertebereich der Aktivierungsfunktion entspricht. Da die *ReLU* Aktivierungsfunktion jedoch nur im unteren Wertebereich auf den Wert 0 beschränkt ist und über keine obere Schranke verfügt, entspricht dieser Wertebereich den Fahrraddaten.

4.5 Bildung eines Testdatensatzes

Der letzte Schritt der Datenvorbereitung beschäftigt sich mit der Aufteilung des Datensatzes in ein Trainings- und ein Testset. Ersteresdient dem Training des

Netzes und letzteres testet die Generalisierbarkeit des Netzes. Raschka and Mirjalili (2019) empfehlen den Datensatz in drei statt zwei Sets zu untergliedern (Abbildung 25). Hierbei unterscheiden sich Trainings-, Validierungs- und Testdaten. Das Netz trainiert zunächst mit den Trainingsdaten. Die Performance des Netzes wird anschließend, iterativ über den Lernprozess nach einer definierten Anzahl an Trainingsdaten, anhand der Validierungsdaten bestimmt. Der Testdatensatz findet erst nach der Beendigung des Trainings Anwendung. Da das Modell die Testdaten noch nicht bearbeitet hat, kann anhand dieses Datensatzes ein finaler Test der Modellperformance stattfinden. Hierfür berechnet das Netz für jeden Datenpunkt eine Vorhersage, welche anschließend mit dem tatsächlichen Ergebnis verglichen wird.

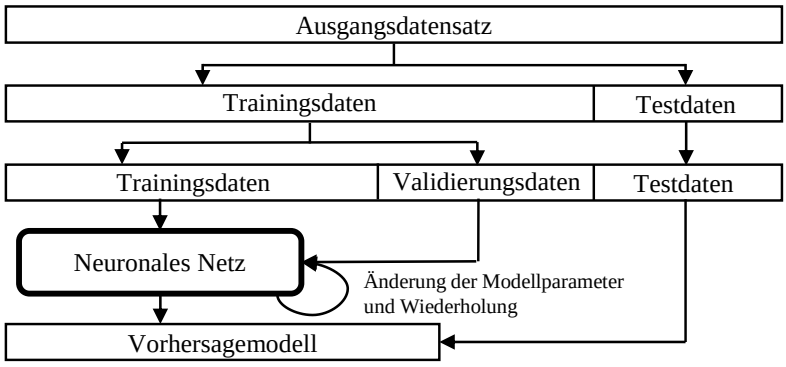

Abbildung 25 – Aufteilung des Datensatzes, nach Raschka and Mirjalili (2019)

Der Vorteil von drei Datensätzen besteht in der Bestimmung der Generalisierbarkeit des Netzes in der Trainingsphase anhand der Validierungsdaten. Zusätzlich hat das Netz den Testdatensatz während des Trainings noch nicht bearbeitet. Dadurch lässt sich die Netzperformance abschließend anhand eines unbekannten Datensatzes bestimmen.

Der Datensatz beschreibt in die folgenden drei Teile. Zeitlich liegen Daten vom 05.02.2018-12:25:00 bis zum 29.12.2019-12:10:00 vor. Die Trainingsdaten umfassen 70% des Datensatzes und die Validierungs- und Testdaten jeweils 15%.

- **Trainingsdaten:** 05.02.2018-12:25:00 bis 29.05.2019-03:45:00
- **Validierungsdaten:** 29.05.2019-03:50:00 bis 28.08.2019-21:45:00
- **Testdaten:** 28.08.2019-21:50:00 bis 29.11.2019-12:10:00

Die Zielvariable der Datenreihe ist die *Anzahl an Fahrrädern*, die zu einem definierten Zeitschritt n in der Zukunft, an einer bestimmten Station vorliegt. Infolgedessen beschreibt das Merkmal *Anzahl an Fahrrädern* die Zielvariable ohne die zeitliche Verschiebung und bleibt zum Zeitpunkt t als Merkmal im Datensatz bestehen. Anschließend wird das Merkmal aus dem Datensatz kopiert. Nachdem diese gesonderte Datenreihe um die gewünschten n Zeitschritte verschoben ist, entspricht diese der Zielvariable, wodurch allen Merkmalen zum Zeitpunkt t die *Anzahl an Fahrrädern* zum Zeitpunkt $t + n$ zugeordnet ist.

5 Versuche

Im Folgenden werden unterschiedliche Versuche mit verschiedenen Netzstrukturen durchgeführt. Die einzelnen Versuche sollen die Auswirkungen von einem veränderten Datensatz, und damit einem abweichenden Zusammenspiel zwischen Verzerrungen und Varianzen in den Daten, untersuchen. Neben den Änderungen im Datensatz werden zudem verschiedene Netzstrukturen untersucht. Hierbei erfolgt eine Betrachtung der Performance zwischen einem FFN, einem LSTM und einem GRU Netz über alle Versuche hinweg. Weiterhin befassen sich die Versuche mit verschiedenen Modellparametern und Modellarchitekturen. Die Untersuchung der Modellparameter beschränkt sich insbesondere darauf, eine geeignete Anzahl an Epochs bis zum frühzeitigen Stopp zu finden. Weiterhin soll herausgefunden werden, ob für die definierten Netze ein Optimierer mit einer konstanten Lernrate oder ein adaptiver Optimierer besser geeignet ist und welche Auswirkungen eine veränderte Sequenzlänge bewirkt. Neben einer steigenden Modellkomplexität durch zusätzliche Informationen während des Lernens und der Vorhersagen, ist hierbei die Art der vorliegenden zeitlichen Abhängigkeiten in den Daten ein entscheidendes Betrachtungsobjekt.

5.1 Vorhersagen mit allen Daten

Für den ersten Versuch werden jeweils ein FFN, ein LSTM und ein GRU Netz darauf trainiert, Vorhersagen über die Verfügbarkeit von Fahrrädern an den 22 verschiedenen Stationen des Fahrradverleihsystems VRNextbike in Kaiserslautern zu tätigen. Die verfügbaren Fahrräder pro Station werden in 11 verschiedene Klassen unterteilt, sodass die Vorhersage einer Klasse im Intervall [0,10] entspricht. Hierfür werden alle Beobachtungen, bei denen mehr als 10 Fahrräder an einer Station vorhanden sind, der Klasse 10 zugeordnet. Für die Vorhersagen werden alle Merkmale des in Kapitel 4 vorgestellten Datensatzes verwendet.

Modellarchitektur

Die Modellarchitektur entspricht den in Kapitel 3 vorgestellten Modellen. Die Architektur der einzelnen Netze ist in Tabelle 10 nochmal zusammengefasst.

Tabelle 10 – Modellarchitektur Versuch 1

	Modell	**FFN**	**LSTM**	**GRU**
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	45	45	45
	Aktivierungsfunktion	-	-	-
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
	Aktivierungsfunktion	ReLU	tanh	tanh
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	46	46
	Aktivierungsfunktion	ReLU	ReLU	ReLU
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	11	11	11
	Aktivierungsfunktion	Softmax	Softmax	Softmax

Modellparameter

Die Modellparameter entsprechen ebenfalls den in Kapitel 3 vorgestellten Parametern. Tabelle 11 gibt eine Übersicht über die Einstellung der Parameter. Hierbei ist zu beachten, dass zum einen der *ADAM* Optimierer für die beiden RNN verwendet wird, und zum anderen, dass die rekurrenten Netze zusätzlich über eine längere Sequenzlänge verfügen. Weiterhin wird eine unterschiedliche Anzahl an Epochs bis zum frühzeitigen Stopp verwendet.

Tabelle 11 – Modellparameter Versuch 1

Parameter	FFN	LSTM	GRU
Verlustfunktion	categorical crossentropy	categorical crossentropy	categrical crossentropy
Optimierer	SGD	ADAM	ADAM
Lernrate	0.1	0.1	0.1
Nesterov Momentum	0.9	-	-
Lernverfahren	Mini Batch Gradienten Abstieg	Mini Batch Gradienten Absteig	Mini Batch Gradienten Abstieg
Batches Pro Epoch	10	10	10
Beispiele pro Batch	262144	262144	262144
Epochs vor dem frühzeitigen Stopp	30	100	30
Dropout	-	0.1	0.1
Rekurrenter Dropout	-	0.5	0.5
Sequenzlänge	1	3	3
Gesamtzahl an Modellparametern	13570	50937	38332

Versuchsauswertung

Der Verlauf der Verlustfunktionen im Trainings- und Validierungsdatendatz der einzelnen Netze gibt einen ersten Aufschluss über das Konvergenzverhalten (Abbildung 26). Die Trainings- und Validierungsverlustfunktionen des FFN und GRU Netzes konvergieren beide gegen einen ähnlichen Wert. Anders verhält sich der Verlauf der Funktionen des LSTM Netzes. Bei diesem nähern sich die beiden Funktionen in den ersten 40 Epochs dem gleichen Wert. Anschließend konvergieren beide Funktionen gegen unterschiedliche Werte, wobei sich die Trainingsverlustfunktion einem niedrigerem Wert annähert. Ähnlich ist der Verlauf der Akkuratheit im Trainings- und Validierungsdatensatz. Das Ergebnis des LSTM Netzes weicht hierbei wiederum von den Ergebnissen der anderen Netze ab. Bei diesem nähern sich die jeweiligen Funktionen zunächst dem gleichen Wert an, jedoch konvergieren die beiden Funktionen nach ca 40 Epochs gegen verschiedene Werte, bei welchen die Akkuratheit in den Trainingsdaten wiederum höher ist.

Zusätzlich ist anzumerken, dass sowohl der Verlauf der Trainings- und Validierungsverlustfunktion, als auch die Akkuratheit der Vorhersagen im Trainings- und Validierungsdatensatz, beim FFN fast identisch ist. Anders verhält sich dies beim LSTM und GRU Netz. Während sich die jeweiligen Funktionen beim GRU Netz dem gleichen Wert annähern, konvergieren die Funktionen des LSTM Netzen gegen unterschiedliche Werte. Die Berechnungsdauer der Netze bis zum frühzeitigen Stopp unterscheidet sich stark (Tabelle 12). Hierbei benötigt das LSTM Netz

die meisten Epochs, gefolgt vom FFN, während das GRU Netz das Training nach der geringsten Anzahl an Epochs beendet. Es ist jedoch zu berücksichtigen, dass die frühzeitiger Stopp Funktion des LSTM Netzes erst nach einer deutlich höheren Anzahl an Epochs das Lernen beendet. Sofern die zusätzlichen Epochs des frühzeitigen Stopps von der Anzahl an gelernten Epochs subtrahiert werden, benötigt das LSTM Netz die kleinste Anzahl an Epochs für den Lernprozess und das FFN die höchste. Da dieses Netz jedoch eine deutlich kürzere Berechnungsdauer pro Epoch aufweist, ist der Lernvorgang des FFN Netzes am schnellsten abgeschlossen (Tabelle 12).

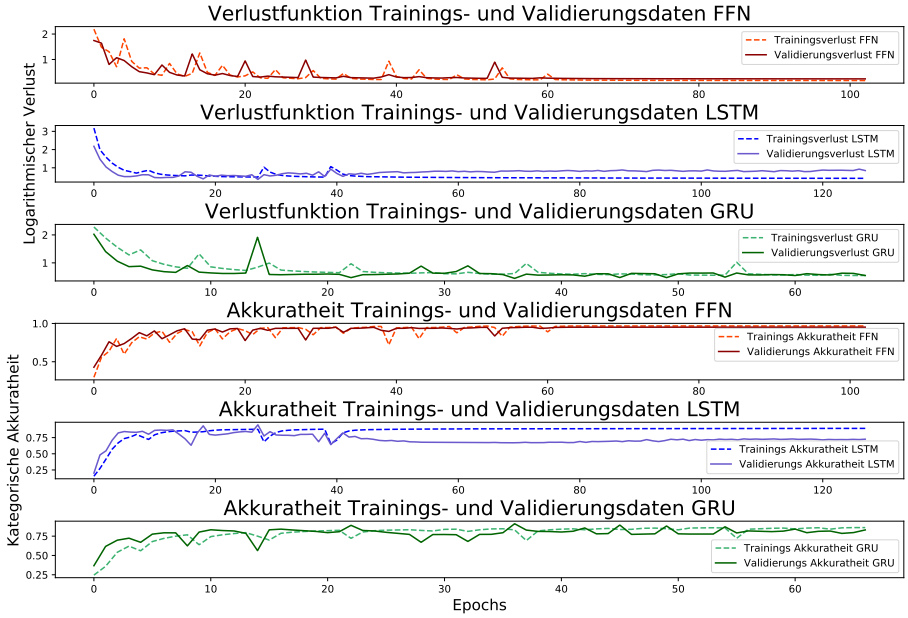

Abbildung 26 – Trainingsdaten Versuch 1

Die Akkuratheit der Vorhersagen im Testdatensatz ist sowohl beim LSTM, als auch beim FFN Netz im sehr guten Bereich (Tabelle 12). Lediglich das GRU Netz liefert ein etwas schlechteres Ergebnis, liegt mit knapp 93% jedoch immer noch in einem guten Bereich. Obwohl sich die Akkuratheit des LSTM und des FFN Netzes lediglich um 1% unterscheiden, zeigt die Konfusionsmatrix, sowie der Klassifikationsreport der beiden Netze (Kap. 7.1), dass die Performance des FFN Netzes, der des LSTM, überlegen ist. Das FFN trifft bei den 563422 Testdaten

lediglich 210 falsche Vorhersagen, wodurch der Klassifikationsreport, bei welchem die Ergebnisse auf 2 Nachkommastellen gerundet werden, ein Ergebnis von 100% für so gut wie jeden Parameter aufweist. Das LSTM Netz hingegen hat insbesondere in den Klassen 6 bis 9 etwas Schwierigkeiten bei der Vorhersage, wodurch die Ergebnisse im Klassifikationsreport von denen des FFN abweichen. Auch die Konfusionsmatrix des LSTM Netzes bestätigt das Ergebnis. Diese zeigt, dass das Netz die Klassen 0-5 sehr gut vorhersagen kann, bei den folgenden Klassen steigt jedoch die Anzahl der Vorhersagefehlern an. Das schlechteste Ergebnis liefert das GRU Netz. Obwohl auch dieses gute Vorhersagen, insbesondere in den Klassen 0-4 tätigt und jeweils ein Ergebnis von über 95% im Klassifikationsreport aufweist, sinkt der Wert in den folgenden Klassen auf ca 90% ab. Zusätzlich trifft das GRU Netz insbesondere für die Klasse 9 sehr viele falsche Vorhersagen, wodurch der Recall der Klasse lediglich bei 3% liegt. Auch die Konfusionsmatrix des GRU Netzes zeigt die schlechtere Performance auf. Hier existiert in jeder Klasse eine weite Streuung an Vorhersagefehlern.

Tabelle 12 – Auswertung Versuch 1

Parameter	**FFN**	**LSTM**	**GRU**
Akkuratheit der Vorhersage	0.9996	0.9895	0.9280
Dauer der Vorhersage [sek]	2.46	11.00	8.62
Epochs bis zum frühzeitigen Stopp	103	129	67
Durchschnittliche Berechnungsdauer pro Epoch [sek]	22	99	93

Bezüglich der Vorhersagedauer ist das FFN Netz den anderen beiden Netzen überlegen, und benötigt etwas weniger als $\frac{1}{3}$ der Zeit für die Prognosen des gesamten Testdatensatzes. Ursachen hierfür liegen zum einen darin, dass die vom LSTM- und GRU Netz bearbeiteten Inputdaten, aufgrund der längeren Sequenzlänge, umfangreicher sind. Hinzu kommt, dass innerhalb der beiden Netze jeweils ein interner Zellzustand ermittelt wird. Da das GRU Netz, verglichen mit dem Lang- und Kurzzeit Zustand der LSTM Zelle, über lediglich einen internen Zustand verfügt, ist die Vorhersagedauer des GRU Netzes etwas kürzer als die des LSTM Netzes. Die in Tabelle 12 dargestellte Zeit wird jedoch für 563418 Vorhersagen benötigt, wodurch die Abweichung der Vorhersagedauer der jeweiligen Netze für ein einzelnes Beispiel minimal ist.

Versuchsergebnis

Obwohl die Performance aller Netze im guten Bereich liegt, ist die des FFN Netzes am Besten. In allen betrachteten Performancemaßen schneidet dieses besser ab. Die schlechtere Performance des LSTM und GRU Netzes kann durch verschiedene Ursachen begründet sein. Zum einen verfügen beide Netze über deutlich mehr Netzparameter, wodurch die Modellkomplexität ansteigt und gemäß Abbildung 12 zusätzliche Vorhersagefehler entstehen. Neben der höheren Modellkomplexität verursachen die zusätzlichen Parameter weiterhin eine längere Berechnungsdauer pro Epoch und eine längere Vorhersagezeit.

Da für das GRU und LSTM Netz eine andere Optimierungsfunktion verwendet wurde, als beim FFN Netz, ist die Konvergenz der Netze nur schwer zu vergleichen. Zum einen ist insbesondere beim GRU Netz nicht klar, ob es in einem lokalen Optimum konvergiert oder in einem globalen. Hier ist es möglich, dass das Lernen zu früh abgebrochen wurde und zusätzliche Epochs dazu führen, dass ein Modell mit einem genaueren Resultat im Testdatensatz gefunden werden kann. Anders ist dies beim LSTM Netz, bei welchem ersichtlich ist, dass das Netz, aufgrund der hohen Anzahl an zusätzlichen Epochs, ab einem gewissen Punkt den Trainingsdatensatz überangepasst. Dadurch sinkt die Performance des Netzes im Validierungsdatensatz. Die Überanpassung zeigt, dass sich das Netz durch die zusätzlichen Epochs nicht weiter verbessert. Hier verdeutlicht sich der Vorteil der Modellcheckpoints, die die beste Netzstruktur speichern und die Überanpassung des LSTM Netzes egalisieren. Die hohe Prozentzahl an richtigen Vorhersagen im Testdatensatz bestätigt dieses Ergebnis.

Die schlechteren Vorhersageergebnisse des LSTM und GRU Netzes in den Klassen $6 - 10$ lassen sich durch eine geringere Anzahl an Trainingsbeispielen für diese Klassen erklären (Abbildung 27). Zusätzlich ist die Anzahl an Testbeispielen, verglichen mit der Anzahl an Validierungsbeispielen, sehr hoch. Insbesondere die Klasse 9, bei welcher alle 3 Netze die meisten Fehler in der Vorhersage aufweisen, hat hier eine extreme Ausprägung. Von der Gesamtanzahl an Beispielen sind lediglich ca 10% im Validierungsdatensatz, jedoch fast 25% im Testdatensatz. Dadurch verfügt der Datensatz bei der Validierung nur über vergleichsweise wenig Beispiele. Während das FFN und LSTM Netz auch für diese Klasse Ergebnisse im sehr guten Bereich liefern, trifft das GRU Netz für so gut wie jedes Beispiel der Klasse eine falsche Vorhersage. Hierbei ist anzunehmen, dass die Performance

aller Netze, durch eine ausgeglichenere Aufteilung des Datensatzes in Trainings-, Validierungs- und Testdaten, wiederum genauere Ergebnisse vorhersagt.

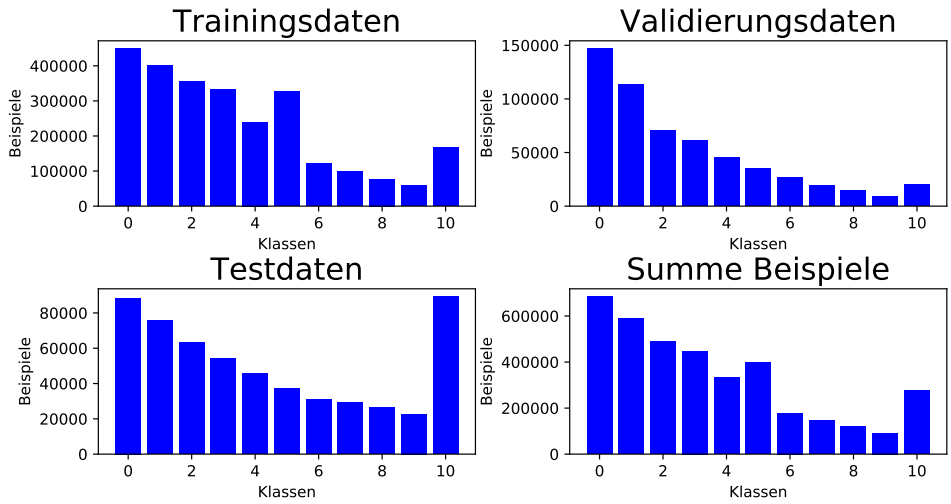

Abbildung 27 – Datenverteilung Versuch 1

5.2 Vorhersagen mit einem verkürzten Datensatz

Die Netze des ersten Versuchs haben allesamt gute Ergebnisse bei der Vorhersage geliefert. Jedoch sind die Ergebnisse der beiden Netze mit einer höheren Modellkomplexität schlechter. Da eine höhere Modellkomplexität zusätzliche Vorhersagefehler verursachen kann, wird in diesem Versuch die Modellkomplexität reduziert, durch die Kürzung des Datensatzes. Hierbei können Merkmale wie bspw. die Windrichtung oder der Tag der Woche lediglich einen marginalen Einfluss auf das Leihverhalten der Nutzer besitzen, wodurch diese Merkmale im Datensatz zum einen Verzerrungen und zum anderen Varianzen bewirken. Ersteres entsteht durch die zusätzlichen Merkmale, die keinen Einfluss auf die Ausleihen nehmen. Varianzen entstehen wiederum durch die höhere Anzahl an Merkmalen.

Die Wetterdaten werden als erstes reduziert. Unter der Annahme, dass insbesondere die Merkmale Temperatur, Luftfeuchtigkeit und Windgeschwindigkeit einen signifikanten Einfluss auf das Leihverhalten von Kunden besitzen, bleiben diese im Datensatz bestehen. Die verbliebenen Wettermerkmale, Windrichtung, Luftdruck und Anzahl an Wolken, werden aus dem Datensatz entfernt. Die Fahrraddaten

verlieren das Merkmal Stationsart, da die Stationen bereits durch die Längen- und Breitengrade voneinander abgegrenzt sind, wodurch jede Station und somit auch das stationsspezifische Leihverhalten bereits durch diese Merkmale charakterisiert ist. Zuletzt erfolgt die Reduktion der Zeitdaten. Die Merkmale dieser Datenreihe dienen insbesondere der Modellierung saisonaler Schwankungen und Trends. Da das Merkmal Jahreszeiten lediglich eine weitere Spezifikation der einzelnen Monate ist und der Tag der Woche eine Spezifikation des Merkmals Art des Tages, werden diese beiden Merkmale aus dem Datensatz entfernt. Die verbliebenen Merkmale sind in Tabelle 13 dargestellt. Deren Transformation entspricht der Datenvorbereitung für den ersten Versuch. Dadurch ergibt sich eine neue Anzahl an Inputmerkmalen, die nach der Transformation dem Wert 23 entspricht.

Tabelle 13 – Übersicht Merkmale verkürzter Datensatz

Wetterdaten	**Fahrraddaten**	**Zeitdaten**
Temperatur	Längengrad	Monat
Windgeschwindigkeit	Breitengrad	Art des Tages
Luftfeuchtigkeit	Anzahl Fahrräder	Tageszeit

Modellarchitektur

Da das FFN Netz im ersten Versuch sehr gute Ergebnisse geliefert hat, wird die Modellstruktur dieses Netzes nur hinsichtlich der Anzahl an Neuronen in der Inputschicht verändert. Diese entsprechen in diesem Versuch wiederum der Anzahl an Merkmalen im Datensatz 23. Das LSTM- und GRU Netz erhalten die gleiche Anpassung. Zusätzlich verändert sich die Anzahl an Neuronen in der versteckten Schicht. Die durch Bedingung 3.2 berechnete Neuronenzahl hat im ersten Versuch bessere Ergebnisse, verglichen mit der durch Bedingung 3.1 berechneten Neuronenzahl, erzeugt. Die Berechnung der Anzahl an Neuronen in der versteckten Schicht des LSTM und GRU Netzes erfolgt nun entsprechend Bedingung 3.2 und beläuft sich somit auf:

$$N_h = 2 * N_i + 1 = 2 * 23 + 1 = 47 \tag{5.1}$$

Tabelle 14 – Modellarchitektur Versuch 2

	Modell	FFN	LSTM	GRU
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	23	23	23
	Aktivierungsfunktion	-	-	-
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	47	47
	Aktivierungsfunktion	ReLU	tanh	tanh
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	47	47
	Aktivierungsfunktion	ReLU	ReLU	ReLU
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	11	11	11
	Aktivierungsfunktion	Softmax	Softmax	Softmax

Modellparameter

Ähnlich zur Modellarchitektur ändern sich die Modellparameter. Aufgrund des besseren Konvergenzverhaltens der *SGD* Verlustfunktion im ersten Versuch, erhalten alle Netze diese Verlustfunktion. Zusätzlich verändert sich die Anzahl der Epochs bis zum frühzeitigen Stopp. Im ersten Versuch hat das LSTM Netz zu viele und das GRU Netz zu wenige Epochs Berechnungszeit gehabt. Die Anzahl an Epochs wird in diesem Versuch auf 50 festgelegt, wodurch der Wert zwischen 30 und 100 Epochs liegt. Zuletzt erfährt die Sequenzlänge des LSTM und GRU Netzes eine Anpassung. Hier hat das FFN Netz im ersten Versuch gezeigt, dass bereits eine Sequenzlänge von einem Zeitschritt für eine nahezu perfekte Vorhersage ausreicht. Diesen Wert erhalten nun auch das LSTM- und GRU Netz.

Tabelle 15 – Modellparameter Versuch 2

Parameter	FFN	LSTM	GRU
Verlustfunktion	categorical crossentropy	categorical crossentropy	categrical crossentropy
Optimierer	SGD	SGD	SGD
Lernrate	0.1	0.1	0.1
Nesterov Momentum	0.9	0.9	0.9
Lernverfahren	Mini Batch Gradienten Abstieg	Mini Batch Gradienten Absteig	Mini Batch Gradienten Abstieg
Batches Pro Epoch	10	10	10
Beispiele pro Batch	262144	262144	262144
Epochs vor dem frühzeitigen Stopp	50	50	50
Dropout	-	0.1	0.1
Rekurrenter Dropout	-	0.5	0.5
Sequenzlänge	1	1	1
Gesamtzahl an Modellparametern	11568	36060	21177

Versuchsauswertung

Das Konvergenzverhalten der jeweiligen Verlustfunktionen ist in diesem Versuch besser als im ersten. Während im vorherigen Versuch lediglich die Verlustfunktion des FFN Netzes ein gutes Konvergenzverhalten aufgewiesen hat, konvergieren in diesem Versuch die Trainings- und Validierungsverlusfunktionen aller Netze gegen den gleichen Wert. Gleichermaßen gilt dies für den Verlauf der Trainings- und Validierungsakkuratheit der einzelnen Netze (Abbildung 28).

Die Akkuratheit der Vorhersagen im Testdatensatz hat sich bei zwei Netzen etwas verbessert und bei einem etwas verschlechtert (Tabelle 16). Während die Akkuratheit des FFN Netzes lediglich um einen Faktor im Bereich 10^{-3} gestiegen ist, hat sich die Akkuratheit in der Vorhersage des GRU Netzes um mehr als 3% Prozentpunkte verbessert. Die Akkuratheit des LSTM Netzes ist hingegen um etwa 3% gesunken. Ein Blick auf die Konfusionsmatritzen gibt weitere Aufschlüsse über die Performance der Netze (Kapitel 7.2). Hierbei wird ersichtlich, dass das FFN lediglich 20 Beispiele falsch vorhergesagt hat und somit ein nahezu perfektes Ergebnis im Testdatensatz erzielt. Obwohl die Akkuratheit der Vorhersagen des LSTM Netzes gesunken ist, hat sich die Performance des Netzes in den ersten 9 Klassen verbessert, indem die Streuung der Vorhersagefehler geringer ist. Die schlechtere Gesamtperformance des Netzes lässt sich insbesondere durch die Klasse 9 erklären. Die Erkennung dieser ist im ersten Versuch sehr gut gewesen mit einem *F1-Score* von 93%. Dieser sinkt im 2. Versuch auf

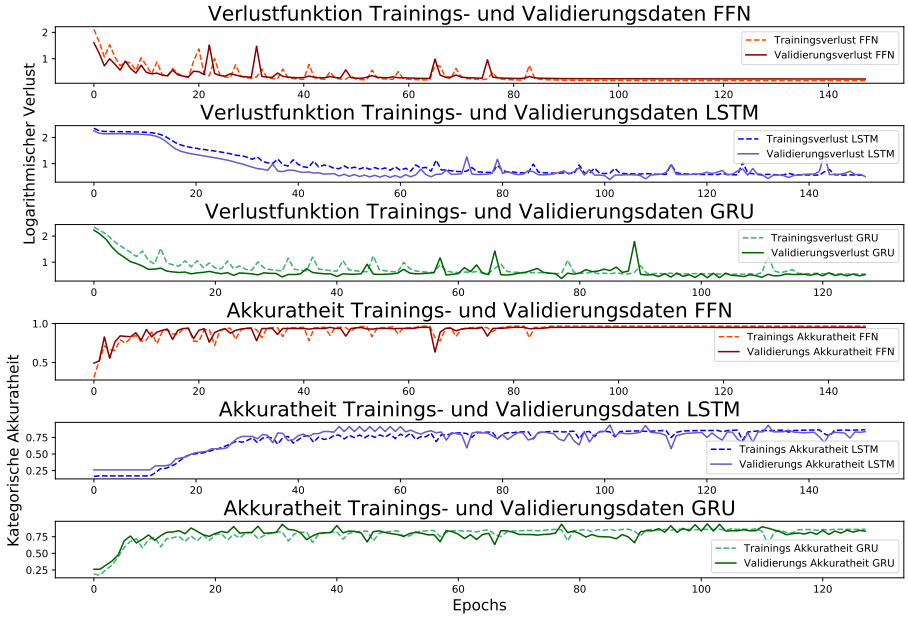

Abbildung 28 – Trainingsdaten Versuch 2

0% (Klassifikationsreport). Das Ergebnis der Klasse 10 liegt, anders als bei den verbliebenen Klassen, nicht bei 100%. und die *Precision* dieser Klasse bei lediglich 80%. Die Zuordnung aller Beispiele der 9. Klasse zu der Klasse 10 ist die Ursache dafür. Der Klassifikationsreport des GRU Netzes ist identisch zu dem des LSTM Netzes. Während alle Beispiele der 9. Klasse falsch in die Klasse 10 eingeordnet werden, entspricht der *F1-Score* der verbliebenen Klassen 100%. Die Konfusionsmatrix erklärt zudem das verbesserte Gesamtergebnis auf Grund einer stark verringerten Streuung der Vorhersagefehler.

Während die Vorhersagedauer des FFN Netzes sich nur marginal verbessert, verkürzt sich die Zeit beim LSTM- und GRU Netzs jeweils um etwas mehr als die Hälfte. Hierfür ist zum einen die verringerte Anzahl an Parametern im Netz verantwortlich und zum anderen bewirkt die verringerte Sequenzlänge eine beschleunigte Ermittlung des internen Zellzustandes. Ähnlich beeinflusst dies den Lernvorgang der Netze. Während die durchschnittliche Berechnungsdauer pro Epoch im FFN nur im geringen Maße verbessert wird, sinkt die Berechnungsdauer bei den anderen beiden Netzen auf etwas weniger als $\frac{1}{3}$ verglichen mit dem Ergebnis des ersten Versuchs.

Tabelle 16 – Auswertung Versuch 2

Parameter	FFN	LSTM	GRU
Akkuratheit der Vorhersage	0.9999	0.9593	0.9595
Dauer der Vorhersage [sek]	2.32	4.71	4.26
Epochs bis zum frühzeitigen Stopp	148	152	128
Durchschnittliche Berechnungsdauer pro Epoch [sek]	18	31	31

Versuchsergebnis

Auch in diesem Versuch ist die Performance des FFN Netzes am Besten, indem die Berechnung der Vorhersagen ca 4% genauer ist. Die Performance der beiden anderen Netze ist nahezu identisch und liegt mit etwas mehr als 95% Akkuratheit auch in einem guten Bereich. Durch die Reduzierung der Netzparameter und der damit verbundenen einfacheren Modellkomplexität, beschleunigt sich der Trainingsprozess der Netze deutlich und zudem hat sich die Vorhersagezeit verkürzt. Zusätzlich berechnen sowohl das LSTM, als auch das GRU Netz die vorliegende Problemstellung mit einer kurzen Sequenzlänge gut. Hierdurch kommt die Vermutung auf, dass das Leihverhalten der Kunden des Verleihsystems nicht auf Langzeit Abhängigkeiten beruht, sondern insbesondere durch Kurzzeit Abhängigkeiten geprägt ist.

Durch die Festlegung einer gleichen Anzahl an Epochs bis zum frühzeitigen Stopp liegt die Gesamtzahl an benötigten Epochs für das Training bei allen Netzen in einem ähnlichen Bereich. Das GRU Netz benötigt jedoch mit 128 Epochs, über 20 Epochs weniger, verglichen mit den anderen Netzen. Wird die Berechnungsdauer der jeweiligen Epochs mitbetrachtet, ist das Training des FFN Netzes am schnellsten beendet. Neben der reduzierten Modellkomplexität lässt sich zusätzlich die gewählte Optimierungsfunktion für die geringere Lerndauer ausmachen. Der Verzicht auf einen adaptiven Lernalgorithmus bewirkt zum einen eine schnellere Konvergenz der Verlustfunktionen und zum anderen eine geringere Berechnungsdauer der Parameteranpassungen.

Neben der Anzahl und der Verteilung der Beispiele pro Klasse im Test- und Validierungsdatensatz, führt dieser Versuch zu einer weiteren Annahme, durch

welche sich die schlechte Performance der Netze bei der Vorhersage der Klasse 9 erklären lässt. Unter der Annahme, dass das Zuordnen aller Beobachtungen von mehr als 10 Fahrrädern an einer Station in die Klasse 10, die Beispiele der Klasse stark verzerrt, besteht die Möglichkeit, dass das eigentliche Problem der Netze bei der Vorhersage dieser Klasse nicht ausschließlich durch eine geringe Anzahl an Validierungs- und eine hohe Anzahl an Testbeispielen begründet ist. Das Zuordnen aller Beispiele der 9. Klasse zu der 10. Klasse, lässt darauf schließen, dass der Berechnungsgraph die Klasse 10 über- und dadurch die Klasse 9 unterangepasst hat. Dieser Sachverhalt wird in Versuch 3 genauer untersucht.

5.3 Vorhersagen mit wenigen Daten und zusätzlichen Klassen

Da die Ergebnisse des 2. Versuchs gezeigt haben, dass die Netze im verkürzten Datensatz ähnlich gute Ergebnisse liefern, dabei jedoch weniger Rechenleistung und weniger Netzparameter benötigen, nutzt der dritte Versuch wiederum die gleichen Datenmerkmale. Veränderungen erfahren lediglich die Fahrraddaten im Datensatz, sodass diese nicht wie zuvor in 11 unterschiedlichen Klassen vorliegen, sondern die Klassen auf 15 festgelegt werden. Hierdurch testet der Versuch die Performance der Netze in einem Datensatz mit einer erhöhten Anzahl an Varianzen. Zusätzlich wird insbesondere das Ergebnis der Vorhersagen für die Klassen 9 und 10 überprüft, um zu testen, inwiefern das schlechte Ergebnis des vorherigen Versuchs in diesen Klassen durch Verzerrungen im Datensatz entstanden ist.

Modellarchitektur

Die Modellarchitektur der Netze bleibt weitestgehend unverändert. Entsprechend der Anzahl an Klassen erhöhen sich Outputneuronen von 11 auf 15. Die Datenmerkmale bleiben unverändert, sodass wiederum 23 Inputneuronen vorhanden sind. Auch die Neuronenanzahl in der versteckten Schicht bleibt unverändert. Der Grund dafür besteht darin, dass sich die Verzerrungen des Datensatzes durch eine präzisere Abbildung der einzelnen Klassen zwar reduzieren, jedoch entstehen durch die zusätzlichen Klassen Varianzen. Die Reduktion der Verzerrungen empfiehlt zwar eine höhere Komplexität der Modelle, doch die angestiegene Zahl

der Varianzen deutet darauf hin, dass eine schlichtere Modellstruktur von Vorteil ist.

Tabelle 17 – Modellarchitektur Versuch 3

	Modell	**FFN**	**LSTM**	**GRU**
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	23	23	23
	Aktivierungsfunktion	-	-	-
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	47	47
	Aktivierungsfunktion	ReLU	tanh	tanh
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	91	47	47
	Aktivierungsfunktion	ReLU	ReLU	ReLU
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	15	15	15
	Aktivierungsfunktion	Softmax	Softmax	Softmax

Modellparameter

Die Modellparameter bleiben ebenfalls weitestgehend unverändert. Lediglich die Sequenzlänge des LSTM- und GRU Netzes wird angepasst und auf den Wert 2 gesetzt. Neben der erhöhen Anzahl an Outputneuronen, nimmt auch die Sequenzlänge einen Einfluss auf die Gesamtzahl an Modellparametern. Während die Modellparameter im FFN und LSTM Netz lediglich um einen dreistelligen Wert erhöht werden, steigen diese im GRU Netz um über 6000 Parameter an.

Tabelle 18 – Modellparameter Versuch 3

Parameter	FFN	LSTM	GRU
Verlustfunktion	categorical crossentropy	categorical crossentropy	categrical crossentropy
Optimierer	SGD	SGD	SGD
Lernrate	0.1	0.1	0.1
Nesterov Momentum	0.9	0.9	0.9
Lernverfahren	Mini Batch Gradienten Abstieg	Mini Batch Gradienten Absteig	Mini Batch Gradienten Abstieg
Batches Pro Epoch	10	10	10
Beispiele pro Batch	262144	262144	262144
Epochs vor dem frühzeitigen Stopp	50	50	50
Dropout	-	0.1	0.1
Rekurrenter Dropout	-	0.5	0.5
Sequenzlänge	1	2	2
Gesamtzahl an Modellparametern	11936	36252	27369

Versuchsauswertung

Die Konvergenz der Verlustfunktionen verhält sich ähnlich zum 2. Versuch (Abbildung 29). Jedoch nähern sich die jeweiligen Funktionen, insbesondere beim LSTM Netz, dem entsprechenden Konvergenzwert bereits nach einer geringeren Anzahl an Epochs an. Gleichermaßen ändern sich die Funktionen der Akkuratheit der jeweiligen Datensätze. Deren Verlauf ändert sich in diesem Versuch beim FFN und GRU Netz nicht sonderlich. Lediglich die Akkuratheit des LSTM Netzes erreicht ein verändertes Ergebnis, indem der Konvergenzwert bereits nach einer geringeren Anzahl an Epochs erreicht wird.

Die Vorhersagen werden mit einer ähnlichen Präzision wie in Versuch 2 getätigt. Hierbei sinkt die Performance des LSTM- um ca 0.5% und die des GRU Netzes um ca 1.5% ab (Tabelle 19). Die Konfusionsmatrix des FFN (Kapitel 7.3) hat einen ähnlichen Verlauf wie im vorangegangenen Versuch und das Netz trifft für alle Klassen die Vorhersagen mit einer Genauigkeit von annähernd 100%. Im Gegensatz dazu stehen die Konfusionsmatritzen der anderen beiden Netze. Während die Netze im vorherigen Versuch so gut wie keine Fehler bei der Klassifikation vorhergesagt haben, weisen die Konfusionsmatritzen des dritten Versuchs eine geringe Streuung an Vorhersagefehlern auf. Bei so gut wie allen Klassen sind Beispiele in die beiden höheren, bzw. niedrigeren Klassen falsch eingeordnet. Bestätigt wird dies durch den Klassifikationsreport, bei welchem das LSTM Netz lediglich 5 und das GRU Netz nur 4 Klassen zu 100% richtig prognostiziert. Obwohl sich das

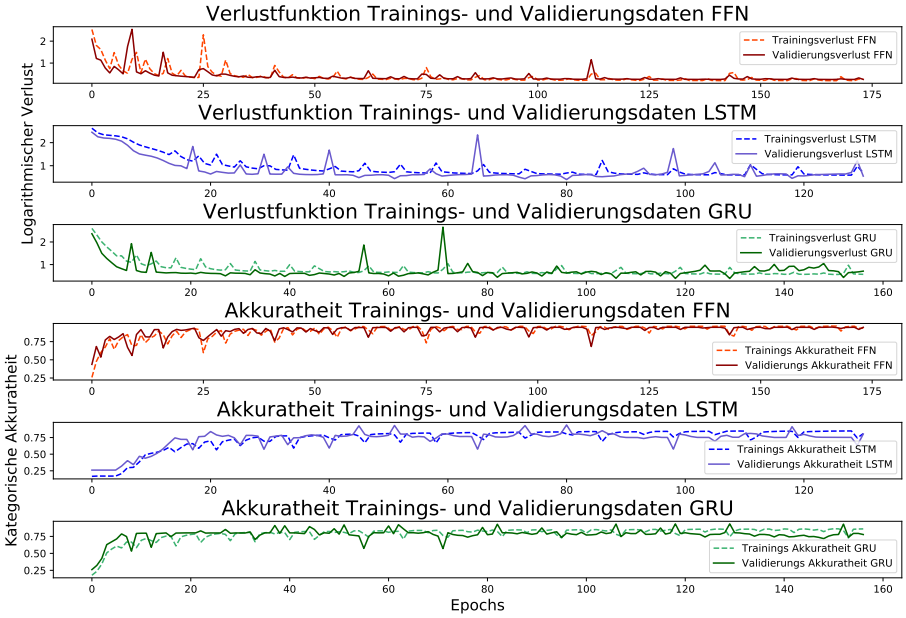

Abbildung 29 – Trainingsdaten Versuch 3

Ergebnis des Klassifikationsreports der Klassen 9 und 10 deutlich verbessert, verschiebt sich die in Versuch 2 aufgetretene Problemstellung, bei welcher bestimmte Klassen nicht prognostiziert werden, auf die Klassen 12 und 13. Die Zuordnung dieser erfolgt in starken Maße der Klasse 14, wodurch die *Precision* der Klasse niedrig ist, verglichen mit dem guten Ergebnis des *Recall*.

Die Vorhersagedauer des FFN steigt marginal an, während sich die Vorhersagedauer im LSTM und GRU Netz um 1.5-2 Sekunden erhöht. Der starke Anstieg

Tabelle 19 – Auswertung Versuch 3

Parameter	FFN	LSTM	GRU
Akkuratheit der Vorhersage	0.9999	0.9527	0.9435
Dauer der Vorhersage [sek]	2.41	6.52	5.74
Epochs bis zum frühzeitigen Stopp	175	131	157
Durchschnittliche Berechnungsdauer pro Epoch [sek]	17	54	52

der Vorhersagedauer der beiden rekurrenten Netze lässt sich auf die erhöhte Sequenzlänge zurückführen. Dadurch steigt der Rechenaufwand, um den internen Zellzustand der LSTM- und GRU Zellen zu ermitteln. Dies wirkt sich ebenfalls auf die durchschnittliche Berechnungsdauer pro Epoch aus, welche sowohl beim LSTM, als auch beim GRU Netz um fast 20 Sekunden ansteigt.

Versuchsergebnis

Das FFN liefert auch in diesem Versuch die beste Performance und hat in allen drei Versuchen ein fast identisches Ergebnis in den Testdaten erreicht. Die Performance des LSTM und GRU Netz sinkt geringfügig und die Berechnungsdauer im Training und während den Vorhersagen steigt etwas an. Beide Beobachtungen lassen sich auf die erhöhte Modellkomplexität zurückführen. Hierbei bewirken die durch die zusätzlichen Klassen bedingten Varianzen, die Notwendigkeit einer geringeren Modellkomplexität und die zusätzlichen Modellparameter, insbesondere die erhöhte Sequenzlänge, führen zu einem höheren Rechenaufwand. Auch die Streuung der Vorhersagefehler lässt sich hierdurch begründen. Gemäß Abbildung 12 empfiehlt eine größere Anzahl an Varianzen eine reduzierte Modellkomplexität. Da in diesem Versuch jedoch sowohl die Modellkomplexität, als auch die Varianzen gestiegen sind, entstehen zusätzliche Fehler in den Vorhersagen.

Während im 2. Versuch beide rekurrenten Netze alle Beispiele der Klasse 9 falsch zugeordnet haben, liegt die Akkuratheit der Zuordnung der Daten dieser Klasse von beiden Netzen bei über 97% (LSTM) bzw. 90% (GRU). Daraus lässt sich schließen, dass insbesondere die Einteilung der Beobachtungen von mehr als 10 Fahrrädern in die Klasse 10 dazu geführt hat, dass die Netze die Klasse 9 nicht prognostizieren. Darüber hinaus bestätigt die schlechte Performance der beiden Netze in den Klassen $11, 12$ und 13 dies. Weiterhin kann der Sachverhalt durch die Aufteilung der Daten begründet sein. Hierbei sind die beiden Klassen 11 und 12 wiederum die Klassen mit der geringsten Anzahl an Gesamtbeispielen (Kapitel 7.3). Durch weitere Beispiele dieser Klassen könnte eine bessere Performance der Netze erreicht werden. Zusätzlich nimmt die Verteilung der Beispieldaten (Abbildung 30) in den jeweiligen Klassen und den unterschiedlichen Datensätzen einen Einfluss auf die Netzperformance. Durch eine sehr geringe Anzahl an Beispielen im Validierungsdatensatz, kombiniert mit einer vergleichsweise hohen Anzahl an Beispielen in den Testdaten, führt dies dazu, dass beim Training der Netze die

Validierungsdaten einen Einfluss nehmen, durch welchen die Verfügbarkeit dieser Klassen sehr gering ist. Zudem beeinflusst die hohe Anzahl an Beispielen in den Testdaten die Performance, da die Nichtberücksichtigung dieser in der Vorhersage, eine hohe Anzahl an Fehlern verursacht.

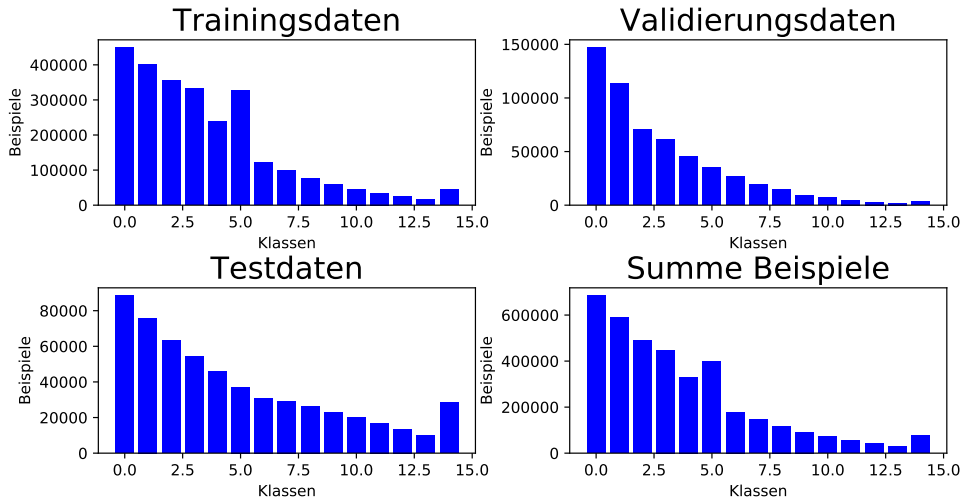

Abbildung 30 – Datenverteilung Versuch 3

5.4 Vorhersagen über mehrere Zeitschritte

Der letzte Versuch befasst sich mit der Prognose über mehrere Zeitschritte hinweg. Dafür erfolgt eine Verschiebung der Zieldaten um 3 weitere Zeitschritte in die Zukunft, sodass jedem Datenpunkt ein Zielwert zugeordnet ist, der 20 Minuten in der Zukunft liegt. Da der Datensatz hierbei eine stark Verzerrung erfährt, werden die Varianzen in diesem reduziert. Dadurch eignet sich ein komplexeres Modell (Abbildung 12) für die Prognosen. Die Varianzen nehmen durch das Reduzieren der Anzahl an Klassen ab. Hierfür werden die Fahrraddaten in lediglich 7 Klassen eingeteilt und entsprechen somit einer Klasse im Intervall $[0, 6]$. Alle Beobachtungen von mehr als 6 Fahrrädern an einer Station werden somit der Klasse 6 zugeordnet.

Modellarchitektur

Die Erhöhung der Anzahl an Neuronen in der versteckten Schicht steigert die Modellkomplexität der einzelnen Netze. Eine Ausnahme bildet das GRU Netz welches bei den vorherigen Versuchen die niedrigste Performance aufgewiesen hat. Die Ursache kann u. a. durch eine hohe Modellkomplexität begründet sein, weshalb die Modellstruktur des GRU Netzes lediglich hinsichtlich der Outputneuronen angepasst wird. In allen Netzen reduziert sich die Anzahl auf 7 Neuronen. Das FFN verwendet 181 und das LSTM 59 versteckte Neuronen in den versteckten Schichten.

Tabelle 20 – Modellarchitektur Versuch 4

	Modell	**FFN**	**LSTM**	**GRU**
Input Schicht	Art der Schicht	Flatten Layer	LSTM Layer	GRU Layer
	Neuronenzahl	23	23	23
	Aktivierungsfunktion	-	-	-
1.versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	181	59	47
	Aktivierungsfunktion	ReLU	tanh	tanh
2. versteckte Schicht	Art der Schicht	Dense Layer	LSTM Layer	GRU Layer
	Neuronenzahl	181	59	47
	Aktivierungsfunktion	ReLU	ReLU	ReLU
Output Schicht	Art der Schicht	Dense layer	Dense Layer	Dense Layer
	Neuronenzahl	7	7	7
	Aktivierungsfunktion	Softmax	Softmax	Softmax

Modellparameter

Beide rekurrenten Netze erhalten in diesem Versuch die ADAM Optimierungsfunktion. Zusätzlich reduziert sich die Sequenzlänge bei den beiden Netzen und entspricht dem Wert 1. Dadurch ergibt sich eine Reduktion der Modellparameter im GRU Netz, eine Erhöhung im LSTM Netz um mehr als 15000 Parameter und eine Verdreifachung im FFN (Tabelle 21).

Tabelle 21 – Modellparameter Versuch 4

Parameter	FFN	LSTM	GRU
Verlustfunktion	categorical crossentropy	categorical crossentropy	categrical crossentropy
Optimierer	SGD	ADAM	ADAM
Lernrate	0.1	0.1	0.1
Nesterov Momentum	0.9	-	-
Lernverfahren	Mini Batch Gradienten Abstieg	Mini Batch Gradienten Absteig	Mini Batch Gradienten Abstieg
Batches Pro Epoch	10	10	10
Beispiele pro Batch	262144	262144	262144
Epochs vor dem frühzeitigen Stopp	50	50	50
Dropout	-	0.1	0.1
Rekurrenter Dropout	-	0.5	0.5
Sequenzlänge	1	1	1
Trainierbare Modellparametern	38560	52416	26985

Versuchsauswertung

Anders als bei den bisherigen Versuchen ist der Verlauf der Verlustfunktionen (Abbildung 31). Lediglich beim FFN weisen die diese eine durchgehende Konvergenz auf. Jedoch konvergiert die Trainingsverlustfunktion gegen einen niedrigeren Wert als die Funktion des Validierungsdatensatzes. Der Verlauf der jeweiligen Funktionen ähnelt dem Verlauf aus dem ersten Versuch. Hierbei konvergieren beide Verlustfunktionen in den ersten 30 Epochs gegen den gleichen Wert. Anschließend ändert sich deren Verlauf. Während die jeweiligen Trainingsverlustfunktionen gegen einen Wert im Bereich um 0.5 konvergieren, beginnen die Validierungsverlustfunktionen wieder zu steigen. Die Akkuratheit der Vorhersagen sinkt in diesem Versuch deutlich ab. Dem stärksten Verlust unterliegt dabei das FFN mit fast 7.5%. Das GRU Netz büßt ca 2% Genauigkeit ein und das LSTM Netz hat im vorherigen Versuch die Testdaten mit einer fast 5% höheren Akkuratheit vorhergesagt (Tabelle 22). Die Konfusionsmatritzen (Kapitel 7.4) weisen hierbei nach, dass die niedrige Akkuratheit der Vorhersagen insbesondere durch eine hohe Streuung falscher Vorhersagen begründet ist. Während bei allen Netzen die Vorhersagefehler in den vorangegangenen Versuchen durch einzelne Klassen begründet sind, zeigt der Klassifikationsreport dieses Versuchs, dass der *F1-Score* bei allen Klassen in einem Bereich von $85\% - 95\%$ liegt. Lediglich die 6. Klasse weist mit 98% ein Ergebnis, ähnlich den vorherigen Versuchen, auf. Während das FFN in diesem Versuch eine längere Zeit als bisher für die Vorher-

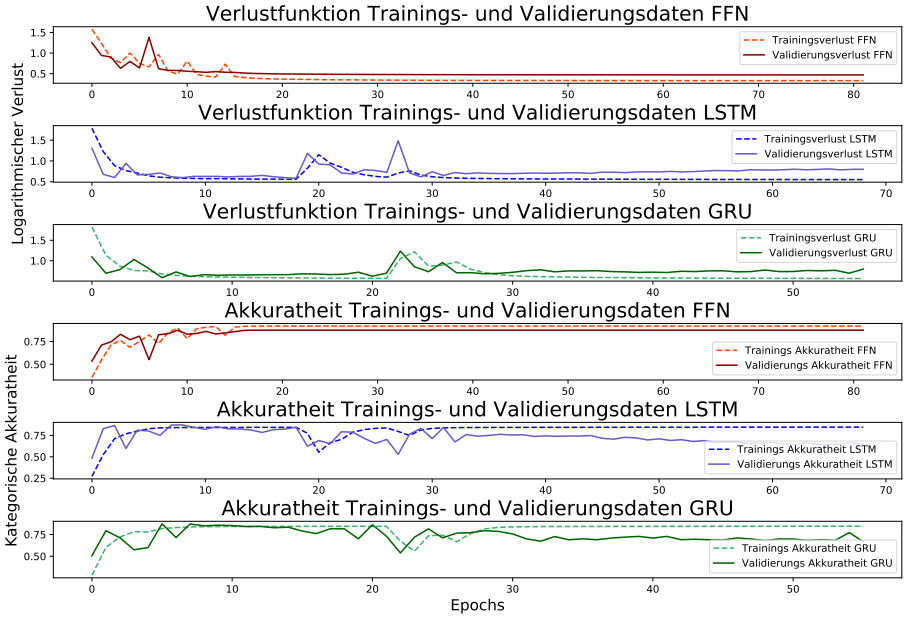

Abbildung 31 – Trainingsdaten Versuch 4

sagen benötigt, weisen die beiden rekurrenten Netze eine vergleichsweise geringe Vorhersagezeit auf. Gleichermaßen verhält sich die Berechnungsdauer pro Epoch. Auch hier benötigt das FFN mehr Zeit, während das LSTM- und GRU Netz die Rechenzeit reduzieren. Die Ursache hierfür kann in der erhöhten Anzahl an Paramtern im FFN ausgemacht werden. Zuletzt ist noch anzumerken, dass im Gegensatz zu den vorherigen Versuchen die für das Training benötigten Epochs in einem deutlich geringeren Bereich liegen.

Tabelle 22 – Auswertung Versuch 4

Parameter	**FFN**	**LSTM**	**GRU**
Akkuratheit der Vorhersage	0.9251	0.9091	0.9240
Dauer der Vorhersage [sek]	3.00	4.58	4.40
Epochs bis zum frühzeitigen Stopp	82	69	56
Durchschnittliche Berechnungsdauer pro Epoch [sek]	19	36	30

Versuchsergebnis

Während alle Netze eine schlechtere Performance aufweisen, ist die des GRU Netzes vergleichsweise gut, da es zum ersten Mal in einem gleichen Wertebereich wie das FFN Netz liegt. Für die schlechtere Performance der Netze lässt sich insbesondere die gesteigerte Komplexität der Lernaufgabe ausmachen. Diese ist durch eine stark erhöhte Anzahl an Verzerrungen begründet, welche zum einen mit der Einteilung der Fahrraddaten in 7 Klassen einhergeht und zum anderen mit der Verschiebung der Zielvariable um zusätzliche 3 Zeitschritte in die Zukunft, verursacht ist.

Da sich die schlechtere Performance durch alle Klassen zieht, ist fraglich ob eine bessere Modelldefinition das Ergebnis der Vorhersagen steigert. Gestützt wird die Vermutung durch den Sachverhalt, dass alle Netze ein deutlich schlechteres Konvergenzverhalten aufweisen. Da das Konvergenzverhalten der rekurrenten Netze ähnlich zum ersten Versuch verläuft, in welchem ebenfalls die ADAM Verlustfunktion verwendet wurde, ist zusätzlich anzunehmen, dass diese Optimierungsfunktion für die betrachteten Lernaufgaben weniger geeignet ist.

Zuletzt können wiederum die Verteilungen der Beispiele pro Klasse die Modellperformance beeinflussen (Tabelle 32). Hierbei repräsentieren die Trainingsdaten die Verteilung des gesamten Datensatzes gut, jedoch weichen die Validierungsdaten leicht davon ab. Zusätzlich weisen die Testdaten eine stark abweichende Verteilung auf.

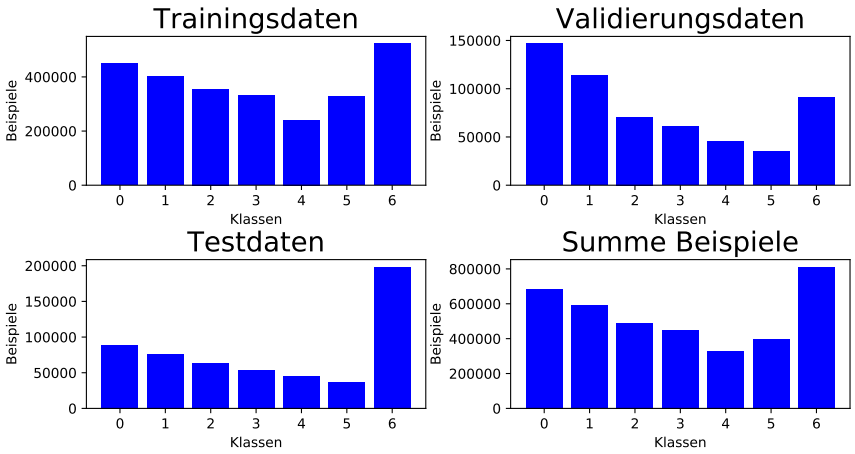

Abbildung 32 – Datenverteilung Versuch 4

6 Fazit

Die FFN Netze haben in allen Versuchen die beste Performance aufgewiesen. Neben einem besseren Konvergenzverhalten, sowie einem schnelleren Lernprozess, hat sich dies insbesondere durch eine sehr hohe Akkuratheit und einer damit verbundenen marginalen Anzahl an Vorhersagefehlern in den Testdaten verdeutlicht. Beide rekurrenten Netzstrukturen haben ebenfalls gute Ergebnisse vorhergesagt, jedoch ist die Performance über alle Versuche schlechtere. Die Ursache dafür liegt vornehmlich an der komplexeren Modellstruktur. Trotz einer geringeren Anzahl an versteckten Neuronen besitzen die rekurrenten Netze bis zu viermal mehr Modellparameter. Diese entstehen durch die zusätzlichen rückwärtsgerichteten Verbindungen der Neuronen. Aufgrund der erhöhten Modellkomplexität ist es deutlich schwieriger eine optimale Modellstruktur zu definieren. Der direkte Vergleich der LSTM und GRU Netzstrukturen verdeutlicht über alle Versuche hinweg den Kompromiss zwischen einer besseren Vorhersageperformance und einer kürzeren Berechnungsdauer. Hierbei haben die LSTM Netze jeweils eine höhere Akkuratheit und damit eine niedrigere Anzahl an Vorhersagefehlern prognostiziert, dabei jedoch eine längere Berechnungsdauer benötigt. Gegensätzlich dazu schneiden die GRU Netze ab. Bei diesen sind eine höhere Anzahl an Vorhersagefehlern prognostiziert worden. Zusätzlich ist die Streuung der Vorhersagefehler über eine größere Anzahl an Klassen vorhanden, jedoch ist die Dauer des Trainings und der Vorhersagen bei dieser Netzstruktur kürzer.

Weiterhin sind die Definition des Modells und die Vorbereitung des Datensatzes entscheidende Faktoren für den Erfolg eines neuronalen Netzes. Bei ersterem liegt die Herausforderung, neben der Wahl von Anzahl und Art der Schicht, sowie der Neuronenzahl innerhalb der jeweiligen Schichten, insbesondere in der Bestimmung des Optimierers. Von den beiden genutzten Optimierern hat der *SGD* ein deutlich besseres Konvergenzverhalten der Verlustfunktionen erzeugt. Zurückführen lässt sich dies auf das verwendete Nesterov Momentum, welches die Anpassungen der Netzparameter beeinflusst, indem die Updates stark in eine Richtung beschleunigt werden. Beim Vergleich der Optimierer ist jedoch zu

berücksichtigen, dass der Trainingsprozess der jeweiligen Netze nach einer geringen Anzahl an Epochs beendet wird. Bei einer Lernaufgabe, für welche eine höhere Anzahl an Epochs erforderlich ist, kann dieses Ergebnis abweichen und der *ADAM* Optimierer ein besseres Ergebnis erzeugen als der *SGD*.

Bei der Datenvorbereitung spielt zunächst die Transformation eine wichtige Rolle. Verfahren wie die *one hot encoding* Methode bieten zwar eine einfache Möglichkeit der Transformation, jedoch wird hierdurch die Anzahl an Merkmalen im Datensatz stark erhöht. Hier bieten Vorgehensweisen wie bspw. die Verwendung von *hash Funktionen* eine gute Alternative. Auch bei der Transformation der numerischen Inputwerte sind einige Aspekte zu beachten. Zunächst müssen die Aktivierungsfunktionen mit Werten gespeist werden, die deren Aktivierungsbereich entsprechen. Herausforderungen bieten hier insbesondere die für die rekurrenten Netze verwendeten *tanh* und *sigmoid* Aktivierungsfunktionen. Beide sind darauf angewiesen, Eingabewerte in ihren mittleren Wertebereichen zu erhalten. Die *tanh* Aktivierungsfunktion ist auf den Wert 0 zentriert und benötigt somit Eingabewerte, die möglichst nah an diesem Wert liegen. Anders ist dies bei der *sigmoid* Aktivierungsfunktion, welche auf den Wert 0.5 zentriert ist. Da der Output der Schichten in den rekurrenten Netzen durch eine tanh Aktivierungsfunktion erzeugt wird, bietet sich das Verfahren von LeCun et al. (2012) an.

Zusätzlich zur richtigen Transformation ist die Zusammensetzung des Datensatzes ein entscheidender Faktor für einen erfolgreichen Lernvorgang. Insbesondere die in Kapitel 5 durchgeführten Versuche geben hier einen genaueren Aufschluss. Der Vergleich des Datensatzes zwischen den beiden ersten Versuchen zeigt, trotz der Reduktion der Anzahl an Inputmerkmalen ändert sich die Performance der Netze nur geringfügig. Dabei verringern sich auf der einen Seite Verzerrungen da Merkmale, die keinen signifikanten Einfluss auf die betrachtete Problemstellung besitzen, entfernt werden und die Vermeidung von Merkmalen, die überhaupt keinen Einfluss auf die Problemstellung besitzen, reduziert weitere Verzerrungen. Auf der anderen Seite ist jedoch zu berücksichtigen, dass das Entfernen von zu vielen Merkmalen wiederum zu der Problemstellung führt, dass der verwendete Input nicht für eine genaue Beschreibung der Lernaufgabe ausreicht. Die Konsequenz daraus sind insbesondere Unteranpassungen und somit eine schlechte Generalisierbarkeit.

Weiterhin zeigt der Vergleich zwischen dem zweiten und dritten Versuch den ge-

genläufigen Einfluss von Verzerrungen und Varianzen auf den Datensatz. Hierbei bewirkt die Reduktion von Varianzen Verzerrungen, und umgekehrt. In den Versuchen wird dies insbesondere durch den *F1-Score* und die Konfusionsmatritzen der Klassen 9 und 10 verdeutlicht. Im zweiten Versuch, bei welchem die Varianz der Klassen geringer ist, wird die Klasse 9 nicht erkannt. Im folgenden Versuch sind die Anzahl an Klassen und somit die Varianzen höher, wodurch die Netze lernen, die Klassen 9 und 10 besser auseinander zu halten. Im letzten Versuch wird die Varianz an unterschiedlichen Klassen minimiert, sodass eine deutlich ausgeglichenere Verteilung zwischen der Anzahl an Trainingsbeispielen der einzelnen Klassen vorhanden ist. Dadurch entstehen wiederum starke Verzerrungen in den Daten, da Beobachtungen mit bspw. 11 Fahrrädern an einer Station der Klasse 6 zugeordnet sind. Zusätzliche Verzerrungen entstehen durch die Prognose weiterer Zeitschritte. Während in den ersten drei Versuchen lediglich ein Zeitschritt aus der Vergangenheit als Input genutzt wird, liegt dieser beim letzten Versuch bereits 20 Minuten in der Vergangenheit. Da insbesondere in den Stoßzeiten des Verleihsystems die Anzahl an Fahrrädern zwischen einzelnen Beobachtungen stark variiert, stellt der Input den tatsächlichen Zustand an der Station nur schlecht dar, wodurch das Lernproblem unterangepasst wird und die Performance aller Netze in den Testdatensatz stark verschlechtert ist.

Zuletzt haben die Versuche einen weiteren Aufschluss für das schlechtere Abschneiden der rekurrenten Netze gegeben. Da deren Performance mit zunehmender Sequenzlänge abnimmt, ist davon auszugehen, dass das Lernproblem stark von Kurzzeit Abhängigkeiten geprägt ist. Gestützt wird diese Vermutung durch das sehr gute Abschneiden der vorwärtsgerichteten Netzstruktur ohne Sequenzlänge. Da sich die beiden rekurrenten Netze jedoch insbesondere zur Beschreibung von Langzeit Abhängigkeiten eignen, ist die Performance dieser bei Problemstellungen, die stark auf Kurzzeit Abhängigkeiten beruhen, schlechter. Somit muss die Eignung der verwendeten Netzstrukturen für die vorliegende Problemstellung als zusätzlicher Faktor für das schlechtere Abschneiden der rekurrenten Netzstrukturen in den Versuchen betrachtet werden.

7 Anhang

Tabelle 23 – Übersicht Stationen

Nummer	Stationsname	Fahrradgestelle	Stationsart
1	Stadtmitte / Fruchthalle	10.0	Feste Station
2	Stiftsplatz	10.0	Feste Station
3	Richard-Wagner-Straße / Königstraße	10.0	Feste Station
4	Friedenstr. / Hilgardring	10.0	Feste Station
5	Hauptbahnhof	14.0	Feste Station
6	Hauptbahnhof / Zollamtstraße	14.0	Feste Station
7	Hilde-Mattauch-Platz	10.0	Feste Station
8	Rummelstraße	6.0	Feste Station
9	Pfaffplatz	10.0	Feste Station
10	Stadtpark	10.0	Feste Station
11	Westbahnhof / Gartenschau	10.0	Feste Station
12	Trippstadter Straße / Institute	10.0	Feste Station
13	Demando / PRE-Park	6.0	Feste Station
14	Davenportplatz	10.0	Feste Station
15	Volkspark	10.0	Feste Station
16	SWK Verkehrs AG (Betriebshof)		Freie Station
17	HS Kaiserslautern		Freie Station
18	SWK Kaiserslautern Versorgung		Freie Station
19	TUK Mensa		Freie Station
20	TUK - Innenhof Gebäude 47		Freie Station
21	TUK - Pfaffenbergstraße		Freie Station
22	Goetheschule	10.0	Feste Station

7.1 Tabellen Versuch 1

Tabelle 24 – Konfusionsmatritzen Versuch 1

	Klasse	0	1	2	3	4	5	6	7	8	9	10
	0	88410	1	1	1	2	1	0	0	0	0	0
	1	2	75811	0	0	0	1	0	0	0	0	1
	2	1	1	63292	0	1	0	0	0	0	0	0
	3	0	0	0	54370	0	0	0	0	0	0	0
FFN	**4**	0	0	1	0	45774	0	0	0	0	1	0
	5	0	0	0	0	0	37083	0	0	0	0	0
	6	0	0	0	0	0	0	30937	0	0	0	0
	7	0	0	0	0	0	0	0	29283	0	0	0
	8	0	0	1	0	0	0	0	0	26461	0	0
	9	0	0	1	0	0	0	0	0	0	22573	192
	10	0	0	0	1	1	0	0	0	1	0	89215
	0	88410	1	1	1	1	2	0	0	0	0	0
	1	2	75811	0	0	1	0	0	0	0	0	1
	2	1	9	63282	1	0	0	0	0	0	0	0
	3	0	0	41	54329	0	0	0	0	0	0	0
	4	0	0	1	166	45591	17	0	1	0	0	0
LSTM	**5**	0	0	0	0	143	36940	0	0	0	0	0
	6	0	0	0	0	0	176	30757	0	0	0	4
	7	0	0	0	0	0	0	1044	28236	0	0	3
	8	3	0	1	0	0	0	6	941	25474	33	4
	9	6	0	1	0	0	0	0	13	1465	21275	6
	10	8	0	0	0	1	1	0	13	55	1718	87422
	0	88267	145	0	0	0	0	0	0	0	0	4
	1	1374	74270	164	2	4	1	0	0	0	0	0
	2	40	1576	61492	180	5	0	0	0	0	0	0
	3	24	50	1608	52515	160	11	2	0	0	0	0
	4	13	1	35	1261	44312	133	16	5	0	0	0
GRU	**5**	7	0	15	22	1216	35680	118	14	11	0	0
	6	9	3	4	12	28	3475	27176	168	42	4	16
	7	0	0	2	3	1	111	1747	26527	788	2	102
	8	0	0	1	2	1	24	117	1536	23264	117	1400
	9	0	0	0	1	0	15	20	96	794	318	21522
	10	0	0	0	0	0	25	25	43	91	1	89033

Tabelle 25 – Klassifikationsreport Versuch 1

	Klasse	Precision	Recall	F1-Score	Anzahl an Testbeispielen
FFN	**0**	1	1	1	88416
	1	1	1	1	75815
	2	1	1	1	63295
	3	1	1	1	54370
	4	1	1	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	1	1	1	29283
	8	1	1	1	26462
	9	1	0.99	1	22766
	10	1	1	1	89218
LSTM	**0**	1	1	1	88146
	1	1	1	1	75815
	2	1	1	1	63293
	3	1	1	1	54370
	4	1	1	1	45776
	5	0.99	1	1	37083
	6	0.97	0.99	0.98	30937
	7	0.97	0.96	0.97	29283
	8	0.94	0.96	0.95	26462
	9	0.92	0.93	0.93	22766
	10	1	0.98	0.99	89218
GRU	**0**	0.98	1	0.99	88416
	1	0.98	0.98	0.98	75815
	2	0.97	0.97	0.97	63293
	3	0.97	0.97	0.97	54370
	4	0.97	0.97	0.97	45776
	5	0.90	0.96	0.93	37083
	6	0.93	0.88	0.90	30937
	7	0.93	0.91	0.92	29283
	8	0.93	0.88	0.90	26462
	9	0.72	0.01	0.03	22766
	10	0.79	1	0.88	89218

Tabelle 26 – Aufteilung Daten Versuch 1 und 2

Klasse	Trainingsdaten	Validierungsdaten	Testdaten	Summe Beispiele
0.0	448821	146863	88416	684100
1.0	401177	113459	75815	590451
2.0	356061	70826	63295	490182
3.0	332416	61141	54370	447927
4.0	240229	45453	45777	331459
5.0	327199	35054	37083	399336
6.0	120898	26866	30937	178701
7.0	98116	19327	29283	146726
8.0	77432	14813	26462	118707
9.0	59858	9116	22766	91740
10.0	167081	20504	89218	276803

7.2 Tabellen Versuch 2

Tabelle 27 – Konfusionsmatritzen Versuch 2

	Klasse	0	1	2	3	4	5	6	7	8	9	10
	0	88410	1	1	1	2	1	0	0	0	0	0
	1	2	75811	0	0	0	1	0	0	0	0	1
	2	1	1	63292	0	1	0	0	0	0	0	0
	3	0	0	0	54370	0	0	0	0	0	0	0
	4	0	0	1	0	45774	0	0	0	0	1	0
FFN	**5**	0	0	0	0	0	37083	0	0	0	0	0
	6	0	0	0	0	0	0	30937	0	0	0	0
	7	0	0	0	0	0	0	0	29283	0	0	0
	8	0	0	1	0	0	0	0	0	26461	0	0
	9	0	0	1	0	0	0	0	0	0	22765	0
	10	0	0	0	1	1	0	0	0	1	0	89215
	0	88410	1	1	1	2	1	0	0	0	0	0
	1	2	75811	0	0	0	1	0	0	0	0	1
	2	1	1	63292	0	1	0	0	0	0	0	0
	3	0	0	0	54370	0	0	0	0	0	0	0
	4	0	0	1	0	45774	0	0	0	0	0	1
LSTM	**5**	0	0	0	0	0	37083	0	0	0	0	0
	6	0	0	0	0	0	0	30937	0	0	0	0
	7	0	0	0	0	0	0	0	29283	0	0	0
	8	0	0	1	0	0	0	0	120	26341	0	0
	9	0	0	1	0	0	0	0	0	0	0	22765
	10	0	0	0	1	1	0	0	0	1	0	89215
	0	88410	1	1	1	2	1	0	0	0	0	0
	1	2	75811	0	0	0	1	0	0	0	0	1
	2	1	1	63292	0	1	0	0	0	0	0	0
	3	0	0	0	54370	0	0	0	0	0	0	0
	4	0	0	1	0	45774	0	0	0	0	0	1
GRU	**5**	0	0	0	0	0	37083	0	0	0	0	0
	6	0	0	0	0	0	0	30937	0	0	0	0
	7	0	0	0	0	0	0	0	29283	0	0	0
	8	0	0	1	0	0	0	0	0	26461	0	0
	9	0	0	1	0	0	0	0	0	0	0	22765
	10	0	0	0	1	1	0	0	0	1	0	89215

Tabelle 28 – Klassifikationsreport Versuch 2

	Klasse	Precision	Recall	F1-Score	Anzahl an Testbeispielen
FFN	0	1	1	1	88416
	1	1	1	1	75815
	2	1	1	1	63295
	3	1	1	1	54370
	4	1	1	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	1	1	1	29283
	8	1	1	1	26462
	9	1	1	1	22766
	10	1	1	1	89218
LSTM	0	1	1	1	88146
	1	1	1	1	75815
	2	1	1	1	63295
	3	1	1	1	54370
	4	1	1	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	1	1	1	29283
	8	1	1	1	26462
	9	0	0	0	22766
	10	0.80	1	0.89	89218
GRU	0	1	1	1	88416
	1	1	1	1	75815
	2	1	1	1	63295
	3	1	1	1	54370
	4	1	1	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	1	1	1	29283
	8	1	1	1	26462
	9	0	0	0	22766
	10	0.80	1	0.89	89218

7.3 Tabellen Versuch 3

Tabelle 29 – Konfusionsmatritzen Versuch 3

	Klassen	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	88410	1	1	1	2	1	0	0	0	0	0	0	0	0	0
	1	2	75811	0	0	0	1	0	0	0	0	0	1	0	0	0
	2	1	1	63292	0	1	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	54370	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	1	0	45774	0	0	0	0	1	0	0	0	0	0
	5	0	0	0	0	0	37083	0	0	0	0	0	0	0	0	0
FFN	6	0	0	0	0	0	0	30937	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	29283	0	0	0	0	0	0	0
	8	0	0	1	0	0	0	0	0	26461	0	0	0	0	0	0
	9	0	0	1	0	0	0	0	0	0	22765	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	20159	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	16909	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	0	13492	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	10064	1
	14	0	0	0	1	1	0	0	0	1	0	0	0	0	0	28590
	0	88391	20	1	2	0	1	0	0	0	1	0	0	0	0	0
	1	4	75798	11	0	1	0	0	0	1	0	0	0	0	0	0
	2	1	124	63144	25	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	238	54119	11	2	0	0	0	0	0	0	0	0	0
	4	0	0	1	176	45560	38	0	0	1	0	0	0	0	0	0
	5	0	0	0	0	38	37033	12	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	168	30741	28	0	0	0	0	0	0	0
LSTM	7	0	0	0	0	0	0	100	29155	28	0	0	0	0	0	0
	8	0	0	0	0	1	0	0	117	26281	63	0	0	0	0	0
	9	0	0	0	0	1	0	0	4	93	22583	85	0	0	0	0
	10	0	0	0	0	0	0	0	0	24	651	19401	82	0	0	1
	11	0	0	0	0	0	0	0	0	3	11	240	16003	0	0	652
	12	0	0	0	0	0	0	0	0	0	8	13	54	0	0	13417
	13	0	0	0	0	0	0	0	0	0	5	1	3	0	0	10056
	14	0	0	0	0	0	2	0	0	0	2	1	4	0	0	28584
	0	88406	5	0	0	2	1	0	0	0	0	0	0	0	0	2
	1	365	75448	0	0	0	1	0	0	0	0	0	0	0	0	1
	2	1	106	63017	170	0	0	0	0	0	0	0	0	0	0	0
	3	0	30	1446	52862	26	5	1	0	0	0	0	0	0	0	0
	4	0	0	1	205	45528	41	0	1	0	0	0	0	0	0	0
	5	0	0	0	0	129	36922	32	0	0	0	0	0	0	0	0
	6	0	8	2	0	0	0	30877	50	0	0	0	0	0	0	0
GRU	7	0	0	0	0	0	0	97	29156	30	0	0	0	0	0	0
	8	0	0	0	0	0	1	0	926	25478	49	6	0	0	0	2
	9	0	0	0	0	0	0	1	22	915	21735	68	23	0	0	2
	10	0	0	0	0	0	0	0	11	16	2539	17530	46	0	0	17
	11	0	0	0	0	0	0	0	1	1	58	2219	14569	6	0	55
	12	0	0	0	0	0	0	0	6	2	13	54	1363	1525	0	10529
	13	0	0	0	0	0	0	0	2	3	1	6	28	24	0	10001
	14	0	0	0	0	0	0	2	0	1	5	2	6	1	0	28576

Tabelle 30 – Klassifikationsreport Versuch 3

	Klasse	Precision	Recall	F1-Score	Anzahl Beispiele
FFN	0	1	1	1	88416
	1	1	1	1	75815
	2	1	1	1	63295
	3	1	1	1	54370
	4	1	1	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	1	1	1	29283
	8	1	1	1	26462
	9	1	1	1	22766
	10	1	1	1	20159
	11	1	1	1	16909
	12	1	1	1	13492
	13	1	1	1	10065
	14	1	1	1	28593
LSTM	0	1	1	1	88416
	1	1	1	1	75815
	2	1	1	1	63294
	3	1	1	1	54370
	4	1	1	1	45776
	5	0.99	1	1	37083
	6	1	0.99	1	30937
	7	0.99	1	1	29283
	8	0.99	0.99	0.99	26462
	9	0.97	0.99	0.98	22766
	10	0.98	0.96	0.97	20159
	11	0.99	0.95	0.97	16909
	12	0	0	0	13492
	13	0	0	0	10065
	14	0.54	1	0.7	28593
GRU	0	1	1	1	88416
	1	1	1	1	75815
	2	0.98	1	0.99	63294
	3	0.99	0.97	0.98	54370
	4	1	0.99	1	45776
	5	1	1	1	37083
	6	1	1	1	30937
	7	0.97	1	0.98	29283
	8	0.96	0.96	0.96	26462
	9	0.89	0.95	0.92	22766
	10	0.88	0.87	0.88	20159
	11	0.91	0.86	0.88	20159
	12	0.98	0.11	0.20	13492
	13	0	0	0	10065
	14	0.58	1	0.73	28593

Tabelle 31 – Aufteilung Daten Versuch 3

Klassen	Trainingsdaten	Validierungsdaten	Testdaten	Summe Beispiele
0	448821	146863	88416	684100
1	401177	113459	75815	590451
2	356061	70826	63295	490182
3	332416	61141	54370	447927
4	240229	45453	45777	331459
5	327199	35054	37083	399336
6	120898	26866	30937	178701
7	98116	19327	29283	146726
8	77432	14813	26462	118707
9	59858	9116	22766	91740
10	46319	7010	20159	73488
11	34569	4780	16909	56258
12	25646	2969	13492	42107
13	15598	2145	10065	27808
14	44949	3600	28593	77142

7.4 Tabellen Versuch 4

Tabelle 32 – Konfusionsmatritzen Versuch 4

	Klassen	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	0	84190	3560	515	100	30	9	8
	1	3536	68129	3478	542	93	21	14
	2	421	3382	55641	3251	488	79	28
FFN	**3**	105	492	3110	47342	2696	461	162
	4	56	101	363	2591	39494	2609	558
	5	18	52	84	374	2496	31296	2761
	6	72	91	103	177	482	2612	195126
	0	84186	3560	515	111	20	8	12
	1	3533	68129	3481	567	70	16	17
	2	421	3382	55697	3337	359	66	28
LSTM	**3**	105	492	3542	47616	2073	378	162
	4	53	101	365	8155	34329	2208	561
	5	18	52	84	897	6128	27141	2761
	6	72	91	103	293	699	2279	195126
	0	84190	3560	515	107	23	9	8
	1	3536	68129	3478	557	79	20	14
	2	421	3382	55641	3310	431	77	28
GRU	**3**	105	492	3110	47554	2491	454	162
	4	56	101	363	3355	38747	2592	558
	5	18	52	84	478	2449	31239	2761
	6	72	91	103	201	464	2606	195126

Tabelle 33 – Klassifikationsreport Versuch 4

	Klasse	Precision	Recall	F1-Score	Anzahl Beispiele
FFN	**0**	0.95	0.95	0.95	88412
	1	0.90	0.90	0.90	75813
	2	0.88	0.88	0.88	63290
	3	0.87	0.87	0.87	54368
	4	0.86	0.86	0.86	45772
	5	0.84	0.84	0.84	37081
	6	0.98	0.98	0.98	198663
LSTM	**0**	0.95	0.95	0.95	88412
	1	0.90	0.90	0.90	75813
	2	0.87	0.88	0.88	63290
	3	0.78	0.88	0.83	54368
	4	0.79	0.75	0.77	45772
	5	0.85	0.73	0.78	37081
	6	0.98	0.98	0.98	198663
GRU	**0**	0.95	0.95	0.95	88412
	1	0.90	0.90	0.90	75813
	2	0.88	0.88	0.88	63290
	3	0.86	0.87	0.87	54368
	4	0.87	0.85	0.86	45772
	5	0.84	0.84	0.84	37081
	6	0.98	0.98	0.98	198663

Tabelle 34 – Aufteilung Daten Versuch 4

Klasse	Trainingsdaten	Validierungsdaten	Testdaten	Summe Beispiele
0	448826	146856	88412	684094
1	401167	113459	75813	590439
2	356062	70830	63290	490182
3	332399	61142	54368	447909
4	240223	45457	45773	331453
5	327180	35054	37081	399315
6	523387	90624	198663	812674

Literaturverzeichnis

Redaktion AI-United. Aktivierungsfunktionen, ihre Arten und Verwendungsmöglichkeiten. URL `https://www.ai-united.de/aktivierungsfunktionen-ihre-arten-und-verwendungsmoeglichkeiten/`.

Apoorva Agrawal. Loss Functions and Optimization Algorithms. Demystified., 2017. URL `https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c`.

Ahmad Azharuddin Azhari Mohd Amiruddin, Haslinda Zabiri, Syed Ali Ammar Taqvi, and Lemma Dendena Tufa. Neural network applications in fault diagnosis and detection: an overview of implementations in engineering-related systems. *Neural Computing and Applications*, pages 1–26, 2018.

Igor P Atamanyuk and Yuriy P Kondratenko. Method of generating realizations of random sequence with the specified characteristics based on nonlinear canonical decomposition. *Journal of Automation and Information Sciences*, 48 (10), 2016.

Mariette Awad and Rahul Khanna. *Efficient learning machines: theories, concepts, and applications for engineers and system designers*. Apress, 2015.

Jörg Baetge and Tobias Henning. Künstliche neuronale netze in der jahresabschlussanalyse. In *Quo vadis Wirtschaftsinformatik?*, pages 265–282. Springer, 2008.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3 (Feb):1137–1155, 2003.

Gavin J Bowden, Graeme C Dandy, and Holger R Maier. Data transformation for neural network models in water resources applications. *Journal of Hydroinformatics*, 5(4):245–258, 2003.

Nils Gerrit Brinkmann. Texterkennung auf handgeschriebenen dokumenten mit long short-term memory networks. 2019.

Florian Broch. Prognose relevanter technischer parameter. In *Integration von ökologischen Lebenswegbewertungen in Fahrzeugentwicklungsprozesse*, pages 123–148. Springer, 2017.

Jason Brownlee. Difference Between a Batch and an Epoch in a Neural Network, 2018. URL `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/`.

Jason Brownlee. Stacked Long Short-Term Memory Networks, 2019a. URL `https://machinelearningmastery.com/stacked-long-short-term-memory-networks/`.

Jason Brownlee. A Gentle Introduction to Cross-Entropy for Machine Learning, 2019b. URL `https://machinelearningmastery.com/cross-entropy-for-machine-learning/`.

Jason Brownlee. A Gentle Introduction to the Rectified Linear Unit (ReLU), 2019c. URL `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/`.

Vitaly Bushaev. Stochastic Gradient Descent with momentum, 2017. URL `https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d`.

Barbara Cannas, Alessandra Fanni, Linda See, and Giuliana Sias. Data preprocessing for river flow forecasting using neural networks: wavelet transforms and data partitioning. *Physics and Chemistry of the Earth, Parts A/B/C*, 31(18): 1164–1171, 2006.

Patricio Cerda, Gaël Varoquaux, and Balázs Kégl. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, 107(8-10):1477–1494, 2018.

Francois Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP–Verlags GmbH & Co. KG, 2018.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Sven Crone. *Neuronale Netze zur Prognose und Disposition im Handel*, volume 60. Springer Verlag, 2010.

Sebastian Dörn. *Programmieren Für Ingenieure und Naturwissenschaftler*. Springer, 2016.

Vicki Foss. Multiclass classification model evaluationAn overview of evaluation metrics for a multiclass machine-learning model, 2018. URL `https://parasite.id/blog/2018-12-13-model-evaluation/`.

Jonathan Frankenberger. *Erkennung von Refactorings durch künstliche neuronale Netze*. PhD thesis, 2007.

Jon Edward Froehlich, Joachim Neumann, and Nuria Oliver. Sensing and predicting the pulse of the city through shared bicycling. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

Dragan Gamberger, Nada Lavrac, and Saso Dzeroski. Noise detection and elimination in data preprocessing: experiments in medical domains. *Applied Artificial Intelligence*, 14(2):205–223, 2000.

Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

Coşkun Hamzaçebi, Diyar Akay, and Fevzi Kutay. Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting. *Expert Systems with Applications*, 36(2):3839–3844, 2009.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Daniel Hsu. Multi-period time series modeling with sparsity via bayesian variational inference. *arXiv preprint arXiv:1707.00666*, 2017.

David Hunter, Hao Yu, Michael S Pukish III, Janusz Kolbusz, and Bogdan M Wilamowski. Selection of proper neural network sizes and architectures—a comparative study. *IEEE Transactions on Industrial Informatics*, 8(2):228–240, 2012.

Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

Adventures in Machine Leraning. Stochastic Gradient Descent – Mini-batch and more, 2018. URL `https://adventuresinmachinelearning.com/stochastic-gradient-descent/`.

Fariborz Jolai and A Ghanbari. Integrating data transformation techniques with Hopfield neural networks for solving travelling salesman problem. *Expert Systems with Applications*, 37(7):5331–5335, 2010.

Andreas Kaltenbrunner, Rodrigo Meza, Jens Grivolla, Joan Codina, and Rafael Banchs. Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system. *Pervasive and Mobile Computing*, 6(4):455–466, 2010.

Ayoosh Kathuria. Intro to optimization in deep learning: Momentum, RMSProp and Adam, 2018. URL `https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/`.

Dokumentation Keras. Docs » Layers » Core Layers, a. URL `https://keras.io/layers/core/`.

Dokumentation Keras. Docs » Callbacks, b. URL `https://keras.io/callbacks/`.

Dokumentation Keras. Docs » Layers » Recurrent Layers, c. URL `https://keras.io/layers/recurrent/`.

Dokumentation Keras. Optimizers, d. URL `https://keras.io/optimizers/`.

Dokumentation Keras. Keras, Dokumentation, 2020.

Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Georg Ruß, and Matthias Steinbrecher. Künstliche neuronale netze. In *Computational Intelligence*, pages 7–11. Springer, 2011.

Gerson Lachtermacher and J David Fuller. Back propagation in time-series forecasting. *Journal of forecasting*, 14(4):381–393, 1995.

R Lacroix, F Salehi, XZ Yang, and KM Wade. Effects of data preprocessing on the performance of artificial neural networks for dairy yield prediction and cow culling classification. *Transactions of the ASAE*, 40(3):839–846, 1997.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Bernd Leiner. *Grundlagen der Zeitreihenanalyse*. Walter de Gruyter GmbH & Co KG, 2018.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Ian London. Encoding cyclical continuous features - 24-hour time, 2016. URL `https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/`.

Hamza Mahmood. The Softmax Function, Simplified, 2018. URL `https://towardsdatascience.com/softmax-function-simplified-714068bf8156`.

Shervin Minaee. 20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics, 2019. URL `https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce`.

Aditya Mishra. Metrics to evaluate your machine learning algorithm, 2018. URL `https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234`.

Tom Mitchell. Machine learning. macgraw-hill companies. *Inc., Boston*, 1997.

Hou Muzhou and Moon Ho Lee. A new constructive method to optimize neural network architecture and generalization. *arXiv preprint arXiv:1302.0324*, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

nextbike GmbH. Vrnnextbike – fahrrad mieten in kaiserslautern. URL `https://www.vrnnextbike.de/de/kaiserslautern/`.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Openweathermap. History Bulk. URL `https://openweathermap.org/history-bulk`.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4):7–9, 2017.

PW Chandana Prasad and Azam Beg. Investigating data preprocessing methods for circuit complexity models. *Expert Systems with Applications*, 36(1):519–526, 2009.

PressestelleStadtKaiserslautern. Sehr gute Ausleihzahlen für das Jahr 2018 – Neuer RadCard Monatstarif, 2019. URL `https://www.wochenblatt-reporter.de/kaiserslautern/c-lokales/sehr-gute-ausleihzahlen-fuer-das-jahr-2018-neuer-radcard-monatstarif_a82603`.

Jean-Francois Puget. Feature Engineering For Deep Learning kernel description, 2017. URL `https://medium.com/inside-machine-learning/feature-engineering-for-deep-learning-2b1fc7605ace`.

Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Raúl Rojas. Künstliche neuronale netze als neues paradigma der informationsverarbeitung. *Neurowissenschaften und Philosophie: eine Einführung. Frankfurt am Main: UTB*, pages 269–297, 2001.

Paul L Rosin and Freddy Fierens. Improving neural network generalisation. In *1995 International Geoscience and Remote Sensing Symposium, IGARSS'95. Quantitative Remote Sensing for Science and Applications*, volume 2, pages 1255–1257. IEEE, 1995.

Binxin Ru, Ahsan S Alvi, Vu Nguyen, Michael A Osborne, and Stephen J Roberts. Bayesian optimisation over multiple continuous and categorical inputs. *arXiv preprint arXiv:1906.08878*, 2019.

Thomas A Runkler. Zeitreihenprognose. In *Data Mining*, pages 81–84. Springer, 2010.

Joseph L Schafer. *Analysis of incomplete multivariate data*. CRC press, 1997.

Andreas Scherer. *Neuronale Netze: Grundlagen und Anwendungen*. Springer Verlag, 2013.

Murali Shanker, Michael Y Hu, and Ming S Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996.

Avinash Sharma. Understanding Activation Functions in Neural Networks, 2017. URL `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0`.

Jonathan Jingsheng Shi. Reducing prediction error by transforming input data for neural networks. *Journal of computing in civil engineering*, 14(2):109–116, 2000.

Ashwin Srinivasan, Stephen Muggleton, and Michael Bain. Distinguishing exceptions from noise in non-monotonic learning. In *Proceedings of the Second Inductive Logic Programming Workshop*, pages 97–107. Citeseer, 1992.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Stefan Strecker. Künstliche neuronale netze–aufbau und funktionsweise. *Arbeitspapiere WI*, (10), 1997.

Zbigniew A Styczynski, Krzysztof Rudion, and André Naumann. Künstliche neuronale netzwerke. In *Einführung in Expertensysteme*, pages 131–201. Springer, 2017.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, pages 1–6, 2015.

M Traeger, A Eberhart, G Geldner, AM Morin, C Putzke, H Wulf, and LHJ Eberhart. Künstliche neuronale netze. *Der Anaesthesist*, 52(11):1055–1061, 2003.

A Wald. Allgemeine bemerkungen zur analyse von zeitreihen. In *Berechnung und Ausschaltung von Saisonschwankungen*, pages 1–10. Springer, 1936.

Lin Wang, Yi Zeng, and Tao Chen. Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. *Expert Systems with Applications*, 42(2):855–863, 2015.

Andreas Welsch, Verena Eitle, and Peter Buxmann. Maschinelles lernen. *HMD Praxis der Wirtschaftsinformatik*, 55(2):366–382, 2018.

Rainer Witschel, Julia und Souren. *Kapazitätswirtschaftliche Analyse der Strukturelemente und Determinanten des Bikesharing.* Ilmenau: Verlag proWiWi eV, 2014.

CL Wu, KW Chau, and C Fan. Prediction of rainfall time series using modular artificial neural networks coupled with data-preprocessing techniques. *Journal of Hydrology*, 389(1-2):146–167, 2010.

Karla Yale. Preparng the right data diet for training neural networks. *IEEE Spectrum*, 34(3):64–66, 1997.

Lean Yu, Shouyang Wang, and Kin Keung Lai. An integrated data preparation scheme for neural network data analysis. *IEEE Transactions on Knowledge and Data Engineering*, 18(2):217–230, 2005.

Btissam Zerhari. Class noise elimination approach for large datasets based on a combination of classifiers. In *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pages 125–130. IEEE, 2016.

G Peter Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European journal of operational research*, 160(2):501–514, 2005.

G Peter Zhang, B Eddy Patuwo, and Michael Y Hu. A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers & Operations Research*, 28(4):381–396, 2001.

Yuchen Zhang, Jason D Lee, and Michael I Jordan. l1-regularized neural networks are improperly learnable in polynomial time. In *International Conference on Machine Learning*, pages 993–1001, 2016.

Zhaosheng Zhang, Diange Yang, Tao Zhang, Qiaochu He, and Xiaomin Lian. A study on the method for cleaning and repairing the probe vehicle data. *IEEE Transactions on Intelligent Transportation Systems*, 14(1):419–427, 2012.