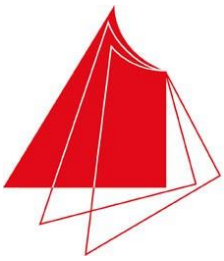


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät Maschinenbau und Mechatronik
STUDIENGANG Mechatronik

BACHELORTHESIS

zur Erlangung des akademischen Grades
Bachelor of Engineering

KI-Methoden zur Vorhersageanalyse in medizintechnischen und ingenieurwissenschaftlichen Anwendungen

von

Daniel Autenrieth
Matrikel-Nr. 59538
Daniel.autenrieth97@gmx.de

Vorgelegt bei

Herrn Prof. Dr.-Ing. Martin Simon
Herrn M.Sc. Tim Schanz

08.Juni 2020 - 05.Oktober 2020

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät für Maschinenbau und Mechatronik

Bachelor-Thesis: **Daniel Autenrieth - 59538**
Mechatronik

Arbeitsplatz: Hochschule Karlsruhe – Technik und Wirtschaft Fakultät für Maschinenbau und Mechatronik

76133 Karlsruhe

Betreuender Dozent: Prof. Dr.-Ing. Martin Simon

Datum der Ausgabe: 08.06.2020 **Abgabetermin:** 05.10.2020

Kurzthema / Subject:

KI-Methoden zur Vorhersageanalyse in medizintechnischen und ingenieurwissenschaftlichen Anwendungen
AI Methods for Predictive Analysis in Medical and Engineering Applications

Hintergrund:

In der Ingenieurwissenschaft gibt es bereits erfolgreiche Ansätze, durch Sensordatenanalyse mit KI-Methoden in Prozessen frühzeitig kritische Trends zu erkennen und rechtzeitig im Sinne einer Schadensvermeidung darauf zu reagieren. Hierbei kommen neuronale Netze zum Einsatz, die durch Sensordaten trainiert werden.

In dieser Arbeit soll untersucht werden, inwieweit die in den Ingenieurswissenschaften eingesetzten Verfahren für medizintechnische Anwendungen, wie beispielsweise Beatmungsmaschinen verwendet werden können.

Aufgabenstellung:

In dieser Bachelorarbeit soll der Einsatz von künstlicher Intelligenz bzw. neuronaler Netze im Bereich der Medizintechnik untersucht werden. In einer ersten Anwendung wird die KI auf menschliche Beatmungsdaten angewendet. Beatmungsdaten beinhalten alle relevanten Daten, die zur Beatmung eines Menschen wichtig sind. Ziel ist es Unregelmäßigkeiten in Beatmungsdaten frühzeitig zu entdecken und gegebenenfalls vorauszusagen. Da die Schwierigkeit in der Beschaffung der Beatmungsdaten liegt, soll zusätzlich ein Generator entwickelt werden. Dieser soll Beatmungsdaten möglichst realistisch generieren.

Um eine KI zu trainieren, müssen verschiedene Architekturen untersucht bzw. entwickelt werden. Die KI soll zunächst mit generierten Daten trainiert und evaluiert werden. In einem weiteren Schritt soll das gleiche Verfahren mit realen Daten, soweit verfügbar, durchgeführt werden.

Im Einzelnen sind die folgenden Punkte zu bearbeiten:

- Literaturrecherche Beatmungsdaten und Auswertung von Beatmungsdaten
- Literaturrecherche Künstliche Intelligenz (Architekturen, Trainingsverfahren, Detektionsverfahren)
- Recherche und Erstellung Stand der Technik
- Erstellen des Datengenerators
- Evaluation der generierten Daten
- Erstellung neuronaler Netze
- Training und Evaluierung der Netze
- Dokumentation und Präsentation der Ergebnisse

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne unzulässige fremde Hilfe selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet zu haben.

Eislingen, den 01.10.2020

Daniel Autenrieth

Kurzfassung

Deutsch

Das Ziel dieser Arbeit ist es, verschiedene Krankheitsbilder mithilfe von neuronalen Netzen zu klassifizieren, um die Diagnostik zu unterstützen. Durch den Einsatz von künstlicher Intelligenz in der Medizintechnik können Vorteile, sowohl für den Pateinten als auch für den Arzt, erzielt werden. In Ländern mit einer schlechten Gesundheitsversorgung, können solche Systeme vielen Menschen den Zugang zu medizinischer Versorgung ermöglichen.

Für die Umsetzung der neuronalen Netze muss eine ausreichende Datenbasis geschaffen werden. Da für diese Arbeit keine Realdaten zur Verfügung stehen, mussten die Daten durch einen Generator erzeugt werden. Die Abhängigkeit der erzeugten Kurven von den betrachteten personenspezifischen Daten ist ein wichtiger Faktor, um naturgetreue Verläufe nachbilden zu können.

Zur Erstellung der neuronalen Netze mussten elementare Parameter, wie die Architektur, das Trainingsverfahren, der Optimierer und weitere, überprüft und ausgewählt werden. Aus diesem Grund ist eine theoretische Betrachtung der Thematik unerlässlich.

Durch die erzeugten Daten und weitreichende Versuchsreihen konnte gezeigt werden, dass es mit einer Feedforward Netzarchitektur möglich ist, eine Klassifizierung von Zeitreihen, die von personenspezifischen Daten abhängen, vorzunehmen.

Englisch

The aim of this thesis is to classify different disease patterns with the help of neural networks to support the diagnostics. Through the use of artificial intelligence, benefits can be achieved for both the patient and the physician, and in countries with poor health care, such systems allow many people to access medical care.

For the implementation of the neural networks a sufficient data base must be created. Since no real data is available for this work the data had to be generated. The dependency of the generated curves on the considered person-specific data is an important factor to be able to reproduce natural behavior.

To create neural networks, elementary parameters such as the architecture, the training procedure, the optimizer and others had to be checked and selected. For this reason, a theoretical view of the topic is essential.

Through the generated data and extensive test series it could be shown that it is possible to classify time-series depending on person-specific data with a feedforward network architecture.

1 Inhaltsverzeichnis

2 Projektplan

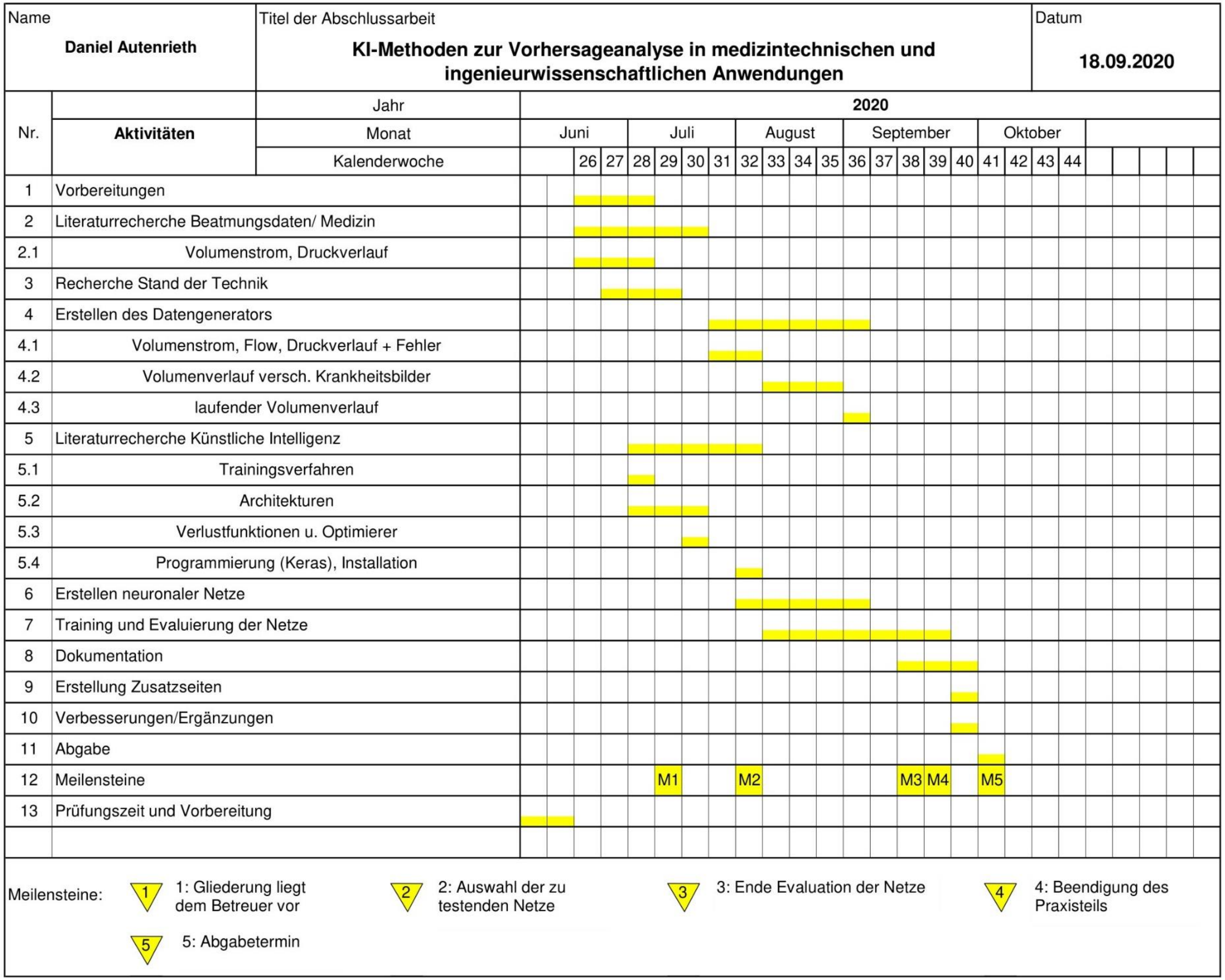


Abbildung 1 Projektplan v7

2.1 Anmerkungen

Vorbereitung:

Beinhaltet: Literatur besorgen, Vorrecherche, Recherche von Datensätzen, Zeitplan, Inhaltsverzeichnis (Strukturierung)

Zusatzseiten:

Kurzfassung, Erklärung, Deckblatt, Erläuterungen Dateien/Ordnerstruktur

Dokumentation:

Beinhaltet die Ergebnisse der praktischen Arbeit. Literaturrecherchen und deren schriftliche Ausarbeitung werden in demselben Zeitraum durchgeführt und formuliert

2.2 Meilensteinplan

Meilenstein 1: Gliederung liegt dem Betreuer vor

Die Arbeit weist eine erste Struktur aufweisen. Die Vorrecherche ist zum Großteil abgeschlossen und ein erster Zeitplan wurde erstellt.

Meilenstein 2: Auswahl der zu testenden Netze

Die Literaturrecherche zu den verschiedenen Netzarchitekturen ist abgeschlossen. Basierend darauf kann eine fundierte Entscheidung getroffen werden. Neben der Architektur sind auch die restlichen Parameter, welche zur Erstellung und Training des Netzes notwendig sind, theoretisch behandelt worden.

Meilenstein 3: Ende Evaluation der Netze

Die Netze wurden erstellt und trainiert. Die Ergebnisse der Netze sind dokumentiert und können für die Ausarbeitung aufbereitet werden.

Meilenstein 4: Beendigung des Praxisteils

Programme zur Darstellung der Ergebnisse und Nutzung der trainierten Netze sind vollendet.

Meilenstein 5: Abgabetermin

Die Ausarbeitung ist vollständig und wurde gedruckt. Alle Programme sind kommentiert und in einer verständlichen Ordnerstruktur für die Abgabe festgehalten.

3 Erläuterung Abkürzungen & Fachbegriffe

Adam	Adaptive Moment Estimation
CEE	Cross Entropy Error
CNN	Convolutional Neural Network
Compliance	Dehnbarkeit (der Atemwege und Lunge)
Flow	Fluss/ Strömung
FNN	Feedforward Neuronales Netz
IVC	Inspiratorische Vitalkapazität
KNN	Künstliches Neuronales Netz
LSTM	Long Short Term Memory
MLP	Multi-Layer-Perzeptron
MSE	Mean Squared Error
PEEP	Positiv End-Expiratory Pressure = Druck in der Lunge nach Exspiration
Resistance	Widerstand (der Atemwege)
SGD	Stochastic Gradient Descent

4 Einleitung

4.1 Einführung in die Thematik

Heutzutage wird bereits eine Vielzahl von unterschiedlichen neuronalen Netzen in den verschiedensten Fachgebieten eingesetzt und entwickelt. Im folgenden Abschnitt werden einige Beispiele aus den Einsatzgebieten gegeben, um dem Leser das breite Verwendungsspektrum von neuronalen Netzen zu verdeutlichen. Aufgrund der Komplexität und Vielzahl beschränkt sich die Darstellung auf einige exemplarische Beispiele.

Neuronale Netze und künstliche Intelligenz im Allgemeinen haben bereits in die meisten technischen Produkte Einzug gehalten. Von der Mikroebene, auf welcher zum Beispiel kleine Sensoren in Smartphones eingesetzt werden, die sich durch neuronale Netze an den Nutzer anpassen oder den Verschleiß von Komponenten kompensieren, bis hin zu großen Gesamtsystemen wie autonomen Fortbewegungsmitteln, in denen sie die Umwelt klassifizieren und wichtige Entscheidungen treffen. Gesichtserkennung und die Unterscheidung von verschiedenen Objekten innerhalb eines Bildes sind mittlerweile zu einem Paradebeispiel geworden, was neuronale Netze alles leisten können. Es existieren frei verfügbare vortrainierte Modelle, wie YOLO (you only look once), die Echtzeit-Objekterkennung für jedermann zugänglich machen. Auch die Wahrnehmung der Umgebung wird mit solchen Netzen bewerkstelligt. Dabei profitiert nicht nur das autonome Fahren von diesen Möglichkeiten. Erkennen von Spielzuständen bei Computerspielen oder bei der Interaktion mit dem Menschen gehören ebenso zu bereits implementierten Systemen. Das Unternehmen OpenAI nutzte künstliche Netze, um die Position und Orientierung eines Rubik Cubes in ein Computerprogramm einzulesen. Ihnen gelang es, einer Roboterhand das Lösen eines Rubik Cubes beizubringen [1].

Auch in der Ökonomie, insbesondere im Finanzsektor, werden eine Vielzahl von Computerprogrammen verwendet, darunter auch neuronale Netze, um Börsenkurse vorherzusagen und schnell auf Änderungen reagieren zu können. Wer heutzutage ein technisches Gerät, wie einen Computer, Tablet oder Smartphone besitzt, ist sicher schon mit Sprachassistenten, wie Google Assistant, Siri oder Alexa in Berührung gekommen. Dort kann jeder persönlich die deutlichen Fortschritte der Technologie selbst miterleben. Aber auch die Erkennung von Handschriften in mehreren Apps und sogar das Antworten auf komplexe Fragen, wie es einige Chatbots tun, ist in der heutigen Zeit bereits mehr als nur ein Forschungsprojekt und in den sozialen Medien angekommen. Aber neuronale Netze sind noch zu deutlich mehr in der Lage. In verschiedenen Papers wurde gezeigt, dass das Komponieren von Songs [2, 3] oder das Kreieren von kreativer Kunst [4] ebenso in den Möglichkeiten der Netze liegt wie eine Vielzahl an weiteren Dingen.

Auch in den Ingenieurswissenschaften und in der Medizintechnik hat das Zeitalter der neuronalen Netze begonnen. Da sich diese Arbeit tiefer mit den dort verwendeten Technologien auseinandersetzt, werden die technologischen Gesichtspunkte unter dem Gliederungspunkt 5 Stand der Technik beschrieben. Die Vorteile, welche künstliche Intelligenz in die Medizin bringt sind weitgreifend. Neuronale Netze können vor allem dazu verwendet werden den Ärzten unter die Arme zu greifen und Sie bei Ihren Diagnosen zu

unterstützen. Vorschläge zu Behandlungen und Hinweise auf mögliche Nebenwirkungen mit dem Gesichtspunkt auf Vorerkrankungen und die Familiengeschichte, können direkte Auswirkungen auf die Qualität der Behandlung haben. Aber auch, wenn repetitive Tätigkeiten von der KI übernommen werden und Formalien dem Arzt erspart bleiben, kann sich dieser intensiver mit dem Patienten auseinandersetzen. Andere Länder, welche nicht den medizinischen Stand der Industrieländer haben, können durch solche Systeme noch deutlich stärker unterstützt werden. Muss zu einer Diagnose kein Arzt mehr anwesend sein, sondern nur Fachpersonal, welches die Empfehlungen der Systeme umsetzt, kann dies in einigen Teilen der Welt bereits einen deutlichen Anstieg in der Verfügbarkeit von medizinischen Leistungen bedeuten.

Die Fülle an Neuerungen und Einsatzgebieten lässt sich kaum noch überblicken und täglich werden neue Arbeiten zu verschiedenen Netzen und Systemen veröffentlicht. Die Entwicklung, welche in den letzten Jahren zu beobachten war, lässt weiterhin auf enorme Fortschritte in den unterschiedlichsten Einsatzbereichen in den nächsten Jahren schließen und künstliche Intelligenz wird auch weiter Einzug in den Alltag vieler Menschen halten.

4.2 Problemstellung

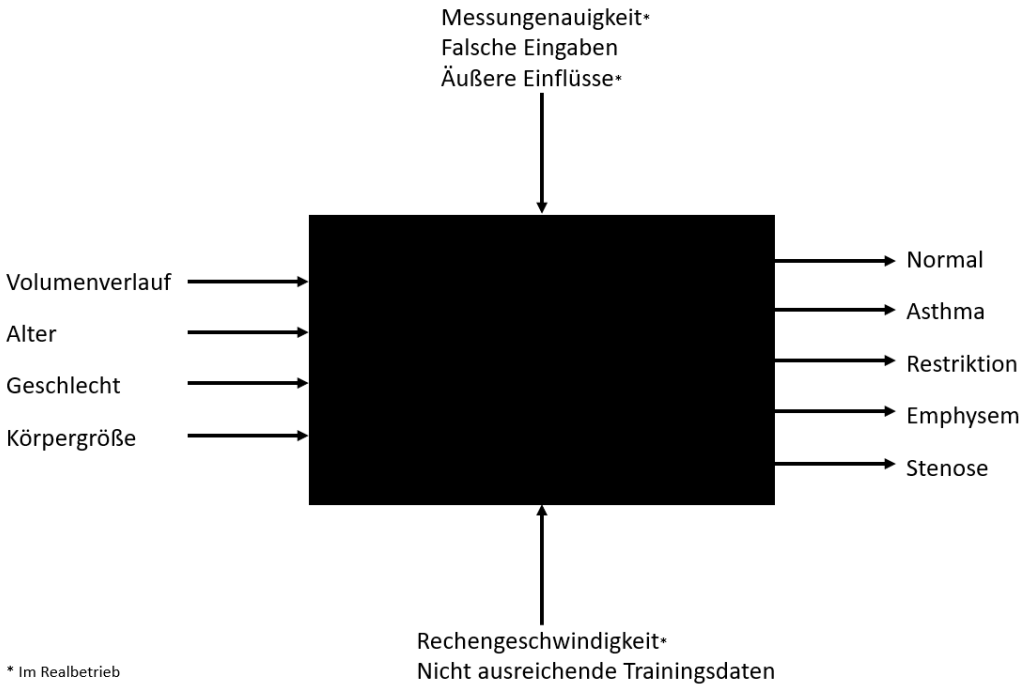
Wie einleitend beschrieben, setzt sich diese Arbeit mit dem Einsatz von KI im medizinischen Bereich auseinander. Bei dem Einsatz von patientennahen Maschinen werden Daten meist automatisch generiert, welche dem Arzt visuell dargestellt oder Algorithmen zur Verfügung gestellt werden, um daraus den Zustand eines Patienten abzuleiten. Ein Beispiel, welches besonders in der momentan herrschenden Corona-Pandemie immer mehr in den Betrachtungsraum der Medien gerückt ist, ist das Beatmungsgerät. Bei diesem können die Daten des Patienten bereits oft gespeichert werden. Allerdings ist es schwer an einen solchen Datensatz zu gelangen. Und auch dann stellt die Interpretation der Daten eine große Herausforderung dar, da die vorhandenen Daten meist nicht mit der dazugehörigen Diagnose beziehungsweise dem Zustand des Patienten versehen sind. Besitzt man einen Datensatz von bereits bearbeiteten Daten, geht es darum, die Daten richtig zu klassifizieren, um den Arzt oder das medizinische Fachpersonal mit den gewonnenen Erkenntnissen zu unterstützen.

4.3 Aufgabenstellung

Die wichtige Voraussetzung der folgenden Ausarbeitung liegt in der Generierung möglichst realer Datensätze. Da es in Deutschland aufgrund der Datenschutzgesetze nahezu unmöglich ist, an reale Patientendaten zu gelangen, muss zuerst ein Datensatz künstlich mit einem sogenannten Datengenerator erzeugt werden. Hierzu müssen die technischen Möglichkeiten geprüft werden, welche ein Beatmungsgerät mit sich bringt. Daraufhin können die verschiedenen Parameter, welche zur Verfügung stehen, auf ihre Tauglichkeit zur medizinischen Diagnose geprüft werden. Ebenso muss der Einfluss von

personenspezifischen Merkmalen, wie Geschlecht, Alter und Größe, auf den ausgewählten Parameter untersucht werden. Hierzu bietet sich eine Recherche in der medizinischen Fachliteratur an. Zur Klassifizierung der Daten sollen neuronale Netze verwendet werden. Hierbei ist es notwendig die unterschiedlichen Architekturen der Netze zu untersuchen und eine geeignete Architektur auszuwählen. Bei dieser Recherche werden die technischen Alternativen, welche ingenieurswissenschaftliche Anwendungen, wie die prädiktive Wartung, mit sich bringen, ebenfalls in Betracht gezogen und auf eine Übertragung auf die erzeugten Daten hin begutachtet. Durch die Variation der Struktur des Netzes sowie dessen Hyperparameter, soll dann ein passendes Netz kreiert werden, welches in der Lage ist, zuverlässig eine Klassifizierung der erzeugten Daten vorzunehmen.

4.4 Blackbox



Das in dieser Arbeit entwickelte neuronale Netz kann als Blackbox dargestellt werden. Die Eingaben müssen nach dem Training des Netzes eine gleichbleibende Form aufweisen. Die Eingänge bilden die personenspezifischen Daten des Patienten und dessen Volumenverlauf. Bei den Ausgängen des Systems handelt es sich die Ausgangsneuronen des Netzes, also um die zuvor definierten Klassen. Neben dem Normalfall werden vier weitere Krankheitsbilder betrachtet.

Restriktionen, welche das System daran hindern das gewünschte Maß an Genauigkeit zu erreichen, sind zum einen eine zu geringe Anzahl an Trainingsdaten. Dies betrifft sowohl den Entwicklungsprozess als auch den späteren Einsatz am Menschen. Reichen die vorhandenen Trainingsdaten nicht aus ist eine Generalisierung des Netzes

schwieriger zu erreichen. In der realen Anwendung ist es schwieriger einzuschätzen, welche Menge an Daten notwendig ist, um ein dauerhaft gutes Ergebnis zu erlangen. Aufgrund der großen Varianz zwischen den Menschen, können trotz einer guten Genauigkeit auf den Trainings- und Testdaten Sonderfälle nicht in den Daten berücksichtigt worden sein.

Ein Thema was die Nutzung des Systems in der realen Anwendung einschränkt ist die Rechengeschwindigkeit des Gesamtsystems. Ist das System nicht in der Lage die erhaltenen Daten innerhalb der Abtastfrequenz zu verarbeiten, kann das System in dieser Form nicht eingesetzt werden. Da neuronale Netze jedoch meist sehr performant, insbesondere auf dafür entwickelten Chips oder Grafikkarten, laufen, dürfte dies in der Praxis selten eine echte Herausforderung für den Entwickler darstellen.

Falsche Eingaben beziehungsweise Datensätze können bereits im Training die Ergebnisse verfälschen und das System für die Implementierung in ein Gesamtsystem unbrauchbar machen. Aber auch durch die Interaktion mit einem menschlichen Bediener können Fehleingaben und Tippfehler entstehen und zu Fehldiagnosen führen.

Die technischen Möglichkeiten des Systems wirken sich direkt auf die Ergebnisse nach der Verarbeitung aus. Sind die Eingaben durch Messungenauigkeiten oder äußere Einflüsse, wie Interferenz oder Stöße, verändert besteht auch hier bei zu großen Abweichungen wieder die Gefahr einer falschen Klassifikation.

4.5 Organisation der Arbeit

In diesem Abschnitt soll der rote Faden, welcher sich durch die Ausarbeitung zieht, aufgezeigt werden. An diesem Punkt sollte der Leser bereits einen ersten Einblick in das Thema Künstliche Intelligenz und neuronale Netze sowie in die Aufgabenstellung bekommen haben.

Im folgenden Kapitel Stand der Technik soll der Überblick insbesondere in Bezug auf die verwendeten Netzarchitekturen vertieft werden. Die technologische und mathematische Beschreibung der Basis Architekturen sowie darauf aufbauende Sonderfälle, welche bezüglich der Aufgabenstellung von Interesse sind, finden sich in Kapitel 7 wieder. An dieser Stelle soll zuerst einmal ein genereller Überblick über den momentanen Stand der Technik gegeben werden, um eine Einordnung des hier behandelten Themas zu erlauben.

In Kapitel 6 wird das Thema Daten behandelt. Dabei wird zunächst eine Unterscheidung der Daten vorgenommen, bevor die theoretischen Grundlagen und Hintergründe zur Erstellung der Daten erläutert wird. Aufbauend darauf wird die Erzeugung der Daten am Beispiel des Datengenerators gezeigt. Am Ende des Kapitels sind die Daten und damit die Grundlage für das Training der neuronalen Netze geschaffen.

Um eine passende Netzarchitektur und Hyperparameter auswählen zu können, werden in Kapitel 7 die theoretischen Grundlagen, Modelle und Funktion, welche für die

Aufgabenstellung von Interesse sind, begutachtet. Am Ende des Kapitels kann dann anhand dieser Ausführungen eine Auswahl vorgenommen werden.

Kapitel 8 beschäftigt sich mit der praktischen Umsetzung des zuvor Besprochenen. Aufbauend auf der getroffenen Auswahl, wird die Vorgehensweise und die Ergebnisse der verschiedenen Tests thematisiert. In diesem Kapitel findet sich der Hauptteil der praktischen Arbeit wieder und erläutert die sich auf dem Speichermedium befindlichen Programme.

Zuletzt wird eine kurze Zusammenfassung der Ergebnisse aus den vorherigen Kapitel gegeben sowie ein Ausblick bezüglich der Weiterentwicklung der Thematik.

5 Stand der Technik

In dieser Arbeit soll auch der Transfer von ingenieurswissenschaftlichen Technologien zu medizintechnischen Anwendungen geprüft werden. Ein Gebiet, welches dabei besonders im Fokus liegt, ist die prädiktive Wartung. Durch den Einsatz von prädiktiver Wartung in Kombination mit künstlicher Intelligenz lassen sich eine Vielzahl an Vorteilen generieren. Die Senkung der Reparatur- und Ersatzteilkosten, die erhöhte Betriebssicherheit sowie eine erhöhte Produkt- und Prozessqualität sind nur einige der Faktoren, welche insgesamt zu einer Leistungssteigerung führen. Trotzdem hat das Thema die Industrie noch nicht in ihrer Gesamtheit durchdrungen. Laut einer Umfrage des VDMA boten 2017 gerade einmal knapp 40 Prozent der befragten Unternehmen entsprechende Technologien und Dienstleistungen an [5]. Weitere Befragungen legen nahe dass der relative Anteil der Unternehmen, welche die angebotenen Technologien bereits nutzen, nur ein Bruchteil der zuvor genannten 40 Prozent beträgt [6].

Heutzutage gibt es viele verschiedene Methoden, mit welchen sich eine prädiktive Wartung realisieren lässt. Grob lassen sich diese Methoden in vier Kategorien einteilen: physikalische, wissensbasierte und Daten-getriebene Modelle sowie die Kombination aus diesen. Aufgrund der Herangehensweise dieser Arbeit sind jedoch nur die künstlichen neuronalen Netze, welche in die Kategorie Daten-getrieben fallen, von Interesse, auch wenn in der Praxis oft stochastische oder statistische Modelle verwendet werden [7].

In der Industrie sowie in Veröffentlichungen finden sich eine Vielzahl an verschiedenen Anwendungsbereichen. Dementsprechend unterscheiden sich auch die Herangehensweisen der verschiedenen Autoren sowie die Wahl der Architekturen ihrer Netze. Einen ausgeweiteten Überblick über die verschiedenen Architekturen/ Technologien bietet die Quelle [8]. Die am häufigsten verwendeten Architekturen sind das Feedforward Neuronale Netz (FNN), das Faltungsnetz (CNN) und das rekurrente Netz (RNN). Diese werden jeweils für einen unterschiedlichen Aufgabentyp eingesetzt. FNNs werden oft verwendet, um Klassifizierungen vorzunehmen oder nicht bekannte Beziehungen zwischen verschiedenen Parametern aufzudecken. Das Netz kann beispielsweise genutzt werden, um anhand verschiedener Sensordaten die Belastung eines Systems aufzuzeigen [9]. CNNs werden meist verwendet, um Bilddaten zu analysieren und Klassifikationen oder Merkmalserkennungen durchzuführen. Jedoch werden sie auch für eine Vielzahl von anderen Anwendungen verwendet. Beispielweise lassen sich auch die bildlichen Darstellungen von Sensordaten oder Geräuschen klassifizieren. RNNs kommen meist bei der Vorhersage von Zeitreihen zum Einsatz. Mit ihnen ist es nicht nur möglich Börsenkurse schon im Voraus zu wissen, sondern auch wichtige Parameter der Produktion können prognostiziert und somit ein Ausfall eines Elementes verhindert werden.

Auch in der Medizin haben die Künstliche Intelligenz und neuronale Netze bereits Einzug gehalten. Firmen wie Siemens Healthineers oder IBM bieten bereits Produkte an, welche die Ärzte bei ihrer Arbeit unterstützen sollen. Ein bereits weit verbreitetes Anwendungsfeld ist die Analyse von Bilddaten, insbesondere Computertomografie-Scans und Röntgenaufnahmen. Aufnahmen von verschiedenen Organen lassen sich bereits mit einer

hohen Genauigkeit analysieren und betroffene Gewebebereiche farblich visualisieren [10]. Ein weiterer Bereich, welcher hier exemplarisch genannt werden kann, um die Reichweite der Technologie aufzuzeigen, ist die Klassifikation von Sprache. Eine Veröffentlichung verschiedener IBM Mitarbeiter zeigt auf, wie die Klassifizierung der ärztlichen Aufnahmen dazu verwendet werden kann, den Inhalt zu identifizieren und Schlussfolgerung daraus abzuleiten [11]. Ein Beispiel, welches in der Einleitung des Papers zu finden ist, verdeutlicht den Nutzen, welchen diese Technologie mit sich bringen kann. "Der Patient lebt bei seiner Mutter, die ihr Zuhause nicht verlassen kann" ist wichtig in einer Pflege-Management-Umgebung. Es ist jedoch schwierig, Folgendes mit vorhandenen Tools zu modellieren: Erstens bezieht sich das Fragment eher auf die Mutter des Patienten als auf den Patienten. Zweitens ist es ein Indikator für eine soziale Ausgrenzung des Patienten, ohne, dass ein Wort verwendet wurde, welches mit sozialer Ausgrenzung an sich verbunden ist.

Wer heutzutage eine Apple Watch trägt, hat auf dieser mit hoher Wahrscheinlichkeit eine App der Firma AliveCor installiert. Mit dieser ist es möglich, nur anhand des Elektrokardiogramms, welches durch die neue Hardware der Uhr erstellt werden kann, Herzrhythmusstörungen und erhöhte Kaliumwerte zu erkennen [12].

Wie schon in den ingenieurswissenschaftlichen Anwendungen, welche zuvor beschrieben wurden, wird auch in der Medizintechnik auf dieselbe Technologie zurückgegriffen. Auch in dieser Sparte finden neuronale Netze mit den zuvor beschriebenen Architekturen ihre Anwendung. So werden, um bei den zuvor beschriebenen Beispielen zu bleiben, im Fall der Bildverarbeitung sowie bei der Sprachklassifikation Faltungsnetze verwendet, um die gewünschten Ergebnisse zu erzielen. Betrachtet man die ingenieurswissenschaftlichen Anwendungen, welche sich mit Maschinen beschäftigen und die medizintechnischen Anwendungen, die meist auf den Menschen abzielen, ergeben sich im groben Überblick nur wenig Gemeinsamkeiten. Bricht man jedoch die technischen Fragestellungen der beiden Disziplinen auf die Datenbasis herunter, stellen sich meist die Fragen und Probleme. Aus diesem Grund finden dieselben Technologien bei, auf den ersten Blick, unterschiedlichen Einsatzbereichen Anwendung.

6 Daten

6.1 Unterschied zwischen Trainings- und Testdaten

Bei künstlichen neuronalen Netzen unterscheidet man typischerweise zwischen der Trainingsphase und der Testphase. In der Trainingsphase werden die Gewichte der Verbindungen anhand von bestimmten Lernregeln (siehe Trainingsverfahren) modifiziert. Diese Regeln beschreiben, wie sich die Gewichtungen während der Trainingsphase verändern. Hierbei wird dem Modell ein bestimmter Prozentsatz der Daten für das Training zu Verfügung gestellt (Trainingsdaten).

In der Testphase werden keine Gewichte verändert. Stattdessen wird der Lernfortschritt des Netzwerkes überprüft. Zum einen können dafür die Trainingsdaten verwendet werden. Zum anderen werden vorher separierte Daten verwendet, welche dem neuronalen Netz bisher noch nicht zugeführt wurden (Testdaten). Auf diese Weise lässt sich feststellen, ob in dem Netz eine Generalisierung stattgefunden hat, also ob das Netz auch über die zu lernenden Reize hinaus in der Lage ist, Aufgaben zu lösen [13].

6.2 Volumen- und Druckverlauf bei der maschinellen Beatmung

Wichtige Parameter bei der künstlichen Beatmung sind der Volumenverlauf und der Druckverlauf. An modernen Beatmungsgeräten kann neben vielen patientenspezifischen Beatmungsmodi in der vollmaschinellen Beatmung generell zwischen einer volumenorientierten und einer druckorientierten Beatmung ausgewählt werden. Bei einer volumenorientierten Beatmungsform ist die Trajektorie des Volumenverlaufs geräteseitig vorgegeben. Das Volumen steigt konstant bis zu einem vorgegebenen Punkt an und stagniert dann während der Pause-Phase, bevor es während der Ausatmung, abhängig von den Werten des Beatmeten, abfällt. Der Druckverlauf stellt bei dieser Beatmungsform die Antwort des Patienten dar. Bei einer druckorientierten Beatmungsform baut das Gerät einen vorgegebenen Druck auf und hält diesen bis zur Ausatmung des Patienten. Der Volumenverlauf kann in diesem Fall als Antwort des Patienten ausgewertet werden. Durch die Ableitung des Volumenverlaufes erhält man den Volumenstrom. Dieser ist im Falle einer volumenorientierten Beatmung konstant, da dieser Parameter von dem Beatmungsgerät geregelt wird.

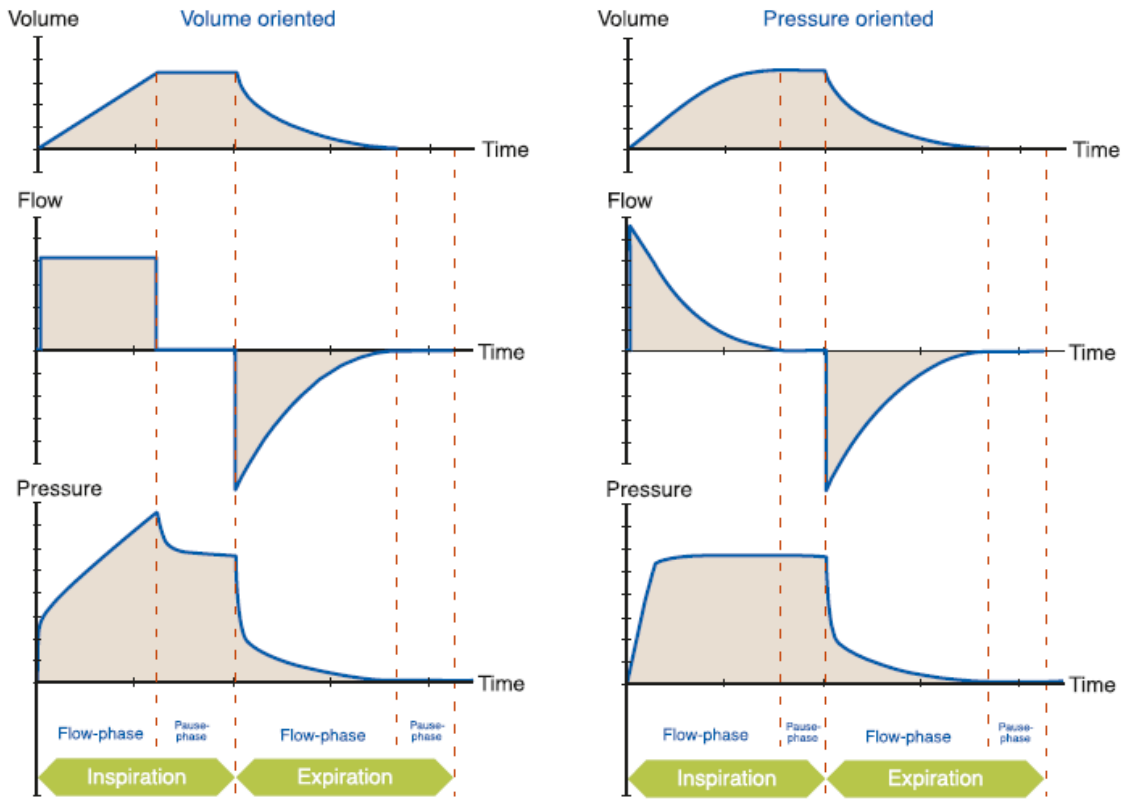


Abbildung 2 Vergleich volumen- / druckorientierte Beatmung [14]

In dieser Ausarbeitung wird exemplarisch und aufgrund der markanten Form, der Druck-Zeit-Verlauf als patientenabhängige Variable bei einer volumenorientierten Beatmungsform verwendet. Bei einer volumenorientierten Beatmungsform haben geräte- und lungenspezifische Resistance- und Compliancewerte (Widerstands- und Dehnungswerte) Einfluss auf den Atemwegsdruck. Da diese Werte jedoch geräteseitig konstant sind, gestattet das Druck-Zeit-Diagramm Rückschlüsse auf den Lungenstatus des Patienten und seine Veränderungen [14].

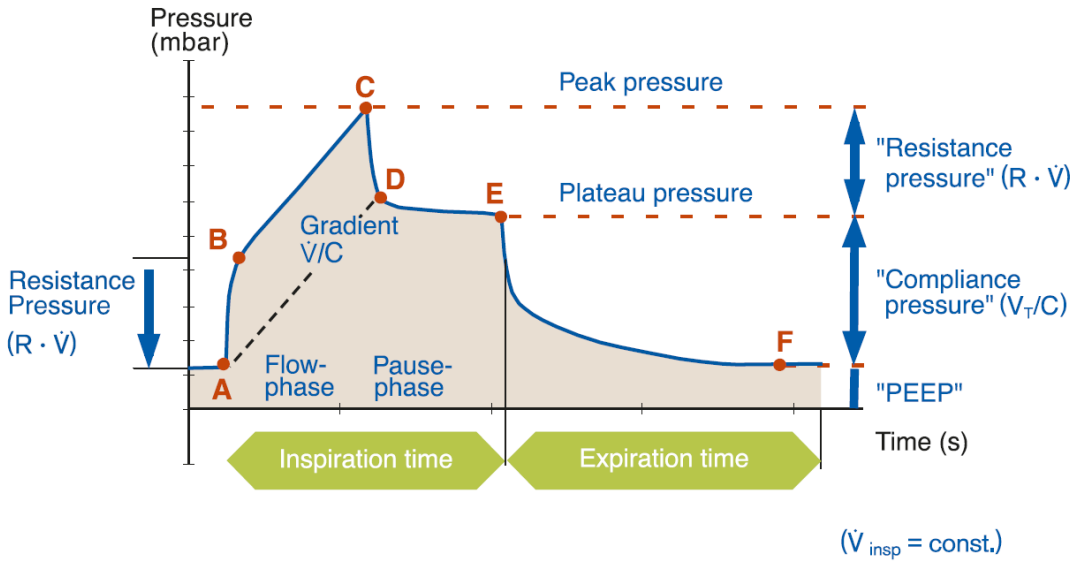


Abbildung 3 Druck-Zeit-Verlauf bei volumenkontrollierter Beatmung [14]

Die folgende Beschreibung des Schaubildes kann in ihrer Gesamtheit in [14] nachvollzogen werden. Zwischen den Anfangspunkten A und B ist ein starker Anstieg des Druckes zu erkennen. Dieser resultiert aus den Widerständen des Systems. Die Höhe des Punktes B kann, wie in Abbildung 3 zu sehen, aus dem Produkt der Resistance (Atemwegswiederstand) und dem Flow ($\dot{V}$) berechnet werden.

$$\Delta p = R \cdot \dot{V} \tag{6.1}$$

Dies gilt jedoch nur, wenn kein intrinsischer PEEP vorliegt. Also wenn der Druck in der Lunge nach dem Ausatmen nicht größer ist als der an dem Gerät eingestellte Wert. Dies lässt zum Beispiel auf eine obstruktive Atemwegserkrankung schließen [15]. Zwischen Punkt B und C steigt der Druck linear abhängig von dem Inspirationsflow $\dot{V}$ und der Gesamtcompliance C.

$$\frac{\Delta p}{\Delta t} = \frac{\dot{V}}{C} \tag{6.2}$$

In Punkt C wurde das eingestellte Tidalvolumen erreicht und der Flow fällt auf null, was ein Absinken des Druckes zur Folge hat. Dieser Abfall entspricht dem umgekehrten, zuvor beschriebenen Anstieg, zwischen Punkt A und B, weshalb die Basislinie zwischen den Punkten A und D parallel zur Strecke B-C verläuft.

Durch Wiedereröffnung von Lungenbereichen, sowie Undichtigkeiten im System, kann der Druck in der Pause-Zeit leicht absinken (D-E).

In Punkt E beginnt die Exspirationsphase. Dies ist ein passiver Vorgang. Aufgrund der elastischen Rückstellkräfte des Thoraxes wird die Luft gegen den atmosphärischen

Druck aus der Lunge gedrückt. Die Druckänderung kann dabei durch das Produkt aus dem Ausatmungswiderstand R des Beatmungsgerätes und dem Exspirationsflows $\dot{V}_{exsp}$ beschrieben werden.

$$\Delta p = R \cdot \dot{V}_{exsp} \tag{6.3}$$

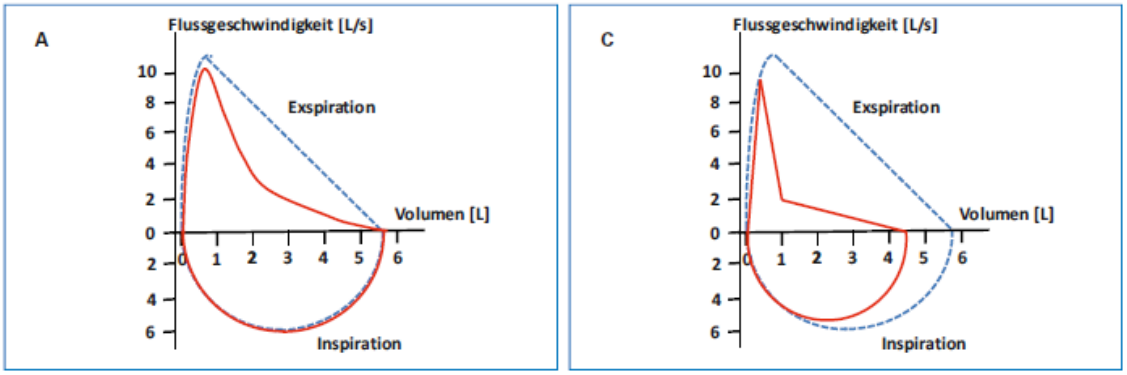
In Punkt F wird anschließend dann das endexspiratorische Niveau F (PEEP) erreicht.

Anhand dieses Verlaufs können nun Beispieldaten erzeugt werden, welche als Trainings- und Testdaten für das KNN dienen.

6.3 Spirometrie

„Die Spirometrie ist eine einfache Lungenfunktionsuntersuchung zur Diagnostik obstruktiver Ventilationsstörungen und zur Bestimmung von Lungenvolumina [16]“. Bei der Spirometrie lässt sich durch die kontinuierliche Messung eine Aussage bezüglich der Ventilation des Patienten treffen. Durch eine sogenannte forcierte Spirometrie, in der der Patient ein willkürliches und maximales Atemmanöver ausführt, können die definierten Volumina und Atemstromstärken bestimmt werden [16]. Zu den großen Vorteilen dieser Methode zählt, dass sie kostengünstig, leicht durchzuführen und nicht invasiv ist. Gemessen werden die Werte entweder durch einen Strömungssensor, wie ein Ultraschallsensor oder ein Hitzdrahtanemometer. Bei dieser Methodik wird das Volumen durch die Integration der Werte bestimmt. Die andere Möglichkeit besteht in der Volumenmessung, wie durch eine Turbine. Hier werden die Werte differenziert, um an die Strömungswerte zu gelangen.

Aus dem Volumenverlauf und dem dazugehörigen Strömungsverlauf, oft auch Fluss oder engl. Flow genannt, lässt sich ein Fluss-Volumendiagramm erstellen. Dieses dient den Ärzten als Indikator für verschiedene Restriktionen und Obstruktionen in den verschiedenen Lungenbereichen und den Atemwegen. Anhand der Veränderung der Kurve zur Normalform lassen sich, zusammen mit anderen Indikatoren, Diagnosen stellen. Beispielhaft werden hier vier verschiedene charakteristische Kurven gezeigt, welche aus [17] entnommen wurden.



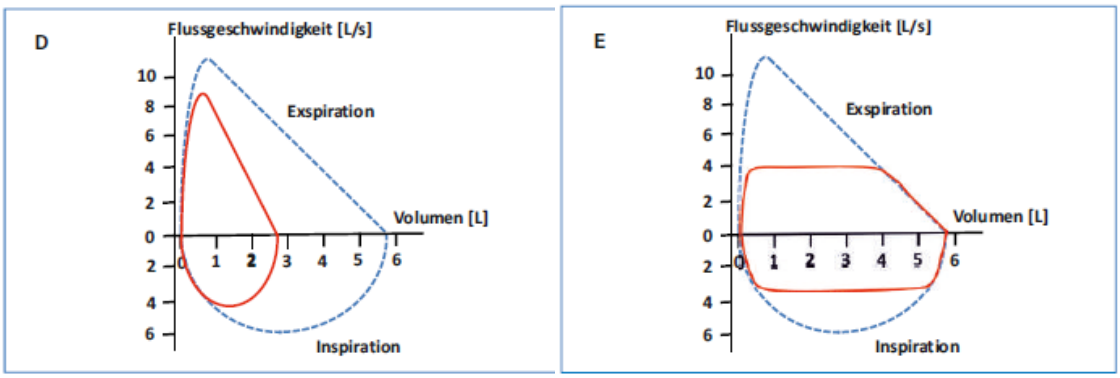


Abbildung 4 Flussvolumenkurven Asthma, Emphysem, Restriktion, Stenose [17]

In den obigen Beispielen sind die standardisierten Flussvolumenkurven von Asthma bronchiale (A), einem Lungenemphysem (C), einer Restriktion (D) und einer fixierten trachealen Stenose zu sehen. Die blau gestrichelte Kurve stellt den Normalzustand bei einem gesunden Patienten dar.

Bei Asthma handelt es sich um eine chronische Entzündung der Bronchien, wobei sich die Atemwege verengen [18]. Diese Obstruktion lässt sich auch an der Kurve beobachten. Der Fluss, also die ausgeatmete Luftmenge bricht im Vergleich zu einem gesunden Patienten ein.

„Bei einem Lungenemphysem sind die Lungenbläschen (Alveolen), an denen der Austausch von Sauerstoff und Kohlendioxid stattfindet, teilweise zerstört und überdehnt, sodass ihre innere Oberfläche verkleinert ist. In der Folge ist die Ausatmung erschwert, weil die kleinen Bronchien, welche in die Lungenbläschen münden, in sich zusammenfallen [19]“. In der Kurve äußert sich dies durch einen deutlichen Knick bei der Ausatmung sowie durch ein verringertes Atemvolumen des Patienten.

Bei einer restriktiven Lungenerkrankung ist die Flexibilität und damit die Entfaltung der Lunge betroffen. Dies äußert sich vor allem in einem deutlichen Rückgang der Totalkapazität und dadurch auch meist in der Flussgeschwindigkeit des Patienten.

Unter einer trachealen Stenose versteht man die Verengung (Stenose) der Luftröhre. Dies kann eine Vielzahl an unterschiedlichen Gründen haben und führt bei den Betroffenen meist zu Atemnot. Durch die Verengung ist der maximale Fluss begrenzt. Dies lässt sich an dem konstanten Verlauf der Kurve in den Mittelbereichen erkennen.

Wie an den obigen Beispielen zu sehen ist, kann durch die Form der Kurve schnell auf eine mögliche Ursache geschlossen werden. Aus diesem Grund wird in der medizinischen Praxis eine Kombination aus Volumenverlauf und Flussgeschwindigkeit verwendet. Dies ist allerdings bei maschineller Betrachtung der Daten nicht notwendig, da wie zuvor beschrieben, beide Kurven durch Integration und Differentiation die Information der jeweils anderen Kurve bereits beinhalten. Daher wird bei der Erstellung des Datensatzes nur auf den Volumenverlauf der genannten Störungen zurückgegriffen.

6.4 Personenspezifische Daten

Neben den bereits vorhandenen Daten des Beatmungsgerätes können dem neuronalen Netz auch personenspezifische Daten zugeführt werden, um die Vorhersage und Diagnose zu verbessern. Gewisse Parameter sind zum Beispiel abhängig von Gewicht und Größe des Patienten. Besondere Erkrankungen treten besonders häufig im Alter auf oder sind geschlechtsspezifisch. Aufgrund all der bekannten sowie auch der unbekannten Zusammenhänge sollten personenspezifische Daten ebenfalls als Eingabeparameter in Betracht gezogen werden.

6.5 Fehlerquellen

In der Praxis können eine Reihe an Ursachen die Daten verfälschen und das Training sowie die Anwendung der Netze erschweren. Äußere Einflüsse in der Messumgebung sowie Messungenauigkeiten durch das System selbst, können auf die Aufnahmen des Patienten einwirken. Aber auch menschliche Fehler lassen sich nicht ausschließen. Dies beginnt schon mit der Falscheingabe eines Geburtsdatums und folgt in einer falschen Korrelation, welche zu einer falschen Diagnose führen kann. Systeminterne Resistance und Compliance von verschiedenen Beatmungsgeräten machen eine direkte Übernahme der trainierten Systeme und Daten schwierig und können bei Nichtanpassung zu deutlichen Genauigkeitseinbrüchen führen.

Auch vor der realen Anwendung des Systems in der Praxis können bei dem Training einige Faktoren das Ergebnis verfälschen. Hier spielt der Faktor Mensch eine große Rolle, da die Daten zumeist von Menschen beschriftet werden. Dies lässt nicht nur eine falsche Übertragung der Daten zu, sondern auch in Sonderfällen eine generell falsche Diagnose. Die zu geringe Anzahl an Trainingsdaten lässt sich anhand des Trainings möglicherweise nicht direkt erkennen, kann aber in der Praxis insbesondere durch die große Varianz des menschlichen Körpers zu Fehldiagnosen führen.

6.6 Generierung / Datengenerator

Die Grundlage für die generierten Daten bildet die in Kapitel 6.3 besprochene Spirometrie. Die erste Version des Datengenerators basierte auf der in Kapitel 6.2 erläuterten maschinellen Beatmung. Jedoch wurden diese Daten auf Wunsch nicht weiter verfeinert und für das Training verwendet. Aus diesem Grund wird auch nur die neuste Version des Spirometrie-Datengenerators in diesem Kapitel behandelt.

Da es nur notwendig ist, ein Signal (Volumen oder Fluss) dem neuronalen Netz zu übergeben, um eine Klassifizierung vornehmen zu können, wird in dieser Arbeit exemplarisch der Volumenverlauf verwendet. Dieser ist allerdings durch den Fluss austauschbar. Das neuronale Netz muss dann nur auf die neuen Daten eingelernt werden, ohne dass die grundlegende Struktur des Netzes verändert werden muss.

6.6.1 Struktur der generierten Daten

Der Datensatz wurde in einem Python-Dictionary organisiert, wodurch einfach auf die unterschiedlichen Daten zugegriffen werden kann. „Ein Dictionary ist ein assoziatives Feld. Ein Dictionary besteht aus Schlüssel-Objekt-Paaren. Zu einem bestimmten Schlüssel gehört immer ein Objekt“ [20]. Während dieses Format für die Erstellung der Daten sowie die Übersicht sehr von Vorteil ist, müssen die Daten allerdings in Arrays umgewandelt werden, um die Netze trainieren zu können. Für das Training der Netze wird zum einen ein Eingangsvektor benötigt und zum anderen ein Ausgangsvektor. Ein solcher Eingangsvektor oder Array sieht dann wie folgt aus:

[Kurvenwerte (75 Werte), Körpergröße, Alter, Geschlecht]

Da die Atmung auf einen Zeitraum von drei Sekunden normiert und eine Abtastfrequenz von 25 Hertz festgelegt wurde, ergeben sich daraus 75 Werte, welche die erzeugte Kurve repräsentieren. Hinzu kommen personenspezifische Daten, die die Maximalwerte der Kurve bestimmen.

Ein Ausgabevektor besteht aus den bei der Klassifikation zur Verfügung stehenden Klassen. In diesem Fall handelt sich dabei um vier Erkrankungen beziehungsweise Symptome und einen Normalzustand. In den erzeugten Beispielen wird jede Kurve direkt einer Klasse zugeordnet. Dies bedeutet in dem erzeugten Vektor wird je nach Kurve ein Wert auf Eins gesetzt, welcher die Klassenzugehörigkeit bestimmt. Ein solcher Ausgangsvektor weist folgende Struktur auf:

[Normal, Emphysem, Restriktion, Stenose, Asthma]

6.6.2 Mathematische Beschreibung der Kurven

In der Literatur lassen sich keine mathematischen Beschreibungen der verschiedenen Volumenverläufe finden. Aus diesem Grund mussten die benötigten Funktionen anhand der zur Verfügung stehenden Informationen approximiert werden.

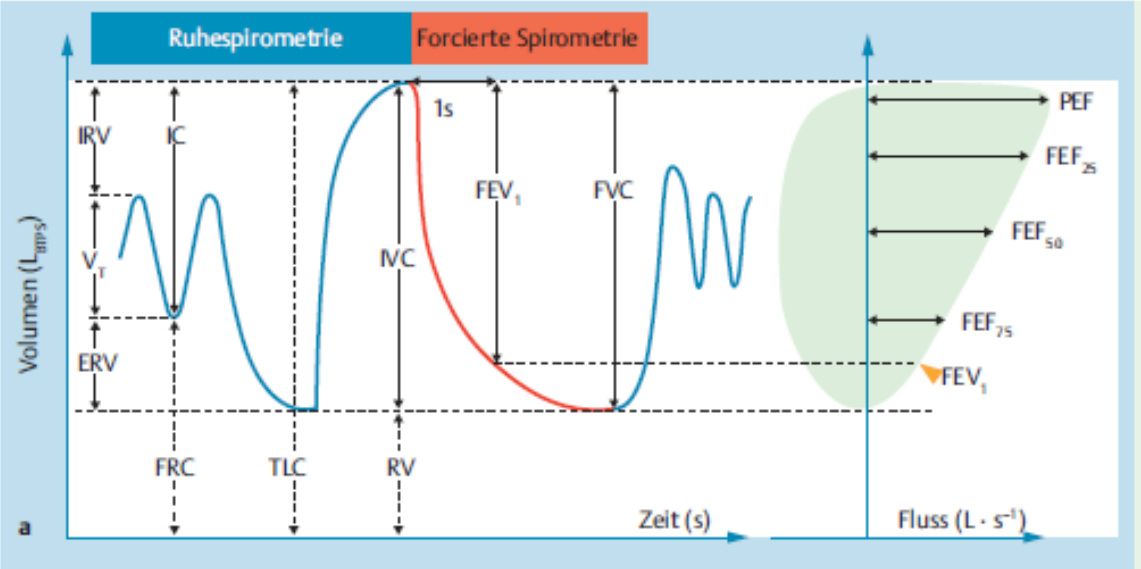


Abbildung 5 Forcierte Spirometrie [16]

Anhand Abbildung 5 lässt sich die Ableitung des, auf der rechten Seite zu erkennenden, Fluss-Volumen-Diagramms ausdrücken. Da insbesondere Daten zu diesem vorhanden sind, empfiehlt es sich von diesem Diagramm auszugehen und auf den eigentlichen Volumenverlauf der zu behandelnden Beispielen zu schließen.

Zunächst einmal lässt sich die Darstellung in zwei Bereiche von Interesse einteilen. Zum einen ist dies die forcierte Einatmung, welche nach der kompletten Ausatmung vollzogen wird. In der obigen Darstellung beginnt diese an dem ersten globalen Tiefpunkt der Volumen-Zeit-Kurve. Der Patient hat das Residualvolumen, also das Volumen, welches nach maximaler Exspiration aus physikalischen Gründen in der Lunge verbleibt, erreicht und atmet nun so stark er kann ein. Dieser Abschnitt stoppt sobald der Patient das maximale Lungenvolumen erreicht hat, also der globale Hochpunkt erreicht wurde. Dieser entspricht der Summe aus dem Residualvolumen (RV) und der inspiratorischen Vitalkapazität (IVC). Dieser Abschnitt lässt sich durch eine Funktion dritten Grades definieren: $f_1(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$. Die Parameter a, b, c und d lassen sich mithilfe von Stützstellen und eines linearen Gleichungssystemlösers, wie er beispielsweise in der Bibliothek numpy vorhanden ist, bestimmen.

Der zweite Abschnitt, welcher sich von dem Ende des ersten Abschnittes bis zum Erreichen des zweiten globalen Tiefpunktes erstreckt, lässt sich nach eigenen Erfahrungen am besten durch eine Exponentialfunktion annähern: $f_2(x) = a * e^{x \cdot k} + b$. Die Bestimmung der Parameter kann ebenfalls durch Stützstellen, sowie in diesem Fall einem Kurvennäherungs-Algorithmus, wie ihn scipy mit sich bringt, vollzogen werden.

In Abbildung 6 wird auf der linken Abbildung in dem oberen Diagramm der Volumenverlauf dargestellt, welcher nach der zuvor erläuterten mathematischen Beschreibung konstruiert wurde. In der linken, unteren Darstellung ist dieses Volumen gegenüber dem Fluss aufgetragen. Vergleicht man nun dies mit dem blau-gestrichelten Normalfall aus der rechten Abbildung, ist eine Übereinstimmung bezüglich der Form zu erkennen.

Aufgrund der Darstellungsweise, der zeitlichen Normierung der erzeugten Daten und aufgrund der nicht definierten personenspezifischen Daten aus Quelle [17] weichen die Darstellungen bezüglich der Werte voneinander ab. Innerhalb der medizinischen Fachliteratur gibt es unterschiedliche Darstellungen der Fluss-Volumen-Kurve, was die Beschreibung der Achsen betrifft. Da das Fluss-Volumen-Diagramm in der linken Darstellung direkt aus dem darüber angeordneten Volumenverlauf abgeleitet und aufgetragen wird, wird hier aus Konsistenzgründen das Vorzeichen des Flusses beibehalten und auch das Volumen bezüglich der eingeatmeten Menge betrachtet.

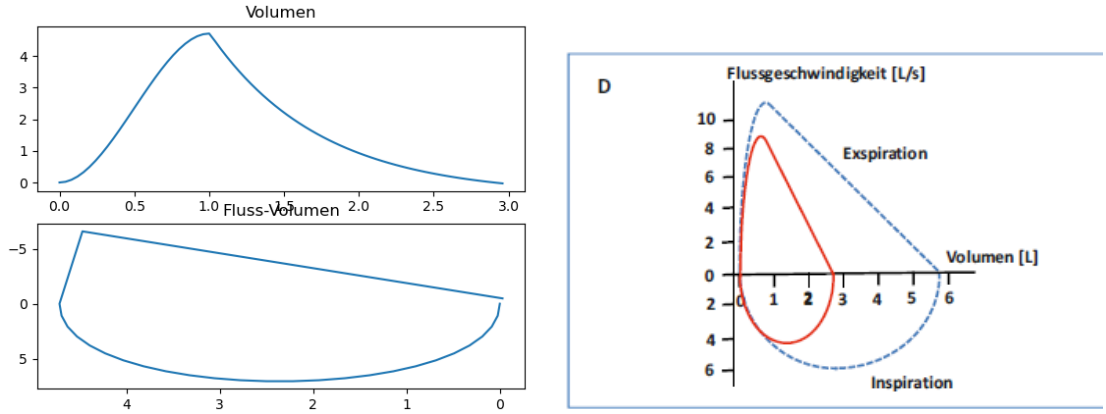


Abbildung 6 Erzeugter Volumenverlauf und Fluss-Volumen-Kurven

Zu sehen ist jedoch, dass die Fluss-Volumen-Kurven auf beiden Seiten dieselbe Form besitzen und somit dasselbe Symptom widerspiegeln. Wie die Volumen-Zeit-Kurve verändert werden muss, um die anderen Fälle zu erzeugen, wird in Kapitel 6.6.3.4 näher beschrieben und kann dort direkt anhand der Programmierung nachvollzogen werden.

6.6.3 Programmierung

Das folgende Kapitel kann als Dokumentation der programmiertechnischen Aufgabe gesehen werden und dient sowohl zur Beschreibung des Gesamtablaufs als auch zur Erläuterung der einzelnen Funktionen.

Während der Projektzeit wurden mehrere Versionen des Datengenerators erstellt. In diesem Abschnitt wird die am weitesten fortgeschrittene Version erläutert. Alle Versionen finden sich auf dem beigefügten Datenträger wieder.

6.6.3.1 Notwendige Bibliotheken und empfohlene Versionen

Verwaltet wurden alle Bibliotheks-Versionen mithilfe von Anaconda. Mit dieser Open-Source-Distribution lassen sich unter anderem einfach Bibliotheken installieren, verwalten und aktualisieren.

Bibliothek	Version
Numpy	1.17.0
Matplotlib.pyplot	3.2.2
Scipy	1.5.0
Pickle	4.0

6.6.3.2 Hauptteil und Ablauf des Programms

In dem Datengenerator wurden einige der Funktionen ausgelagert, um eine bessere Übersicht zu erreichen. Basisfunktionen, wie die mathematische Beschreibung der Kurven und verschiedene Hilfsfunktionen, müssen deshalb zusätzlich eingebunden werden:

```
import functions as f
```

Der Hauptteil des Datengenerators wird verwendet, um über die verschiedenen Kombinationen der Parameter zu iterieren. In diesem Fall werden Kurven für einen Mann mit einer Körpergröße zwischen 1,65 Meter und 1,95 Meter erzeugt sowie einem Alter zwischen 25 und 70 Jahren.

```
# Generate Data for Male
G = 1
for KG in np.arange(1.65, 1.96, 0.01):
    for A in np.arange(25, 71, 1):
```

Jede Datenreihe wird in einem sogenannten Set organisiert. Diese Sets bestehen aus jeweils einem Array für den Volumenverlauf, die Zeit, den Flow und den Output. Ebenso aus Schlüssel-Objekt-Paaren unter dem Schlüsselwort Parameter, welche die Körpergröße, das Alter und das Geschlecht angeben. Eine Funktion, welche ein solches Set erzeugt, wird zu Beginn einer neuen Iteration und Kurve aufgerufen:

```
str_name = create_set(count, KG, A, G)
```

Im Folgenden werden dann die Werte der Kurve durch einen Funktionsaufruf mit den momentanen Parametern erzeugt und zwischengespeichert.

```
x, y, y_abl = f.normal(KG, A, G , Abtastfrequenz)
```

Die erzeugten Werte werden dann dem Dictionary hinzugefügt:

```
append_values(str_name, x, y, y_abl)
```

Sowie der dazugehörige Ausgabevektor:

```
values[str_name]['Output'].append([1, 0, 0, 0, 0])
```

Wünscht der Nutzer eine visuelle Darstellung der erzeugten Werte, kann folgende Zeile auskommentiert werden:

```
# f.save_plot(count, x, y, y_abl, 'normal')
```

Da die Erstellung und Speicherung des Plots jedoch prozentual einen großen Teil der Bearbeitungszeit beansprucht, wird von einer dauerhaften Verwendung abgeraten.

Sind alle Werte erzeugt und in dem Dictionary gespeichert, können die Daten erweitert werden, indem die Kurve verschoben wird:

```
# Erweitere Volumenverlauf durch Verschiebung der Kurve
for sets in values:
    versch = values[sets]['Volumenverlauf'][0].copy()
    for t in range(0, 3*Abtastfrequenz-1):
        versch.append(versch.pop(0))
        values[sets]['Volumenverlauf'].append(versch.copy())
```

Zuletzt müssen dann die Daten gesichert werden. Hierzu wird die Bibliothek pickle und das Dateiformat .pkl verwendet, da diese die Struktur der Dictionaries auch bei erneutem Laden nicht verändert.

```
a_file = open("data_versch.pkl", "wb")
pickle.dump(values, a_file)
a_file.close()
```

6.6.3.3 Hilfsfunktionen

IVC

Die Funktion gibt die inspiratorische Vitalkapazität (IVC) einer Person anhand ihres Geschlechts, ihres Alters und ihrer Körpergröße zurück. Die Herleitung der Mittelwert-Gleichung für die Bestimmung der IVC wird in [16, S.154] beschrieben.

```
def IVC(KG, A, G):
    # Formel aus Paper bekannt
    # Unterscheidung nach Geschlecht
    if G:
        return 6.1*KG - 0.028*A - 4.65
    else:
        return 4.66*KG - 0.024*A - 3.28
```

exponential

Diese Funktion gibt den Wert einer Exponentialfunktion anhand der übergebenen Parameter zurück.

```
def exponential(x, a, k, b):
```

```
    return a*numpy.exp(x*k) + b
```

save plot

Wie die Übersetzung des Namens der Funktion bereits vermuten lässt, ist sie dafür verantwortlich, die visualisierten Daten zu speichern. Mit den unten aufgeführten pyplot-Befehlen lassen sich die Darstellung der Kurven anpassen. Gespeichert wird der Plot dann unter der Nummer des Sets und dem Namen der Klasse, welcher die Kurve zugehörig ist.

```
def save_plot(count, x, y, y_abl, name):
    # Erstellen eines Plots und Speicherung
    pyplot.subplot(211)
    pyplot.title('Volumen')
    pyplot.plot(x,y)
    pyplot.subplot(212)
    pyplot.title('Fluss-Volumen')
    pyplot.plot(y,y_abl)
    pyplot.savefig('Plots/plot_'+str(count)+'_'+str(name)+'.png')
    pyplot.clf()
```

create set

In dieser Funktion werden die einzelnen Schlüssel oder Ebenen des Dictionaries erstellt und dann mit den zuvor erstellten Daten gefüllt.

```
def create_set(count, KG, A, G, x, y, y_abl):
    str_name = 'Set'+str(count)
    values[str_name] = {}
    values[str_name]['Volumenverlauf'] = []
    values[str_name]['Zeit'] = []
    values[str_name]['Flow'] = []
    values[str_name]['Output'] = []
    values[str_name]['Parameter'] = {'Koerpergreße'\
       : KG, 'Alter' : A, 'Geschlecht' : G}
    values[str_name]['Volumenverlauf'].append(y)
    values[str_name]['Zeit'].append(x)
    values[str_name]['Flow'].append(y_abl)
    return str_name
```

6.6.3.4 Funktionen zur Erstellung der Daten

Das Grundprinzip, mit welchem die mathematischen Beschreibungen der Kurven hergeleitet und berechnet werden, lässt sich am einfachsten an der Funktion für den Normalfall eines Volumenverlaufes erläutern.

Normalfall

Die grundlegenden Prinzipien, auf denen die Erzeugung der Kurven aufbauen, werden anhand des Normalfalles verdeutlicht. Im zweiten Abschnitt dieses Kapitels werden dann die Variationen behandelt.

Zu Beginn der Funktion müssen zunächst einmal die später verwendeten Arrays deklariert werden.

```
x = []
y = []
y_abl = []
```

Gespeichert, beziehungsweise erstellt werden, sollen die Zeitwerte, welche sich auf der x-Achse befinden, die Werte für den Volumenverlauf, auf der y-Achse, und die Ableitung dieser, also der Fluss.

Wie bereits zuvor beschrieben, werden die Parameter der Gleichungen bestimmt indem Stützstellen festgelegt werden. Diese wurden aus den vorhandenen Informationen und Quellen abgeleitet.

```
# Punkte definieren
x_A = np.array([0, 0.49, 1 ,2])
y_A = np.array([IVC(KG, A, G), 2.815*(IVC(KG, A, G)/5.63),\
     0.955*(IVC(KG, A, G)/5.63), 0])
```

Da die Punkte anfangs für eine Person mit einem Alter von 25 Jahren und einer Größe von 1,80m erstellt wurden, müssen die Stützstellen der Kurve an die jeweilige Größe und das Alter der momentan betrachteten Person angepasst werden. Zu diesem Zweck wird der Stützpunkt mit dem Term $IVC/5{,}63$ multipliziert.

Wurden die Punkte erstellt, können sie dem Kurvennäherungs-Algorithmus übergeben werden. Dieser gibt dann einen Array mit den gefundenen Parametern zurück.

```
# Funktion mit Punkten approximieren
popt_exponential, pcov_exponential = scipy.optimize.curve_fit(\
    exponential, x_A, y_A, p0=[1,-0.5, 1])
```

Nun werden diese Punkte der Übersicht halber in jeweils eine Variable gespeichert.

```
# Ergebnisse in Variablen speichern
    a = popt_exponential[0]
    k = popt_exponential[1]
    b = popt_exponential[2]
```

Für den Einatmungsvorgang wird eine kubische Gleichung herangezogen. Die Parameter einer solchen Gleichung lassen sich leicht mit einem linearen Gleichungssystem berechnen. Dabei kann nach der Erstellung der Matrix, also der Stützstellen, eine Funktion verwendet werden, um dieses Problem zu lösen.

```
 Definieren eine LGS für den Einatmungsvorgang
```

```
A2 = [[0,0,0,1], [1, 1, 1, 1], [0,0,1,0], [3, 2, 1, 0]]
B2 = [0, IVC(KG, A, G), 0, 0]
# Lösen des LGS
erg = np.linalg.solve(A2, B2)
```

Auch hier werden zur Übersicht die Ergebnisse in einzelne Variablen gespeichert.

```
# Speicherung der Ergebnisse in Variablen
    a2 = erg[0]
    b2 = erg[1]
    c2 = erg[2]
    d2 = erg[3]
```

Nun können die Werte der Kurve erzeugt werden, indem die Zeitwerte der Abtastfrequenz entsprechend eingesetzt und die Ergebnisse dem jeweiligen Vektor hinzugefügt werden.

```
# Hinzufügen der abgetasteten Werte zu Arrays
for x1 in np.arange(0,3, 1/Abtastfrequenz):
    x.append(x1)
    if(x1>1):
        y.append(a*np.exp((x1-1)*k) + b)
        y_abl.append(a*k*np.exp((x1-1)*k))
    else:
        y.append(a2*x1**3 + b2*x1**2 + c2*x1 + d2)
        y_abl.append(3*a2*x1**2+2*b2*x1+c2)
```

Variationen

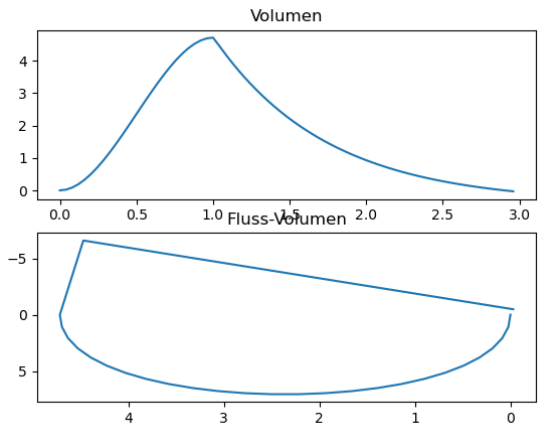


Abbildung 7 Normal

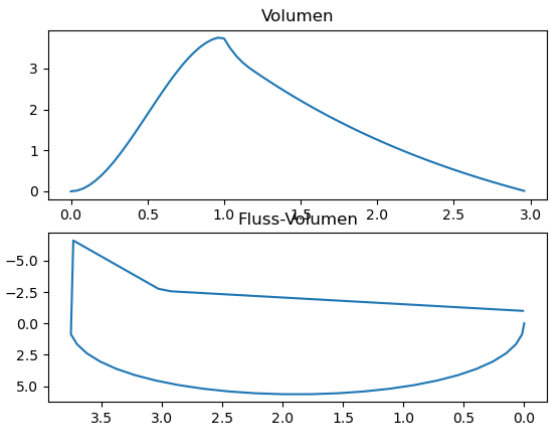


Abbildung 8 Emphysem

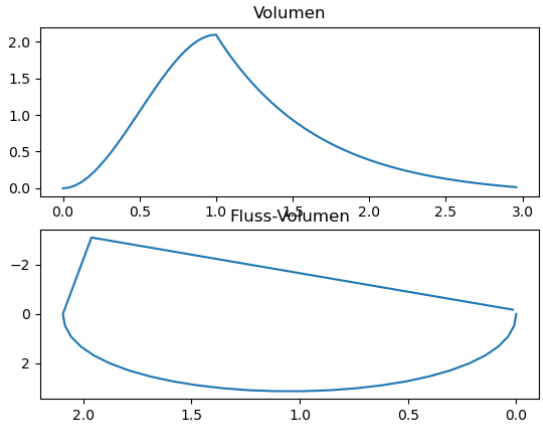


Abbildung 9 Restriktion

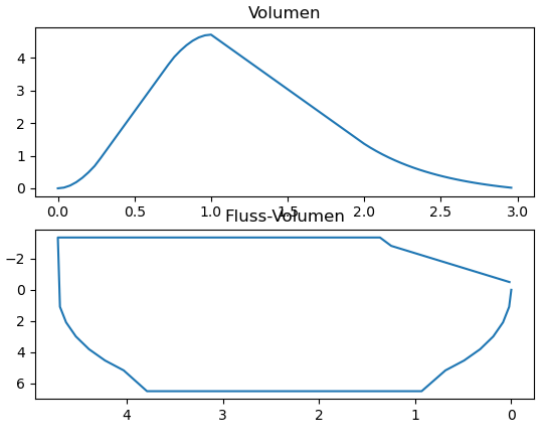


Abbildung 10 Stenose

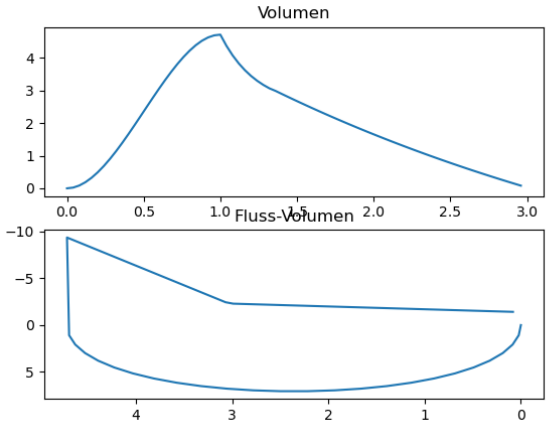


Abbildung 11 Asthma

In Abbildung 7 bis zu Abbildung 11, welche durch den Datengenerator erzeugt wurden, ist zu erkennen, dass sich die Volumenverlaufskurven nur in gewissen Teilabschnitten deutlich voneinander unterscheiden. Mit Ausnahme der Stenose bleibt das Einatmungsmuster der Variationen konsistent gegenüber dem Normalfall. Bei der Restriktion ändert sich zwar das eingeatmete Volumen, die Form der Atmung bleibt aber, wie an der Fluss-Volumenkurve zu erkennen ist, gleich. Um also ein neues Krankheitsbild zu erstellen, reicht es aus, in den gezeigten Beispielen einen Teilbereich der Kurve zu verändern.

Beispielhaft kann ein Asthma-Fall betrachtet werden. In diesem verläuft die Volumenkurve bis zu einem spezifischen Punkt gleich dem Normalfall, bis sie bei dem fortgeschrittenen Ausatmungsvorgang einen fast linearen Verlauf annimmt. Im Code spiegelt sich dies folgendermaßen wider:

```
for x1 in np.arange(0,3, 1/Abtastfrequenz):
    x.append(x1)
    if x1 >1.35:
        y.append(a3*np.exp((x1-1.35)*k3) + b3)
```

```
        y_abl.append(a3*k3*np.exp((x1-1.35)*k3))

    elif(x1>=1):
        y.append(a*np.exp((x1-1)*k) + b)
        y_abl.append(a*k*np.exp((x1-1)*k))

    else:
        y.append(a2*x1**3 + b2*x1**2 + c2*x1 + d2)
        y_abl.append(3*a2*x1**2+2*b2*x1+c2)
```

Vergleicht man nun den Codeausschnitt mit dem zuvor beschriebenen Normalfall, ist ein weiterer Abschnitt in der Generierungssequenz zu erkennen. In diesem wurden die Parameter a3, b3, k3 neu bestimmt. Dies folgt jedoch wieder dem zuvor beschriebenen Muster.

Auch die anderen Fälle lassen sich mit neuen Abschnitten und den bereits erläuterten Methoden zu Bestimmung der Parameter konstruieren, weshalb nicht weiter auf jedes Beispiel eingegangen wird.

6.7 Umgang mit den Daten

Es gibt verschiedene Herangehensweisen, wie man mit den Daten umgehen kann und einige davon finden sich in den Programmen, welche dieser Arbeit beiliegen wieder. Eine Möglichkeit ist es, die Daten direkt als Arrays zu generieren. Dadurch müssen sie im Nachhinein nicht mehr neu strukturiert werden. Allerdings werden dabei Abschläge gemacht, was die Übersichtlichkeit und die Verwechselungssicherheit der Daten angeht. Bei mehreren Versionen von Datensätzen sind diese zwar durch den Dateinamen benannt, besitzen aber nach dem Laden, insbesondere nach einer Normierung, keinerlei Hinweis mehr auf die innere Struktur der Daten. Betrachtet man ein Dictionary, so ist bei richtiger Benennung und Strukturierung dieses, auch nach dem Laden der Daten eine klare Zuordnung der Daten möglich. Aus diesem Grund wurden in dieser Arbeit die Datensätze grundsätzlich einmal in einem solchen Format gespeichert, bevor eine weitere Datei zur Verarbeitung durch die neuronalen Netze erzeugt wurde. Auch nach der Festlegung des Ursprungsformats kann im Verlauf unterschiedlich mit den Daten umgegangen werden. Eine Verarbeitung der Daten durch ein separates Programm im Voraus ermöglicht eine deutliche Zeitersparnis, wenn Trainingsskripte durch Parametrierung und Sweeps häufig aufgerufen werden. Dies ergibt sich logischerweise aus dem Fakt, dass die Daten nicht bei jedem Aufruf erneut neu strukturiert werden müssen, bevor sie verarbeitet werden können. Jedoch ist es nach einer solchen Vorverarbeitung schwieriger und unübersichtlicher, die Daten durch Parameterübergaben, anzupassen. Deshalb sollte auch hierbei zwischen Performance und Einfachheit bei der Programmierung abgewogen werden. Müssen Daten nicht mehr im Trainingsskript verändert werden, bietet es sich an, eingebaute Funktionen wie train_test_split zu verwenden, um Trainings- und Testdaten zu separieren.

Für das Training des neuronalen Netzes und die spätere Klassifizierung sollten die Daten in einem Array gespeichert werden. Der Array hat die Länge der zur Verfügung stehenden Beispiele und jedes Beispiel besteht aus einem Array mit den Eingabewerten. Im Folgenden wird ein Programm zur Erreichung dieser Struktur ausgehend von einem Dictionary erläutert. Dabei können der generelle Aufruf und der Umgang mit einem Python-Dictionary betrachtet werden.

Zu Beginn müssen die erzeugten Daten geladen werden. Hierfür wird, wie bereits zuvor beschrieben, auf die pickle-Bibliothek zurückgegriffen.

```
a_file = open("data_versch.pkl", "rb")
data = pickle.load(a_file)
a_file.close()
```

In diesem Fall sollen die Daten direkt in die endgültige Form gebracht werden, weshalb sie in Trainings- und Testdaten aufgeteilt werden. Ebenso besitzt jeder Eingabevektor einen korrespondierenden Ausgabevektor, wodurch vier Arrays deklariert werden müssen.

```
data_input_train = []
data_output_train = []
data_input_test = []
data_output_test = []
```

Hilfsvariablen werden verwendet, um die Übersicht über die verarbeiteten Daten und die Position in dem Array zu behalten.

```
count = 0
count_train = 0
count_test = 0
```

Nun können die Daten durch eine Schleife aufgerufen werden. Aufgrund von Python kann hier leicht durch den zur Verfügung stehenden Datensatz iteriert werden.

```
for i in data:
    for x in range(0, len(data[i]['Volumenverlauf'])):
```

Trainings- und Testdaten sollen in einem Verhältnis von 2:1 aufgeteilt werden, weshalb jeweils die Beispiele 10 bis 14 den Testdaten zugeordnet werden.

```
        if ((count % 10 == 0 or count % 11 == 0 or count % 12 == 0 or \
       count % 13 == 0 or count % 14 == 0 ) and count != 0):
```

Als nächstes müssen die erzeugten Rohdaten normiert, dem passenden Array hinzugefügt und der Zähler des Arrays erhöht werden.

```
norm_val = [elem/6.85 for elem in data[i]['Volumenverlauf'][x]]
            data_input_test.append(norm_val)
            data_input_test[count_test].append(data[i]['Parameter']['Ko-
erpergroeße']/2)
```

```
            data_input_test[count_test].append(data[i]['Parame-
ter']['Alter']/75)
            data_input_test[count_test].append(data[i]['Parameter']['Ge-
schlecht'])
            data_output_test.extend(data[i]['Output'])
            count_test += 1
```

Auf ein Element des Dictionary kann zugegriffen werden, indem durch die Schlüsselwörter die innere Struktur des Dictionary durchlaufen wird.

`data[i]['Parameter']['Geschlecht']`

data entspricht dem Dictionary an sich und i dem Set, auf welches zugegriffen werden soll. Parameter beinhaltet alle personenspezifischen Parameter und von diesen soll der Wert des Geschlechts entnommen werden. Die Rückgabe entspricht daher einer 0 oder 1. Da diese booleschen Werte nicht mehr normiert werden müssen, können sie direkt dem entsprechenden Array angehängt werden. Derselbe Prozess wird für die Trainingsdaten ausgeführt und nach Durchlauf aller verfügbaren Daten werden die Ergebnisse in verschiedenen Dateien gespeichert. Diese Dateien können dann nach dem Aufruf der Trainingsprogramme geladen werden. Dies folgt in Kapitel 8.

7 Künstliche Neuronale Netze

7.1 Künstliche Neuronen

Abgeleitet aus der Biologie entspricht ein künstliches Neuron in seinen wesentlichen Eigenschaften seinem natürlichen Vorbild. Natürlich wurde bei der Modellbildung das biologische System vereinfacht und verallgemeinert. Die Synapsen der Nervenzellen entsprechen der Addition der gewichteten Eingaben und die Aktivierung des Zellkerns wird durch eine Aktivierungsfunktion mit Schwellenwert abgebildet.

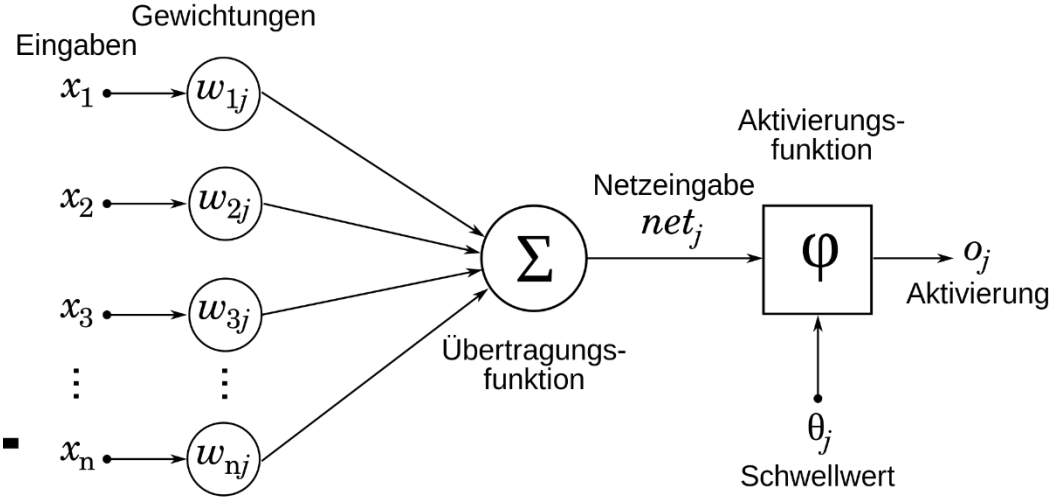


Abbildung 12 Darstellung eines künstlichen Neurons [21]

In Abbildung 12 ist die allgemeine Darstellung eines künstlichen Neurons zu sehen. Dieses dient als Grundeinheit für neuronale Netze. Die Eingaben $x_1, x_2, \dots, x_n$ werden mit den dazugehörigen Gewichtungen $w_{1j}, w_{2j}, \dots, w_{nj}$ multipliziert und anschließend aufsummiert. Gemeinsam mit dem Neuron-eigenen Schwellenwert, welcher der Summe hinzugefügt wird, ergibt sich die Netzeingabe net_j, welche wie folgt definiert ist:

$$net_j = \sum_{i=1}^{n} x_i \cdot w_{ij} + \theta_j \tag{7.1}$$

Dabei ist

net_j	Netzeingabe
x_i	Eingabe i
θ_j	Schwellenwert des Neurons j
w_{ij}	Gewichtung zwischen Neuron i und j

Um die Aktivierung o_j zu erhalten wird die Netzeingabe in eine Aktivierungsfunktion gegeben.

7.2 Basis Architekturen

7.2.1 Feedforward Neuronale Netze/ Multilayer-Perzeptronen

Ein neuronales Netz besteht aus verschiedenen verbundenen Schichten mit Neuronen. Jede Schicht besitzt dabei mindestens ein Neuron. Die erste Schicht wird als Eingabe- und die letzte als Ausgabeschicht bezeichnet. Die dazwischenliegenden Ebenen nennt man verdeckte oder versteckte Schicht. Das Feedforward Neuronale Netz (FNN) wird oft als Standardnetz angesehen und besitzt, wie der Name andeutet, keine Rückkopplungen zu vorrangegangenen Schichten. Ein solches Netz mit mehreren Schichten wird auch als Multilayer-Perzeptron (MLP) bezeichnet. Ein Beispiel für ein solches Netz ist in Abbildung 13 gezeigt.

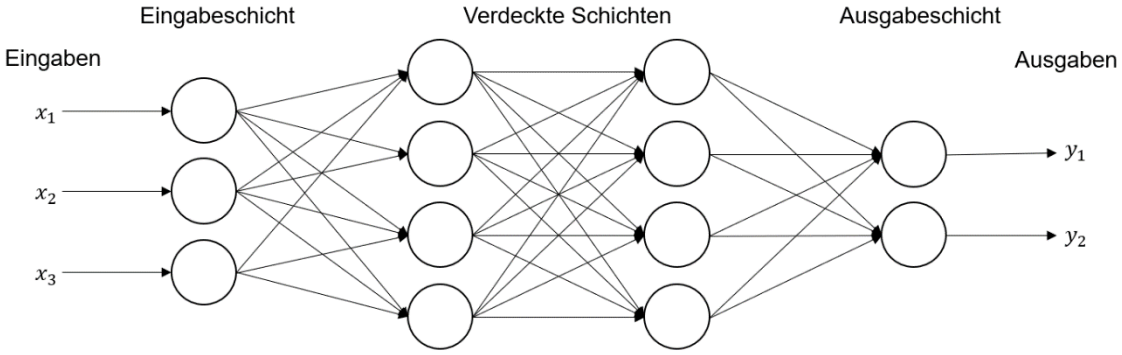


Abbildung 13 Modell eines FNN

Möchte man die Ausgaben y_1, y_2 berechnen, kann dies analog zu einem einzelnen Neuron berechnet werden. Alle Ausgaben der vorliegenden Schicht dienen bei einer Vollverknüpfung als Eingaben der einzelnen Neuronen der nachfolgenden Schicht. Jedoch muss nicht immer ein vollständig konnektiertes KNN vorliegen, dies bedeutet, dass ein Neuron nur mit einem Teil der Neuronen der vorhergehenden Schicht verknüpft sein kann.

Aufgrund der Mehrschichtigkeit dieses Netzwerkes wird meist Backpropagation als Lernmethode verwendet. Solch ein allgemeines KNN lässt sich auf eine Vielzahl von unterschiedlichen Feldern anwenden.

7.2.2 Rekurrente Neuronale Netze

Anders als bei dem zuvor behandelten FNN können im Falle eines rekurrenten neuronalen Netzes (RNN) die Ausgänge der Neuronen Rückkopplungen aufweisen und das Model von der bisher bekannten Verarbeitungsrichtung abweichen. Durch diese Rückkopplungen lassen sich zeitlich codierte Informationen aus Daten gewinnen [22]. Ein RNN erzeugt entweder eine Ausgabe für jede Entität in der Eingabesequenz oder eine einzelne Ausgabe für die gesamte Sequenz [23].

Die RNN lassen sich in verschiedene Rückkopplungsarten unterteilen [24]:

- **Direkte Rückkopplung (direct feedback)**

  Der Neuronen-Ausgang wird als weiterer Eingang für dasselbe Neuron verwendet

- **Indirekte Rückkopplung (indirect feedback)**

  Der Neuronen-Ausgang wird mit einem Eingang eines Neurons der vorhergegangenen Schicht verknüpft

- **Seitliche Rückkopplung (lateral feedback)**

  Der Neuronen-Ausgang wird mit einem Eingang eines Neurons in derselben Schicht verknüpft

- **Vollständige Verbindungen**

  Jeder Neuronen-Ausgang hat eine Verbindung zu jedem anderen Neuronen-Eingang

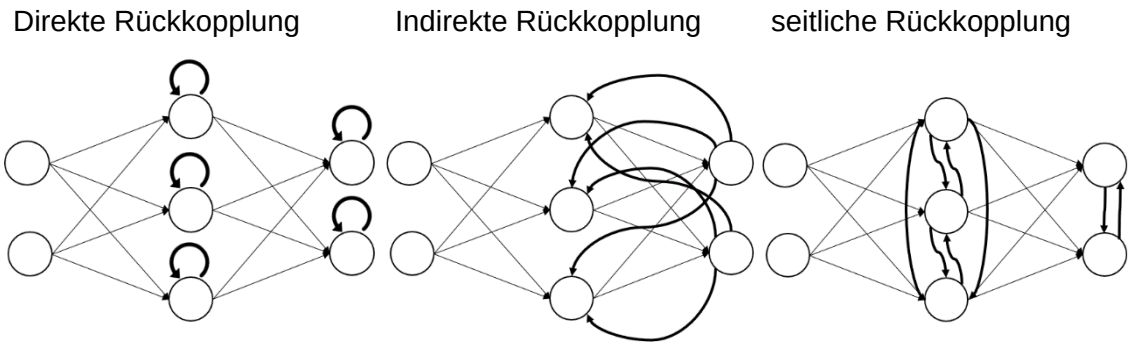


Abbildung 14 Rückkopplungsarten

Aufgrund des Aufbaus sind RNNs in der Lage Daten länger als einen Zyklus zu behalten. Diese Merkfähigkeit prädestiniert sie für den Einsatz in der Zeitreihenvorhersage. Allerdings ist es durch den komplexen Aufbau nicht mehr möglich das Netz durch einen normalen Backpropagation Algorithmus zu trainieren. Ebenfalls ist ein RNN nicht in der Lage, weit zurückliegende Informationen miteinander zu verknüpfen. Versucht man bei einer Architektur mit normalen Neuronen die Abhängigkeiten zwischen Sequenzwerten, welche mehrere Zeitschritte zwischen sich haben, zu modellieren, hängen diese von den Gradienten der Zeitschritte zwischen Ihnen ab. Dies führt dazu, dass der Beitrag des am zeitlich weitesten zurückliegenden Neurons immer kleiner und die Liste von Gradienten, welche Abhängigkeiten besitzen, immer länger wird, wenn man die Zeitschritte rückwärts verfolgt. Dies ist als verschwindendes Gradienten-Problem (engl. vanishing gradient problem) bekannt. Um dieses Problem zu lösen, werden LSTM-Module (Long Short Term Memory) benötigt, welche neben dem üblichen Ein- und Ausgang auch ein Merk- und Vergesstor besitzen.

7.2.3 Faltende Neuronale Netze

Faltungsnetzwerke im englischen Convolutional Neuronal Networks (CNN) genannt, sind eine Sonderform des vorgestellten FNN in Kombination mit einer aus der Signalverarbeitung stammenden Technik zur Filterung von Daten, welche als Faltung bekannt ist.

Die Funktionsweise des CNN basiert auf der Filterung der Eingabedaten. Hierbei durchlaufen die Daten verschiedene Schichten des Netzwerkes, wobei verschiedene Filter angewendet werden. Im Bereich der digitalen Bildverarbeitung wird die Faltung bei der Bildglättung, Bildschärfung sowie als Kantenfilter angewendet. Durch die Bearbeitung lassen sich verschiedene Merkmale hervorheben oder unterdrücken. Ein Objekt besteht aus verschiedenen Merkmalen, zum Beispiel: Kanten, Kreisen oder spezifischen Formen. Durch die Erkennung dieser Merkmale kann eine Einordung dieser Elemente vorgenommen und das Objekt klassifiziert werden. Die Eingabe in die erste Schicht eines CNN besteht zumeist aus zwei oder dreidimensionalen Bildern in Form von Pixelwerten. Während dem Durchlauf werden verschiedene Ausgabewerte berechnet, welche am Ende einer bekannten Klasse zugeordnet werden können [24].

Eine Faltung besteht aus einem Faltungskern, auch Faltungsmatrix genannt, und einem Bild. Der Faltungskern wird über das Bild bewegt. Dabei werden seine Werte mithilfe des Skalarproduktes mit den Werten des Bildes verrechnet [25]. Durch diese Berechnung von korrespondierenden Punkten wird ein neuer oder, abhängig von der Faltung, mehrere neue Pixel erzeugt. Ein Beispiel für eine Faltung ist in Abbildung 12, Abbildung 15 und Abbildung 16 zu sehen.

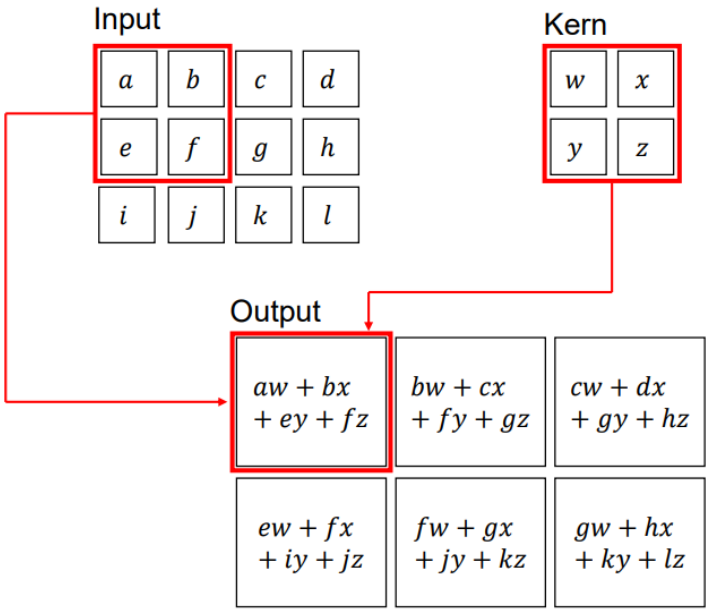


Abbildung 15 Faltung mathematisch [26]

Dies lässt sich mit folgendem Rechenbeispiel verdeutlichen:

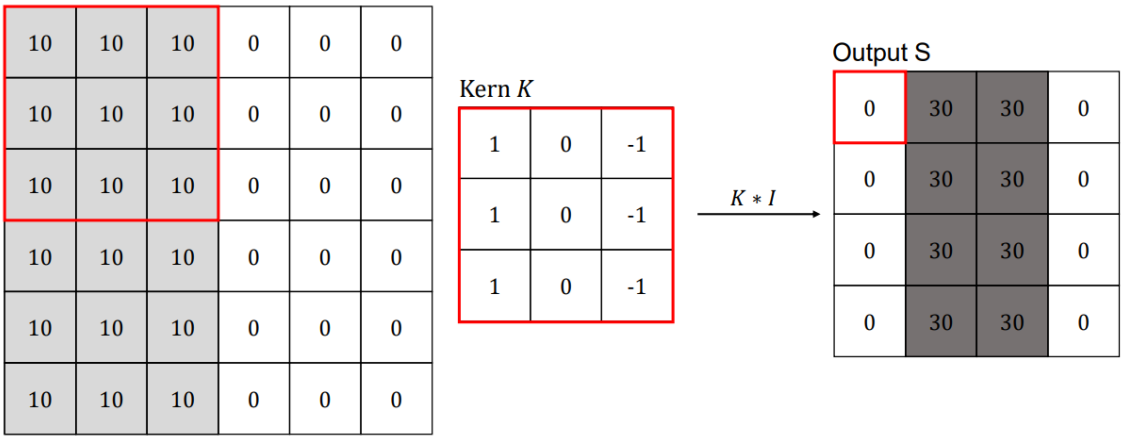


Abbildung 16 Faltung Rechenbeispiel [26]

$$S(1,1) = \sum_{m=-1}^{1} \sum_{n=-1}^{1} I(1+m, 1+n)K(m,n) \tag{7.2}$$
$$= 10 + 10 + 10 + 0 + 0 + 0 - 10 - 10 - 10 = 0$$

Wie in dem Beispiel zu sehen ist, konnte die Größe der Matrix verkleinert und gleichzeitig ein Merkmal der Eingangsmatrix verstärkt werden.

Die Gewichtungen des Kerns sind nicht vorgegeben und durch das Training des Kerns können neue und komplexe Features erlernt und identifiziert werden. Im Vergleich zu vollkommen konnektierten Schichten besitzt eine Faltungsschicht weniger Verbindungen und damit Gewichtungen, weshalb die Faltungsschicht performanter ist als eine vollends verbundene Schicht. Mittels Backpropagation mit Nebenbedingungen können die Werte der Matrix angepasst werden.

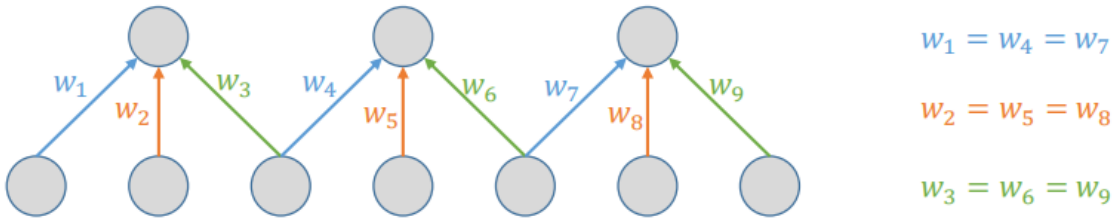


Abbildung 17 Gewichte mit Nebenbedingungen [26]

Neben dem vorgestellten Convolutional Layer besitzt ein CNN zumeist ebenfalls

- eine Subsampling- oder Pooling-Schicht, welche die Auflösung reduziert und somit die Abhängigkeit von präziser Feature-Positionierung in Merkmalskarten verringert. Da bei der Objekterkennung meist nicht die exakte Position eines Features benötigt wird, sondern eine ungefähre Lokalisierung ausreicht [27].
- eine vollverknüpfte Schicht (Fully-Conncected-Layer). Dies bedeutet: alle Neuronen einer Schicht sind mit den Neuronen der darauffolgenden Schicht verbunden. Eine solche Schicht dient zur Klassifikation der Eingaben.

Die Ausgänge der verschiedenen Ebenen werden Feature-Maps also Merkmalskarten genannt. Die Feature-Maps extrahieren pro Stufe immer wieder neue Merkmale aus den Eingabedaten oder verstärken die Vorherigen [24]. Der generelle Aufbau eines CNN könnte wie in Abbildung 18 aussehen.

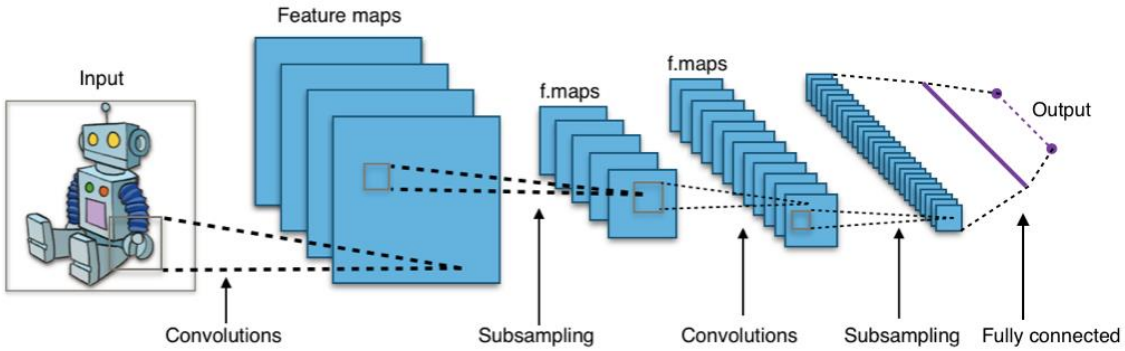


Abbildung 18 Aufbau CNN [28]

Der wesentliche Vorteil von CNN im Vergleich zu normalen KNN ist die höhere Effizienz. Die Rechenzeit zur Ermittlung der Gewichtungen genauso wie der Durchlauf im eingelernten Zustand sind im Vergleich zu einem konventionellen KNN verkürzt, da das CNN, durch die Struktur der Faltungsschichten sowie durch die Pooling-Schichten, weniger Gewichte benötigt. Dies schlägt sich auch auf den Speicherbedarf der Modelle nieder.

Aufgrund dieser Eigenschaften werden CNN in einer Vielzahl von verschiedenen Einsatzgebieten verwendet. In den letzten Jahren übertrafen verschiedene Modelle in Punkto Genauigkeit bei Klassifikationen bereits die menschliche Performance. CNNs werden jedoch nicht nur in der Bildverarbeitung, sondern unteranderem auch in der Audio- und Textverarbeitung eingesetzt.

7.3 Künstliche Neuronale Netze für heterogene Zeitsignale

7.3.1 Long Short Term Memory

Ein LSTM-Netz ist eine Art rekurrentes neuronales Netz. Wie in Kapitel 7.2.2 erläutert, ist ein RNN ein neuronales Netz, welches versucht, zeit- und sequenzabhängiges Verhalten zu modellieren.

LSTM baut auf dem RNN auf, fügt diesem jedoch eine Speicher-Komponente hinzu. Dieser Speicher soll dazu beitragen, die zum Zeitpunkt T erlernten Informationen für die zukünftigen Zeitpunkte T+1, T+2, ... zu speichern und bereitzustellen. Die Hauptidee dabei ist, die irrelevanten Teile früherer Zustände zu vergessen und die Zustände selektiv zu aktualisieren, um bestimmte relevante Teile des früheren Zustandes mit dem momentanen Zustand verarbeiten zu können. Da nun manche Zustände verworfen, manche Zustände aktualisiert und manche Teile eines Zustandes verrechnet werden, gibt es

bei LSTMs keine lange Backpropagation-Ketten mehr, was das verschwindende Gradienten-Problem löst und LSTMs effizienter zu trainieren macht als normale RNNs [29].

Heutzutage existieren variierende Architekturen eines LSTM-Moduls. Die grundlegende Funktionsweise eines solchen Neurons wird anhand einer Beispielarchitektur (in Abbildung 19 gezeigt) erläutert.

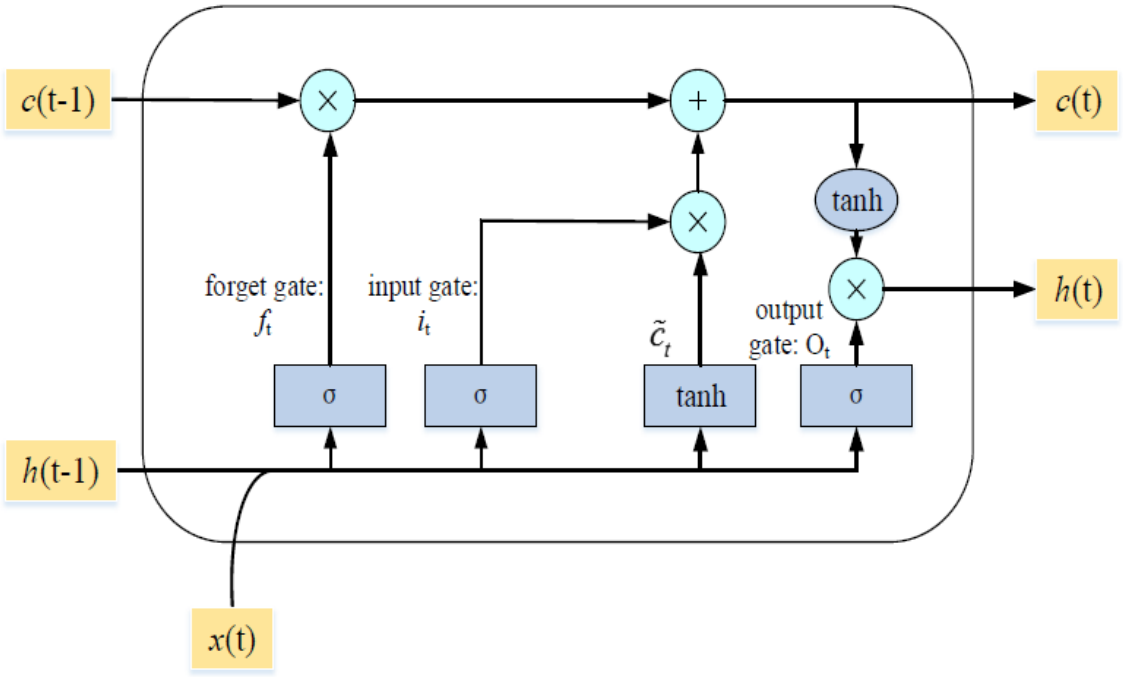


Abbildung 19 Long Short Term Memory Modul [30]

In der LSTM-Einheit finden sich drei wesentliche Strukturen. Das Vergesstor (forget gate), das Eingangstor (input gate) und das Ausgangstor (output gate).

Das Vergesstor ist der erste Teil der LSTM-Phase und entscheidet, wie viele Informationen aus dem früheren Zeitschritt in der Erinnerung bleiben oder vergessen werden sollen. Dies wird erreicht, indem der verborgene Zustand $h_{(t-1)}$ und die aktuelle Eingabe $x_{(t)}$ durch eine Aktivierungsfunktion geleitet werden.

Das Eingangsgatter hilft bei der Entscheidung, wie viele Informationen an die aktuelle Stufe weitergegeben werden sollen, indem es die Aktivierungsfunktion sowie die tanh-Funktion verwendet.

Das Ausgangsgatter steuert, wie viele Informationen vom verborgenen Zustand zurückgehalten und erst im nächsten Zeitschritt weitergeleitet werden. Auch hier wird der momentane Zustand durch die tanh-Funktion geleitet [29].

Die externen Eingänge des Neurons sind der vorherige Zellzustand $c_{(t-1)}$, der vorherige verborgene Zustand (auch hidden state genannt) $h_{(t-1)}$ und der momentane Eingangsvektor $x_{(t)}$. Aus der Abbildung sowie den Eingängen lassen sich die kompakten, mathematischen Beschreibungen der Tore herleiten.

$$f_{(t)} = \sigma(W_{fx}x_{(t)} + W_{fh}h_{(t-1)} + b_f) \quad (7.3)$$

$$i_{(t)} = \sigma(W_{ix}x_{(t)} + W_{ih}h_{(t-1)} + b_i) \quad (7.4)$$

$$O_{(t)} = \sigma(W_{ox}x_{(t)} + W_{oh}h_{(t-1)} + b_o) \quad (7.5)$$

Dabei ist

σ	nichtlineare Aktivierungsfunktion (z.B. Sigmoid Funktion)
w_{ax}	Gewichtung zwischen dem Eingangswert x, h und einem Gate

In dem LSTM-Modul wird ein intermediärer Zustand $C_{(\mathrm{t})}$ generiert.

$$C_{(t)} = tanh(W_{cx}x_{(t)} + W_{ch}h_{(t-1)} + b_c) \quad (7.6)$$

tanh repräsentiert die nichtlineare tanh Aktivierungsfunktion. Die Veränderung der Zelle $C_{(\mathrm{t})}$ und dem Ausgangszustand $h_{(\mathrm{t})}$ lassen sich mit folgender Gleichung beschreiben.

$$C_{(t)} = f_{(t)} \odot C_{(t)} + i_{(t)} \odot C_{(t)} \quad (7.7)$$

$$h_{(t)} = O_{(t)} \odot tanh(C_{(t)}) \quad (7.8)$$

$\odot$ wird verwendet, um eine elementweisen Multiplikationsoperation für zwei Vektoren zu bezeichnen [30].

Das vorgestellte LSTM-Modul kann die verdeckten Neuronen eines neuronalen Netzes ersetzen und dadurch weitere Funktionalität zu dem vorhandenen Netz bringen.

7.3.2 Gleitendes Fenster Verfahren

Unter dem gleitenden Fenster Verfahren (engl. Sliding-Window) versteht man ein Verfahren, bei welchem dem zu trainierenden neuronalen Netz nicht nur der aktuell verfügbare Wert, sondern ebenfalls Vergangenheitswerte zugeführt werden.

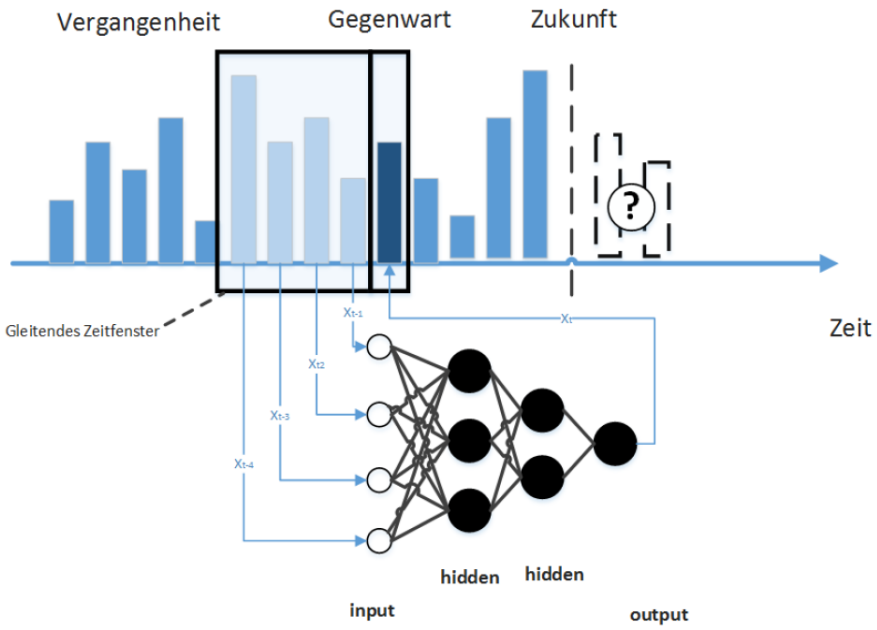


Abbildung 20 Gleitendes Fenster Verfahren [24]

In der Trainingsphase werden dem Modell Werte innerhalb des gleitenden Fensters der Breite n zur Verfügung gestellt. Anhand dieser Werte wird der Ausgabewert berechnet und mit dem nachfolgenden Wert des Fensters abgeglichen. Durch diesen Fehler optimiert sich das Netz während des Trainings. Das Ziel des trainierten Netzes ist es, die Entwicklung der Zeitreihe vorherzusagen. In der realen Anwendung werden dem Neuronalen Netz der aktuelle Wert sowie n-1 Vergangenheitswerte übergeben. Die Ausgabe des Netzes entspricht dann einer Prognose für das zu analysierende System. In Abbildung 20 wird ein FNN mit dem gleitenden Fenster-Verfahren gezeigt, allerdings beschränkt sich diese Methodik nicht auf einen spezifischen Architekturtyp.

7.4 Trainingsverfahren

7.4.1 Überwachtes Lernen

7.4.1.1 Korrigierendes Lernen

Bei dem korrigierenden Lernen werden dem neuronalen Netz Daten mit dazugehörigen Labels zur Verfügung gestellt. Jeder Eingabevektor benötigt einen dazugehörigen Ausgabevektor. Anhand dieser Vektoren passt das Model im Lernprozess die Gewichtungen seiner Bestandteile an und optimiert damit eine Art der Kostenfunktion, welche auf einem Vorhersagefehler beruht. Man nennt diesen Lernvorgang überwacht, da für jede Eingabe die richtige Ausgabe bekannt ist und man das Modell bei falschen Vorhersagen korrigieren kann [31, S. 7]. Nachteil dieses Verfahren ist die Notwendigkeit einer großen Menge von bereits bearbeiteten Daten.

7.4.1.2 Delta-Regel

Die Delta-Regel oder auch Widrow-Hoff-Regel ist ein Verfahren, welches zur Gewichtsadaption von neuronalen Netzen im überwachten Lernen verwendet wird. Dabei ist die Gewichtsänderung proportional zur Differenz der aktuellen Ausgabe eines Neurons und der bereits bekannten, gewünschten Ausgabe. Dieses Verfahren ist jedoch auf vorwärts gerichtete Netzwerke mit nur einer trainierbaren Schicht und lineare Funktionen begrenzt [32]. Dies bedeutet, das zu trainierende Netz darf keine verdeckte Schicht beinhalten.

Die Änderung des Gewichtes zwischen einem Neuron i in der Eingabeschicht und einem Neuron j der Ausgabeschicht wird in Quelle [32] aufgezeigt und kann mit folgender Formel beschrieben werden:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot o_i \tag{7.9}$$

o_{i} ist das Aktivierungslevel des Eingabeneurons. δ_j stellt die Differenz des Aktivierungslevels des Ausgabeneurons j zu dem gewünschten Wert dar. Dies kann mit folgender Formel dargestellt werden:

$$\delta_j = t_j - o_j \tag{7.10}$$

Wobei t_{j} dem gewünschten Ausgabewert entspricht. η wird als Lernparameter bezeichnet, da er die Rate angibt, mit welcher das System die Gewichtungen anpasst. In Gleichung 7.1 ist zu erkennen, dass eine Lernrate von Eins einem direkten Sprung auf den gewünschten Eingangswert entsprechen würde. In der Praxis ist dies jedoch meist nicht gewünscht, da ein Herantasten an den optimalen Wert über alle Eingaben zu einem besseren Verlustwert führt. Der Lernparameter wird vor Beginn der Lernphase festgelegt und sollte eine Größe zwischen Null und Eins repräsentieren.

Nach jedem Durchlauf wird der Wert der Gewichtung durch die eben beschriebene Formel angepasst, bis ein Optimum erreicht ist. Je kleiner δ im Verlauf der Trainingsphase wird, desto näher kommt das neuronale Netz seinem gewünschten Endergebnis [33].

Im Folgenden wird ein weiterer allgemeiner Angang zur Berechnung der Gewichtsänderung auf Basis des zuvor gezeigten Approximationsfehlers aufgeführt. Der quadratische Fehler wird häufig als ein Maß für den Fehler verwendet und wird in diesem Fall wie folgt berechnet:

$$E = \sum_p E_p \ mit \ E_p = \frac{1}{2} \cdot (t_j - o_j)^2 = \frac{1}{2} \cdot e_p^2 \tag{7.11}$$

E_p entspricht dem quadratischen Fehler eines spezifischen Trainingsmusters. Folglich entspricht E dem Gesamtfehler über alle trainierten Muster. Das Ziel des Algorithmus ist es, die Gewichtsänderungen so zu verändern, dass sich der quadratische Fehler für das aktuelle Muster p und den aktuellen Input o_i verringert. Hierbei wird der Gradient, also der Anstieg der Fehlerfunktion, an der Stelle des Gewichts w verwendet. Das Gewicht wird dann in Richtung des negativen Anstiegs verändert. Aufgrund der Verwendung des negierten Gradienten wird die Delta-Regel zu den Gradientenabstiegsverfahren gezählt. Zusammen mit der Lernrate η ergibt sich folgende verallgemeinerte Form der Delta-Regel [34]:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E_p}{\partial w_{ij}} = -\eta \cdot \nabla E_p \tag{7.12}$$

Die Erreichung des globalen Miniums der Fehlerfunktion entspricht der optimalen Gewichtsanpassung.

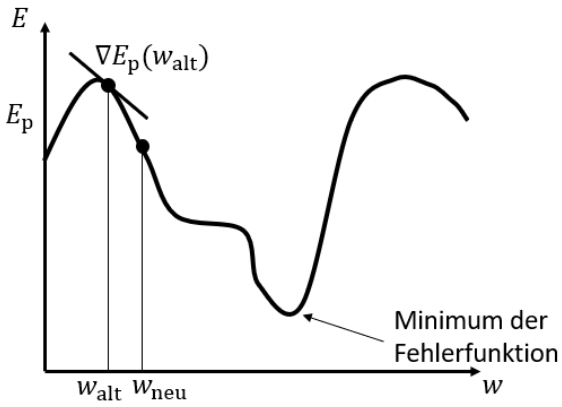


Abbildung 21 Fehlerfunktion mit Minimum [32]

7.4.1.3 Backpropagation

Der Backpropagation Algorithmus basiert auf der zuvor beschriebenen Delta-Regel. Im Gegensatz zu dieser ist er allerdings in der Lage, auch mehrschichtige neuronale Netze zu trainieren. Der Algorithmus durchläuft zunächst vorwärts die Schichten. Dabei werden die Eingabewerte von Schicht zu Schicht weitergegeben und entsprechend der Gewichtungen der Verbindungen und den Aktivierungsfunktionen modifiziert.

Im zweiten Durchlauf werden die Schichten rückwärts durchlaufen. Hierbei werden die Fehlerwerte von der Ausgangsschicht über die verdeckten Schichten zu der Eingangsschicht weitergeleitet und die Gewichtungen korrigiert. Bei der Berechnung der Fehlerwerte muss zwischen den Neuronen der verdeckten Schicht und der Eingangsschicht unterschieden werden. Da den Neuronen in der verdeckten Schicht keine gewünschten

Ausgabewerte zugewiesen werden können, ist auch die Gleichung der Delta-Regel nicht mehr anwendbar [33].

Die zuvor aufgrund der Verständlichkeit nicht weiter definierte Ausgabe o_j kann wie folgt beschrieben werden:

$$o_j = \varphi(net_j)\ mit\ net_j = \sum_{i=1}^{n} x_i \cdot w_{ij} \tag{7.13}$$

Dabei ist

φ	differenzierbare Aktivierungsfunktion
net_j	Netzeingabe
x_i	Eingabe i
n	Anzahl der Eingaben
w_ij	Gewichtung zwischen Neuron i und j

Da auf net_j bereits in Kapitel Künstliche Neuronen eingegangen wurde, wird das Verständnis der Netzeingabe hier vorausgesetzt.

Aufbauend auf den Gleichungen und kann die Gewichtsänderung Δw_{ij} der Verbindung zwischen den Neuronen i und j wie folgt beschrieben werden

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E_p}{\partial w_{ij}} = -\eta \cdot \delta_j \cdot o_i \tag{7.14}$$

Aufgrund der existenten verdeckten Schicht muss eine Unterscheidung bei dem Fehlersignal des Neurons j vorgenommen werden. Eine Abgrenzung findet statt, wenn das Neuron nicht wie bisher ein Ausgabeneuron ist, sondern ein verdecktes Neuron. In diesem Fall ist der Fehler des Neurons abhängig von den nachfolgenden Neuronen k und deren Verbindungen zu dem Neuron j. Aus diesem Grund müssen die Fehler mit den dazugehörigen Gewichtungen aufsummiert werden ($\sum_k \delta_k \cdot w_{jk}$). Durch die Abhängigkeit der Gewichtsänderung von der Ableitung der Aktivierungsfunktion, muss die Ableitung dieser dem Term gemeinsam mit der Netzeingabe hinzugefügt werden $\varphi'(net_j)$.

$$\delta_j = \begin{cases} \varphi'(net_j)(t_j - o_j) & Ausgabeeuron\ j \\ \varphi'(net_j) \sum_k \delta_k \cdot w_{jk} & verdecktes\ Neuron\ j \end{cases} \tag{7.15}$$

Mithilfe dieser Formeln kann die Änderung des Gewichtes dann folgendermaßen vollzogen werden:

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij} \tag{7.16}$$

7.4.1.4 Problematiken bei Gradientenabstiegsverfahren

Der Backpropagation Algorithmus sowie Gradientenabstiegsverfahren im Allgemeinen besitzen einige Schwierigkeiten. Werden alle Gewichtungen mit demselben Wert initialisiert, entsteht eine Symmetrie. Unter „Symmetrie Breaking“ versteht man daher das Phänomen, dass bei einer Gewichtsänderung alle Gewichte dieselbe Änderung erfahren und dabei nicht mehr aus der Symmetrie ausgebrochen werden kann. Ebenso sollten keine zu großen Gewichte gewählt werden, da diese das Lernverfahren ungewollt in die Länge ziehen. Aus diesem Grund werden Startgewichte meist per Zufall zugewiesen, wobei sich der Wert zwischen $[-\frac{2}{N_i}..\frac{2}{N_i},\ N_i = Anzahl\ Eingangneuronen]$ befinden sollte [35].

Eine weitere Schwierigkeit des Algorithmus besteht darin, dass er in einem lokalen Minimum verharrt und nicht in der Lage ist, das globale Minimum zu finden (Abbildung 23). Dies ist umso wahrscheinlicher, je komplexer die Funktion wird, also umso mehr Verbindungen und Neuronen das Netz besitzt. Zur Veranschaulichung ist in Abbildung 22 eine mehrdimensionale Fehlernfunktion dargestellt. An ihr ist zu erkennen, dass eine Funktion mehrere lokale Minima besitzen kann. Dieser Fakt verstärkt sich mit der Anzahl der Dimensionen. In der Abbildung wurden nur 2 Gewichtungen, in der x- und y-Achse, dargestellt und der Fehler in der z-Achse.

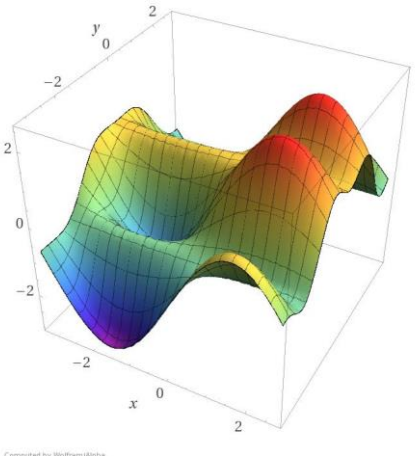


Abbildung 22 Fehlerfunktion [36]

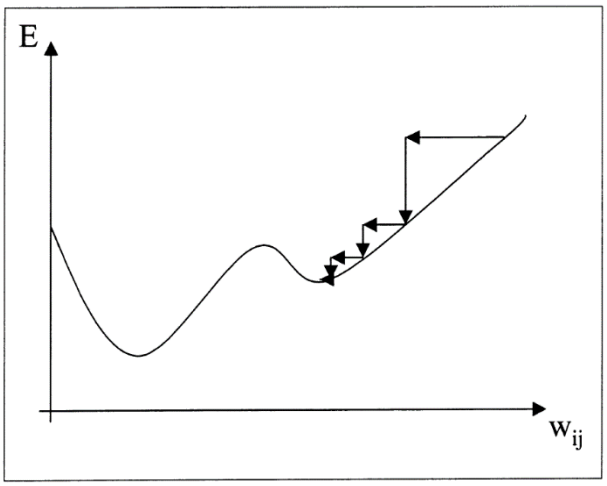


Abbildung 23 Verharren in lokalem Minimum [35]

Eine weitere Schwierigkeit besteht in Plateaus. Abschnitte, in denen die Änderung gegen Null geht, stellen ein Risiko für das Erreichen eines Minimums dar. Wie in den vorangegangenen Kapiteln bereits beschrieben wurde, hängt die Änderung des Gewichtes von dem Gradienten der Funktion ab. Erreicht die Ableitung der Funktion nun einen sehr kleinen Wert, ist die Änderung des Gewichtes dementsprechend gering. Aus diesem Grund braucht der Algorithmus eine Vielzahl an Iterationen, um einen kleinen Teilbereich des Plateaus zu überwinden oder er stagniert bei keiner Änderung sogar komplett [37]. Dies wird in der folgenden Abbildung verdeutlicht:

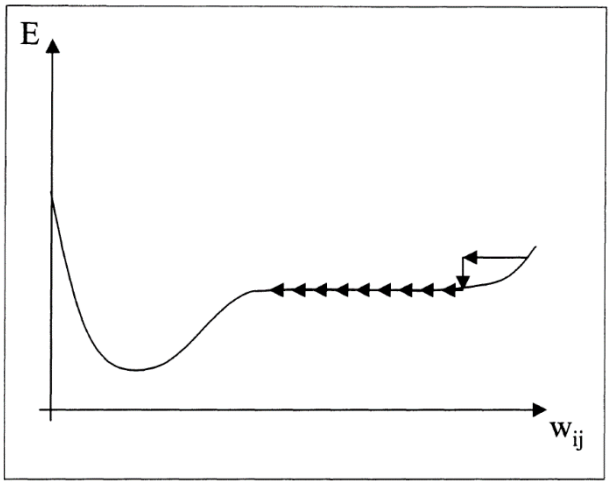


Abbildung 24 Flaches Plateau einer Fehlerfunktion [35]

Treten steile und enge Täler in der Funktion auf, ist es möglich, dass das Tal übersprungen wird, sollte die Lernrate zu groß für diesen Fall sein. Ist die Ableitung nahe dem gewünschten Tiefpunkt betragsmäßig sehr groß, kann der Algorithmus direkt über die nächste Anhöhe springen und damit das Minimum verlassen [35, S. 54].

All diese Schwierigkeiten können durch verschiedene Modifikationen verhindert oder abgeschwächt werden. Ein Beispiel für eine solche Behebung findet sich im nächsten Kapitel.

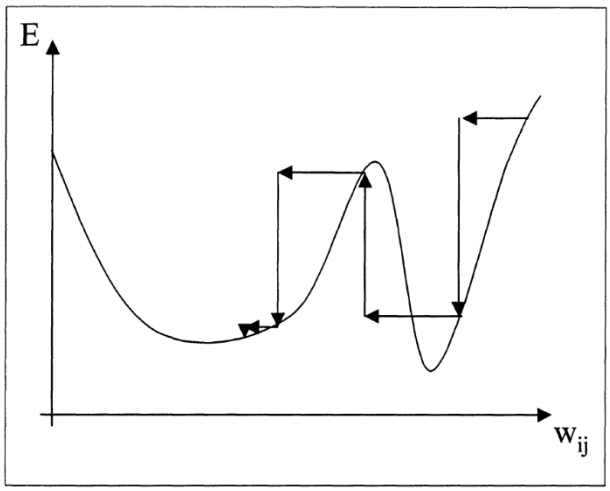


Abbildung 25 Verlassen guter Minima [35]

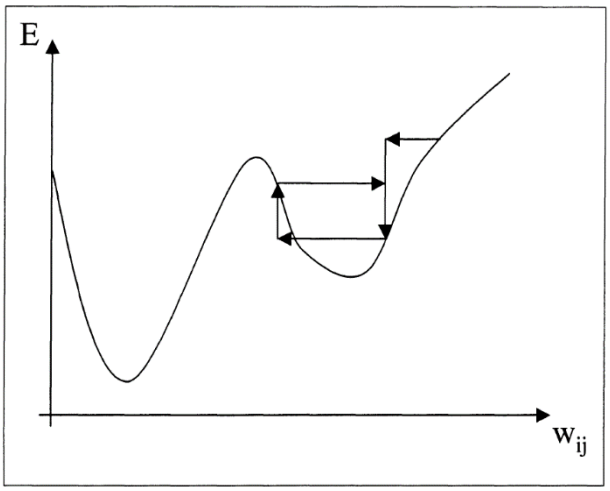


Abbildung 26 Oszillation in steilen Tälern [35]

7.4.1.5 Resilient Propagation

Eine weitere Problematik besteht in einer Oszillation des Algorithmus über einem Minimum. Aufgrund der eingestellten Lernrate wird der Tiefpunkt nicht erreicht, sondern der

Algorithmus springt über diesem hin und her (Abbildung 26). Dies lässt sich jedoch durch die Verwendung von einer resilienten Propagation (engl. resilient propagation) verhindern.

Hierbei werden die Änderungen im vorherigen Iterationsschritt mit einbezogen, um eine dynamische Gewichtsänderung zu schaffen. Bleibt das Vorzeichen des Gradienten im Vergleich zu dem vorangegangenen Schritt gleich, wird die Lernrate erhöht, um schneller zu dem gewünschten Optimum zu gelangen. Wird eine Änderung des Vorzeichens festgestellt, wird die Lernrate verringert, da soeben ein Minimum übersprungen wurde [23].

7.4.1.6 Verstärkendes Lernen

Verstärkendes Lernen (im englischen reinforcement learning) ist eine Herangehensweise, welche es einem intelligenten Programm, oft Agent genannt, erlaubt, sowohl mit bekannten als auch mit unbekannten Umgebungen zu interagieren und seine Handlungen bezüglich der Umgebungsregeln zu optimieren. Der Lernvorgang wird dabei durch Belohnungen und Bestrafungen gesteuert. Dem Agenten wird nicht aufgezeigt, welche Aktion in welcher Situation die besten Resultate liefert. Anstatt dessen erhält der Agent zu bestimmten Zeitpunkten eine positive Rückmeldung (Belohnung), zum Beispiel für das Erreichen eines Ziels, oder eine negative Rückmeldung (Bestrafung), für das Ausscheiden/Verlieren in dem zu erlernenden Spiel [38, S. 1] . Der Agent approximiert eine Nutzenfunktion, welche bestimmten Aktionen oder Zuständen Werte zuordnet. Durch die Bestrebung die Punktzahl zu maximieren, lernt der Agent den vorgegebenen Regeln der Umgebung zu folgen [39].

Reinforcement Learning wird vor allem in Bereichen eingesetzt, in denen eine Umgebung simulativ erzeugt werden und der Agent eine Vielzahl an Durchläufen absolvieren kann. Das wahrscheinlich prominenteste Beispiel für eine solche Herangehensweise ist das Programm AlphaGo, welches von DeepMind entwickelt wurde. Das Computerprogramm war 2015 in der Lage, den mehrfachen Go-Europameister Fan Hui zu besiegen, in einem Spiel, welches als am schwierigsten, für eine Maschine, zu erlernende Brettspiel gilt.

7.4.2 Unüberwachtes Lernen

Wie der Name bereits vermuten lässt, werden bei dem unüberwachten Lernen dem Algorithmus keine gelabelten Daten oder Umgebungsregeln zur Verfügung gestellt. Meist ist es dabei der Fall, dass der Nutzer selbst noch keine genaue Zuordnung der Daten getroffen hat. Die Aufgabe des Algorithmus ist es daher, selbstständig eine Zuordnung zu treffen und ein Muster in den bereitgestellten Daten zu finden. Zum Beispiel können Artikel zugeordnet werden, welche oft zusammen gekauft werden. Aber auch die Einordnung in verschiedene Cluster ist eine häufige Aufgabe des Algorithmus. Die Überprüfung der Ergebnisse findet dabei erst nach Beendigung der Trainingsphase statt und der

Nutzer muss selbst validieren, ob die vorliegenden Gruppierungen mit bekannten Kategorien übereinstimmen [40].

7.5 Verlustfunktionen

Verlustfunktionen (manchmal auch Zielfunktionen oder Errorfunktionen genannt) werden als ein Maß für die Qualität der Ausgaben eines neuronalen Netzes verwendet. Häufig wird eine der folgenden drei Funktionen dafür verwendet.

Die **binäre Kreuzentropie** (engl. Binary Cross entropy) wird meist bei binären Klassifizierungsproblemen eingesetzt. Präziser ausgedrückt sollte diese Verlustfunktion eingesetzt werden, wenn das entworfene Netz darauf ausgelegt ist, die Wahrscheinlichkeiten der Ergebnisse vorherzusagen. In diesem Fall hat die Ausgabeschicht ein Neuron mit einer Sigmoid-Funktion als Aktivierungsfunktion. Die binäre Kreuzentropie kann durch folgenden Ausdruck beschrieben werden [23, 41]:

$$-(y_i \cdot log\, f(x_i, \theta) + (1 - y_i) log\, (1 - f(x_i, \theta))) \tag{7.17}$$

Dabei ist

$\log$	Natürlicher Logarithmus
$\mathrm{f(x_i, \theta)}$	Ausgabe des Neurons für Klasse i → Wahrscheinlichkeit der Klassenzugehörigkeit zur Klasse i
y_i	Binärer Indikator, ob die Klasse korrekt klassifiziert wurde

Handelt es sich bei dem zu lösenden Problem um eine Mulitklassen-Klassifizierung, also ist die Anzahl der Klassen größer als 2 (n > 2), dann wird die Funktion der Kreuzentropie verwendet. In diesem Fall wird meist Softmax als Aktivierungsfunktion für jede mögliche Klasse in der Ausgabeschicht eingesetzt [23]. Dargestellt werden kann die **Kreuzentropie** als

$$-\sum_{i=1}^{n} y_i \cdot log\, f(x_i, \theta)) \tag{7.18}$$

Ist das Netz dafür entwickelt worden ein Regressionsproblem zu lösen, wird meist die **mittlere quadratische Abweichung** (engl. mean squared error) als Verlustfunktion verwendet.

$$\frac{1}{n}\sum_{i=1}^{n} (\hat{y} - y_i) \tag{7.19}$$

7.6 Optimierer

In diesem Kapitel werden einige der Optimierer genauer betrachtet, um ihren Einsatz bei verschiedenen Netzarchitekturen zu bestimmen und um einen Einblick in den Aufbau von Optimierern zu gewähren. Die im Folgenden behandelten Optimierer stellen nicht die Gesamtheit aller verfügbaren Optimierer in Keras dar. Einen ausführlicheren Einblick in die am häufigsten verwendeten Optimierer bietet Quelle [42].

7.6.1 SGD

Stochastischer Gradientenabfall (engl. Stochastic Gradient Descent) führt eine Gewichtsänderung bei jedem Trainingsbeispiel i durch. Aus diesem Grund ist SGD schneller als beispielsweise Batch Gradient Descent, bei dem nur eine Aktualisierung für das gesamte Datenset vorgenommen wird. Durch die häufigen Aktualisierungen mit einer hohen Varianz schwankt die Zielfunktion (meist auch Kosten- oder Errorfunktion genannt) (siehe Abbildung 27).

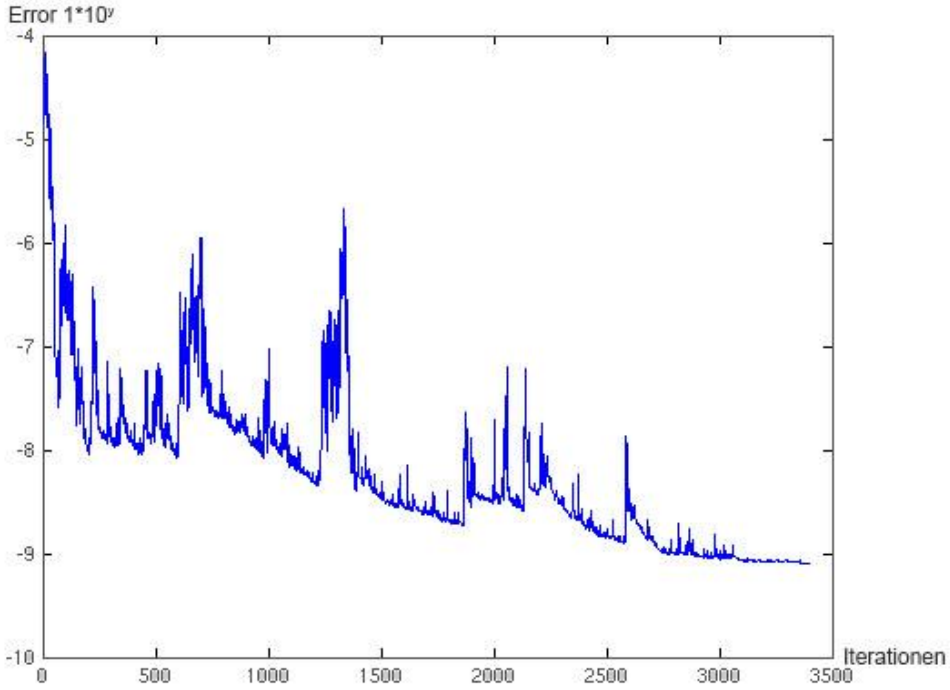


Abbildung 27 SGD Fluktuation [43]

Diese Fluktuation erlaubt es SGD in neue, potenziell bessere, Minima zu springen. Andererseits erschwert dieses Verhalten das Einpendeln auf ein Minimum, da ein solches öfters übersprungen wird. Jedoch kann dieses Verhalten durch eine langsame Verringerung der Lernrate verhindert werden [42, 44]. Das Verhalten des stochastischen Gradientenabfalls lässt sich mit der Formel der Backpropagation, welche in Kapitel 7.4.1.3 bereits erklärt wurde, beschreiben. In der Literatur wird diese meist wie folgt dargestellt:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{7.20}$$

Dabei ist

θ	Gewichtung einer Verbindung
η	Lernrate
∇	Gradient der Funktion J
J	Ziel-/ Errorfunktion
$x^{(i)}$	Eingabewerte für Trainingsbeispiel i
$y^{(i)}$	Erwünschte Ausgabe für Trainingsbeispiel i

Aufgrund der Einfachheit dieses Optimierers ist er eine gute Wahl für flache Netze. Er konvergiert aber auch deutlich langsamer als modernere, weiterentwickelte Algorithmen [45].

7.6.2 Adam

Adaptive Momentenschätzung (engl. Adaptive Moment Estimation, kurz Adam) ist eine Methode, welche eine adaptive Lernrate für jeden Parameter berechnet. Bei anderen Optimierern, wie Adadelta und RMSprop, werden die exponentiell abnehmenden Durchschnitte der vergangenen quadratischen Gradienten v_t gespeichert. Zusätzlich zu dieser Information speichert Adam auch den exponentiell abnehmenden Durchschnitt der vergangenen Gradienten.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{7.21}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7.22}$$

Dabei ist

m_t	Schätzung des ersten Moments (Mittelwert)
v_t	Schätzung des zweiten Moments (Varianz)
g_t	Gradient der Zielfunktion
β_1	Verfallsparameter (Standardwert: 0.9)
β_2	Verfallsparameter (Standardwert: 0.999)

Zu Beginn werden die Startvektoren mit null initialisiert. Die Autoren von Adam stellten dabei während der Entwicklung fest, dass es eine Verzerrung in der Nähe von null gab. Dies trat besonders in den Anfangsschritten und wenn die Verfallsraten sehr gering sind (also, wenn β_1, β_2 nahe an eins liegen) auf. Um dies zu beheben wurden die Momente wie folgt angepasst.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{7.23}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{7.24}$$

Die Aktualisierung kann damit folgendermaßen beschrieben werden.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{7.25}$$

Dabei ist

θ	Gewichtung
η	Lernrate (Standardwert: 0.001)
ϵ	Kleine Zahl, um Division durch 0 zu verhindern (Standardwert: 10^{-8})

Adam performt im Vergleich zu anderen adaptiven Lernraten Optimierern durchschnittlich am besten und wird deshalb auch oft in der Praxis eingesetzt [42, 44].

7.7 Dropout

Dropout ist eine Regulierungs-Methode. Wird diese Methodik auf eine Schicht angewendet, werden zufallsbasiert ausgewählte Neuronen und ihre Verbindungen ignoriert, also fallen aus dem Netz heraus (im Englischen sagt man dazu „dropped-out“) [46]. Durch die Herausnahme einzelner Neuronen lassen sich Co-Adaptionen von Neuronen verhindern. Dies bedeutet Neuronen passen ihre Gewichtungen unabhängig von ihrem Nachbarn an, da dieser nicht zu jedem Zeitpunkt existiert. Durch diese Maßnahme reduziert signifikant das Overfitting [47]. Unter Overfitting versteht man die Überanpassung des Netzwerkes. Diese performt zunehmend besser auf den Trainingsdaten, verliert aber die Generalisierung und hat somit eine schlechtere Genauigkeit auf den Testdaten. In Quelle [47] wird gezeigt, dass Dropout die Performance von neuronalen Netzen, bei dem überwachten Lernen in verschiedenen Aufgabenbereichen und mit verschiedenen Architekturen, erhöht. Für die Verwendung von Dropout wird generell ein Wert zwischen 0,2 (20%) und 0,5 (50%) empfohlen. Insbesondere bei großen Netzen mit vielen Verbindungen bietet Dropout eine bessere Performance. Dropout kann sowohl auf einzelne Schichten als auch auf die Eingangsschicht und alle verdeckten Schichten angewendet werden [46].

7.8 Aktivierungsfunktionen

Die Aktivierungsfunktion stellt den Zusammenhang zwischen dem Netzinput und dem Aktivitätslevel eines Neurons dar. Visualisiert wird diese meist in einem zwei-dimensionalen Diagramm. In diesem werden auf der X-Achse der Netzinput der Einheit und auf

der Y-Achse das Aktivitätslevel aufgetragen [48]. In der Theorie können nicht-lineare Aktivierungsfunktionen mit einem mindestens zweischichtigen KNN jede Funktion approximieren. Aus diesem Grund werden oft nicht-lineare Funktionen in der Praxis verwendet. Des Weiteren sollte die Aktivierungsfunktion kontinuierlich differenzierbar sein, um Gradientenabstiegsverfahren zur Anpassung der Gewichte nutzen zu können. Funktionen, deren Ausgabe endlich sind, führen bei gradientenbasierten Methoden zu einer stabileren Leistung [23].

7.8.1 Lineare Funktion

Die einfachste Aktivierungsfunktion ist eine lineare Funktion $\boldsymbol{f(x) = x}$**.** Die lineare Aktivierungsfunktion wird meist verwendet, um den Mittelwert einer Gaußschen Verteilung zu erzeugen [23].

7.8.2 Binäre Schwellenwertfunktion

Die binäre Schwellenwertfunktion kann nur zwei Zustände einnehmen. Meist ist dies 0 und 1. In manchen Fällen auch -1 und 1.

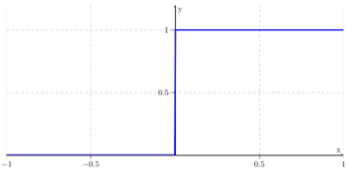


Abbildung 28 Schwellenwertfunktion [49]

7.8.3 Sigmoid

Die Sigmoid Funktion wird vor allem verwendet, weil sich Ihre Werte nur zwischen null und eins bewegen. Da diese Funktion aber zum verschwindenden Gradientenproblem beiträgt, wird sie nur selten in der verdeckten Schicht eines Netzes eingesetzt [29]. Mathematisch lässt sich die Funktion folgendermaßen beschreiben:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7.26}$$

7.8.4 Tanh

Die Tangens hyperbolicus Funktion $f(x) = \tanh(x)$ (siehe Abbildung 29) ist eine häufig genutzte Aktivierungsfunktion und wird beispielsweise in LSTM-Einheiten verwendet.

7.8.5 ReLU

Die gleichgerichtete Lineareinheit (Rectified Linear Unit) und ihre zugrundeliegende Aktivierungsfunktion $\boldsymbol{f(x) = \max(0, x)}$ (siehe Abbildung 29) werden üblicherweise in verdeckten Schichten verwendet. Berichte zeigen, dass ReLU zu großen und konsistenten Gradienten führen, was das gradientenbasierte Lernen fördert [23].

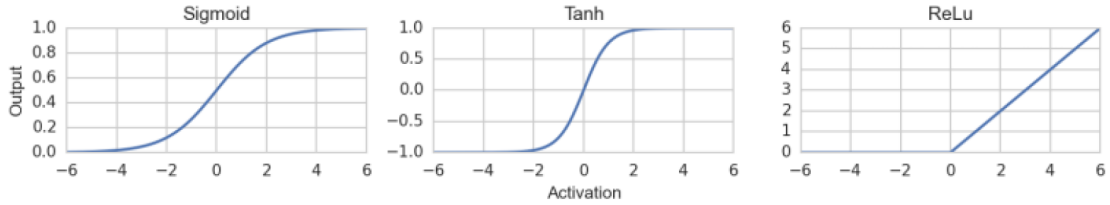


Abbildung 29 Aktivierungsfunktionen [50]

7.8.6 Softmax

Die Softmax-Aktivierungsfunktion beziehungsweise eine Softmax-Schicht, also eine Schicht, in der alle Aktivierungsfunktionen Softmax-Funktionen sind, werden normalweise bei Multiklassen-Klassifikationen zusammen mit der Cross-Entropy-Verlustfunktion eingesetzt. Die Softmax-Schicht normalisiert die Ausgaben der vorangegangenen Schicht, sodass diese aufsummiert Eins ergeben. Bei einer Schicht mit n-Neuronen kann die Ausgabe eines Neurons a durch die Funktion

$$f(x) = \frac{e^{x_a}}{\sum_{c=1}^{n} e^{x_c}} \tag{7.27}$$

beschrieben werden.

7.9 Auswahl

Betrachtet man die verschiedenen Architekturen, welche potenziell zur Verfügung stehen, gibt es allein von der Umsetzbarkeit mehrere Möglichkeiten. Sowohl die Feedforward Netze als auch die faltenden Netze sind in der Lage, eine Klassifikation vorzunehmen und sogar die rekurrenten Netze sind unter bestimmten Bedingungen dazu in der Lage. Die Wahl fiel daher auf das performanteste Netz, das Feedforward Netz. Vergleicht man dieses mit einem faltenden Netz, so besitzt es nur einen Bruchteil der Neuronen und die erzeugten Daten können direkt in das Netz eingespeist werden, ohne dass die ebenfalls generierten Bilddaten verwendet werden müssen.

Ebenfalls wurde eine gleitendes Fenster Verfahren ohne Rückkopplung ausgewählt, um dem Netz eine größere Anzahl an Werten für die Klassifizierung zur Verfügung zu stellen. Dem Netz wird dadurch nicht nur der aktuelle Wert als Eingabe übergeben, sondern auch X zuvor aufgenommene Werte. Da die Daten auf einen festen Zeitbereich normiert wurden, bietet es sich an, als Eingabe eine gesamte Periode zu betrachten.

Bei den Optimierern gibt es allein durch deren Technologie beziehungsweise Aufbau keinen sofort und klar ersichtbaren Favoriten. Weshalb hier die am meist verwendeten Optimierer getestet werden: SGD und Adam.

Aktivierungsfunktionen der Ausgabeschicht treten in der Praxis meist in Kombination mit einer bestimmten Verlustfunktion auf. So ist bei der Klassifikation das Paar aus einer Softmax-Aktivierungsfunktion und einem Kreuzentropie-Error Stand der Technik. Aber auch eine Kombination aus einer ReLU-Aktivierungsfunktion und der mittleren quadratischen Abweichung sind eine Kombination, die Möglichkeiten in sich birgt.

Ein Dropout auf einer verdeckten Schicht sollte genauso geprüft werden wie eine unterschiedliche Anzahl an verdeckten Schichten und Neuronen.

8 Praktische Umsetzung neuronaler Netze

8.1 Training der verschiedenen Netze

8.1.1 Notwendige Bibliotheken und empfohlene Versionen

Verwaltet wurden alle Bibliotheks-Versionen mithilfe von Anaconda. Mit dieser Open-Source-Distribution lassen sich unter anderem einfach Bibliotheken installieren, verwalten und aktualisieren.

Bibliothek	Version
argparse	1.4.0
pickle	4.0
numpy	1.17.0
wandb	0.9.5
keras	2.3.1
sklearn	0.23.1

Wandb

Weights and Biases ist eine Plattform und ein Software-Tool, mit welchem die Nutzer verschiedene Läufe speichern, bearbeiten und darstellen können. Unter anderem werden Hyperparametersuchen und Modeloptimierungen angeboten, genauso wie Datenset Versionierungen und Model Management. Genauere Informationen finden sich unter https://www.wandb.com/.

8.1.2 Programmierung

Im Folgenden wird ein Programm erläutert, mit dem es möglich ist, ein neuronales Netz zu erstellen und zu trainieren. Dabei werden die Codesegmente entlang des Programmablaufs erläutert. Hierbei beziehen sich die Erklärungen und Ausschnitte auf die train_nn_v3_softmax_sweep_one_layer_size_change.py-Datei, da in dieser alle notwendigen Schritte begutachtet werden können. Diese Datei wurde erzeugt, um durch einen Sweep, also die Veränderung der übergebenen Argumente beim Aufruf des Programms, die minimale Anzahl an notwendigen Trainingsdaten zu bestimmen zu können.

Um bei dem Aufruf des Programmes die Möglichkeit zu schaffen, Argumente übergeben zu können, müssen diese zunächst einmal definiert werden:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    # args
    #-----------------------------
    parser.add_argument(
        "-m",
        "--model_name",
```

```
        type=str,
        default="v3_softmax_sweep_onelayer_size_change",
        help="Name of this model/run (model will be saved to this file)")

    args = parser.parse_args()
```

Nachdem der Argument-Parser aufgerufen wurde, können die verschiedenen Argumente initialisiert werden. Beispielhaft wird der Modelname des Netzes verwendet. Alle anderen Argumente folgen jedoch derselben Struktur. Festgelegt werden die Abkürzung und der Name, mit welchem das Argument identifiziert werden kann („-m“, „—model_name“). Ebenfalls muss der Typ des übergebenen Parameters definiert werden, in diesem Fall handelt es sich dabei um einen string. Wird kein Wert übergeben, findet sich unter default der Standardwert, welcher verwendet wird. Unter help sollte eine kurze Beschreibung des Arguments aufgeführt werden. Diese kann dann von dem Nutzer im Terminal abgefragt werden.

Sind alle Argumente definiert und eingelesen worden, werden diese in einer Variablen gespeichert.

```
    args = parser.parse_args()
```

Darauffolgend wird die Funktion run_experiment aufgerufen und die Variable übergeben.

```
    run_experiment(args)
```

In der run_experiment-Funktion läuft der Hauptteil des Programmes ab. Zunächst wird wandb initialisiert und ein Projekt erstellt, sollte noch keines vorhanden sein.

```
def run_experiment(args):
    """ Build model and data generators; run training"""
    wandb.init(project="KNN_v3_softmax_sweep_1layer_size_change")
```

Die build_model-Funktion erzeugt das Model des neuronalen Netzes. Hierzu müssen die Parameter für dieses übergeben werden. Die Parameterwerte sind entweder die Eingaben des Nutzers beim Aufruf des Skriptes oder die zuvor festgelegten Standardwerte.

```
model = build_model(args.dense_layer1, args.dense_layer2, args.optimi-
zer, args.dropout, args.num_classes, args.learning_rate)
```

Im Inneren der build_model-Funktion werden mithilfe von Keras die verschiedenen Schichten dem Model hinzugefügt. In diesem Fall wird eine verdeckte voll konnektierte Schicht mit dls1 Neuronen erzeugt. Die input_dim(ensionen) geben die Anzahl der Eingabewerte an. Danach wird die Aktivierungsfunktion festgelegt und ein Dropout auf die Schicht angewendet. Zuletzt wird die Ausgabeschicht als eine vollkommen verbundene Schicht in der Größe des Ausgabevektors erzeugt und eine Aktivierungsfunktion hinzugefügt.

```
model = Sequential()
model.add(Dense(dls1, input_dim=78, activation='relu'))
model.add(Activation('relu'))
```

```
model.add(Dropout(dropout))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Nun werden die Parameter ausgewählt, welche für die Kompilierung des Models notwendig sind. Dabei handelt es sich um die Verlustfunktion, den Optimierer und die Metrik, nach der optimiert werden soll. Da die Lernraten der Optimierer direkt angegeben werden, müssen diese zuvor definiert werden.

```
if optimizer=="sgd":
    opt = optimizers.SGD(lr=learning_rate)
else:
    opt = optimizers.Adam(lr=learning_rate)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

return model
```

Nach Erstellung des Models werden die benötigten Daten importiert. Beispielhaft wird in folgendem Codeausschnitt gezeigt, wie die Eingabe-Trainingsdaten geladen und verändert werden.

```
#import data
a_file = open("input_data_versch_train.pkl", "rb")
X_train = pickle.load(a_file)
a_file.close()
```

```
X_train = numpy.array(X_train)
```

Der Array muss in einen Numpy-Array umgewandelt werden, bevor er dem Netz zugeführt werden kann.

Das Model sowie andere Argumente werden in Wandb gespeichert.

```
log_model_params(model, wandb.config, args, len(X_train))
```

Nun kann der Trainingsprozess gestartet werden.

```
model.fit(X_train, y_train,  validation_data=(X_test, y_test), \
      epochs=args.epochs, callbacks=\
      [WandbCallback(data_type="values", labels=labels)])
```

Ist das Training abgeschlossen, wird das trainierte Model des neuronalen Netzes gespeichert und das Programm wird beendet.

```
save_model_filename = args.model_name + ".h5"
model.save(save_model_filename)
```

Der Run kann nun auf der Wandb-Webseite genauer nachvollzogen werden.

8.2 Vorgehensweise und Evaluation

Anmerkung: Alle Durchläufe und Trainingsergebnisse finden sich auf dem mitgelieferten Datenträger in einer zusammengeführten Excel-Tabelle wieder. Zu empfehlen ist jedoch die interaktive Betrachtung der Ergebnisse auf der Weights&Biases-Webseite. Die passenden Verlinkungen dazu, sind im Anhang auf Seite 72 aufgeführt.

Zu Beginn des Trainings war zunächst die strukturelle Korrektheit der Daten von Interesse. Zu diesem Zweck wurde eine feste Struktur eines neuronalen Netzes entworfen und implementiert. Diese Struktur bestand aus einer Eingabeschicht mit 78 Neuronen und zwei vollverbundenen verdeckten Schichten mit 50 und zehn Neuronen. Sowie einer Ausgabeschicht mit fünf Neuronen. Die Ergebnisse der ersten Durchläufe finden sich in Tabelle 1 wieder. Anhand dieser Durchläufe konnten bereits einige Aussagen getroffen werden. Erstens: Die erstellten statischen Daten besitzen die richtige Struktur und ein neuronales Netz lässt sich mithilfe dieser Daten trainieren. Zweitens: Die hohe Genauigkeit lässt darauf schließen, dass die gewählte Architektur für das Vorhaben geeignet ist. Drittens: Die Variation innerhalb der Genauigkeit der verschiedenen Durchläufe zeigt, welche Auswirkung unterschiedliche Anfangsgewichtungen auf das Training haben.

Um die Performance von neuronalen Netzen zu verbessern, gibt es mehrere Stellschrauben, die ein Entwickler zur Verfügung hat. Die wichtigsten dieser Möglichkeiten wurden in diesem Projekt experimentell bestimmt. Zunächst wurde die Struktur der Ausgabeschicht variiert. Wie dem Kapitel Auswahl7.9 bereits erwähnt, gibt es bei einer Klassifizierung zwei Optionen, die sich in der Praxis bewährt haben. Zum einen ist dies eine Softmax-Schicht in Kombination mit einem Kreuzentropie-Error und zum anderen eine Schicht mit einer ReLU-Aktivierungsfunktion und einer mittleren quadratischen Abweichung. In den ersten Durchläufen wurde die Softmax-Variante getestet, welche bereits gute Ergebnisse lieferte. Nun wurde die ReLU-Variante dem gegenübergestellt. Bereits nach zwei Durchläufen wurde klar, dass auch diese Variante vergleichbar gute Werte lieferte.

Das Framework von Weight&Biases stellt dem Nutzer eine Funktion zu Verfügung, mit welcher Hyperparameter variiert und somit eine Vielzahl an Durchläufen automatisiert ausgeführt werden kann. In einem Sweep wird ein sogenannter Agent erschaffen, welcher nach Durchlauf des Trainingsskriptes dieses mit anderen Parametern erneut aufruft. Der Nutzer trägt zuvor die zu überprüfenden Parameter in eine .yaml-Datei ein. Dies kann wie folgt aussehen:

```yaml
program: train_nn_v1_mse_sweep.py
method: bayes
metric:
  name: val_loss
  goal: minimize
parameters:
  learning_rate:
    min: 0.001
```

```
    max: 0.1
  optimizer:
    values: ["adam", "sgd"]
```

Definiert werden muss, welches Trainingsprogramm ausgeführt werden soll, die zu verwendenden Werte, sowie mit welcher Methodik diese verwendet werden sollen. In diesem Fall werden verschiedene Lernraten mit zwei unterschiedlichen Optimierern getestet. Bayesian Optimierung verwendet einen Gaußschen Prozess, um die Wahrscheinlichkeiten der Verbesserung zu optimieren. Eine genaue Beschreibung des Vorgangs und weitere Beispiele finden sich auf der Webseite von Weights&Biases.

Die besten Ergebnisse dieses Sweeps sind im Anhang auf Seite 73 zu finden. An ihnen lässt sich erkennen, dass der Optimierer Adam die besten Werte hervorbrachte. Ebenso sind die Lernraten mit der höchsten Genauigkeit nah an dem, in Keras festgelegten Standardwert von 0.001 angesiedelt.

Weiterführend wurden Untersuchungen zum Aufbau des neuronalen Netzes vorgenommen. In den Tabellen 4 und 5 sind Durchläufe zu sehen, in denen die Neuronenanzahl in beiden verdeckten Schichten variiert wurden. Diese Durchläufe stellten sich jedoch als obsolet heraus, da die Tests mit nur einer verdeckten Schicht deutlich bessere Ergebnisse lieferten (siehe Tabelle 6 & 7). Die Konfigurationen mit einem geringen Dropout und einer Neuronenzahl von mindestens 20 zeigten die beste Performance. Der Unterschied im Verhalten zwischen einer Softmax- und einer ReLU-Konfiguration lässt sich am besten durch folgendes Schaubild darstellen.

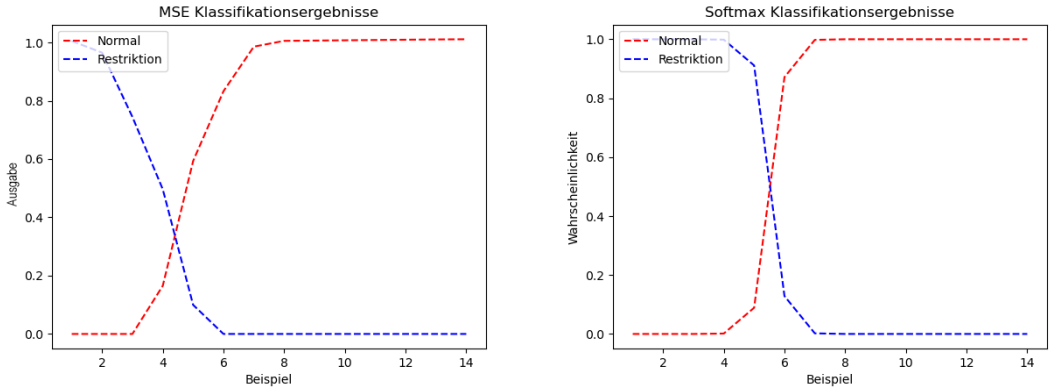


Abbildung 30 Vergleich ReLU+MSE und Softmax+CEE

Die in Abbildung 30 gezeigten Beispiele stellen einen Verlauf von einer Restriktion zu einem gesunden Patienten hin dar. Ohne das Zwischenwerte trainiert wurden, lässt sich das Verhalten der verschiedenen Konfigurationen deutlich beobachten. Die ReLU und MSE Variante, auf der linken Seite, besitzt einen schwächeren Verlauf von einer Klasse zur anderen, während auf der Softmax-Darstellung ein nahezu sprunghafter Wechsel der Klassen zu erkennen ist. Aus diesem Grund lässt sich eine ReLU&MSE-Konfiguration an der Ausgabeschicht des Netzes tendenziell besser, für eine Trendvorhersage verwenden als eine Softmax-Schicht. Ein Nachteil von ReLU ist jedoch, dass die

Ausgaben nicht auf Eins normiert werden, weshalb die Ausgabe nicht direkt als Wahrscheinlichkeit interpretiert werden kann.

In der Trainingsskript-Version 3 wurden die verwendeten Daten von statischen Daten auf die laufenden Daten, welche mit dem Datengenerator aus Kapitel 6.6 erzeugt wurden, geändert. Die Kurven wurden kontinuierlich verschoben um einen Eingangsvektor, wie er in einem realen Einsatz des Systems auftreten würde, zu simulieren.

Bei den Anpassungen der Netze wurde dabei lediglich die Batchgröße verändert, um auf die neue Form der Daten zu reagieren. Ansonsten wurde auf den Erkenntnissen der vorherigen Versionen aufgebaut. Die Lernrate wurde zwischen 0,001 und 0,002 gewandelt. Und die Neuronenanzahl für eine verdeckte Schicht variiert. Ein Dropout wurde ebenfalls getestet, mit Werten zwischen 0 und 0,5. Auch hier lieferten Konfigurationen mit einer höheren Neuronenzahl und einem niedrigen beziehungsweise nicht existenten Dropout die besten Ergebnisse. Eine Genauigkeit von 1, sowohl bezüglich der Trainings- als auch der Testdaten, konnte bei beiden Variationen (Softmax&ReLU) erreicht werden (siehe Tabelle 8 & 9). Weiterhin wurde die Mindestanzahl an Trainingsdaten geprüft. Die Ergebnisse sind in Tabelle 10 zu finden. Können aber durch folgende Darstellung visualisiert werden:

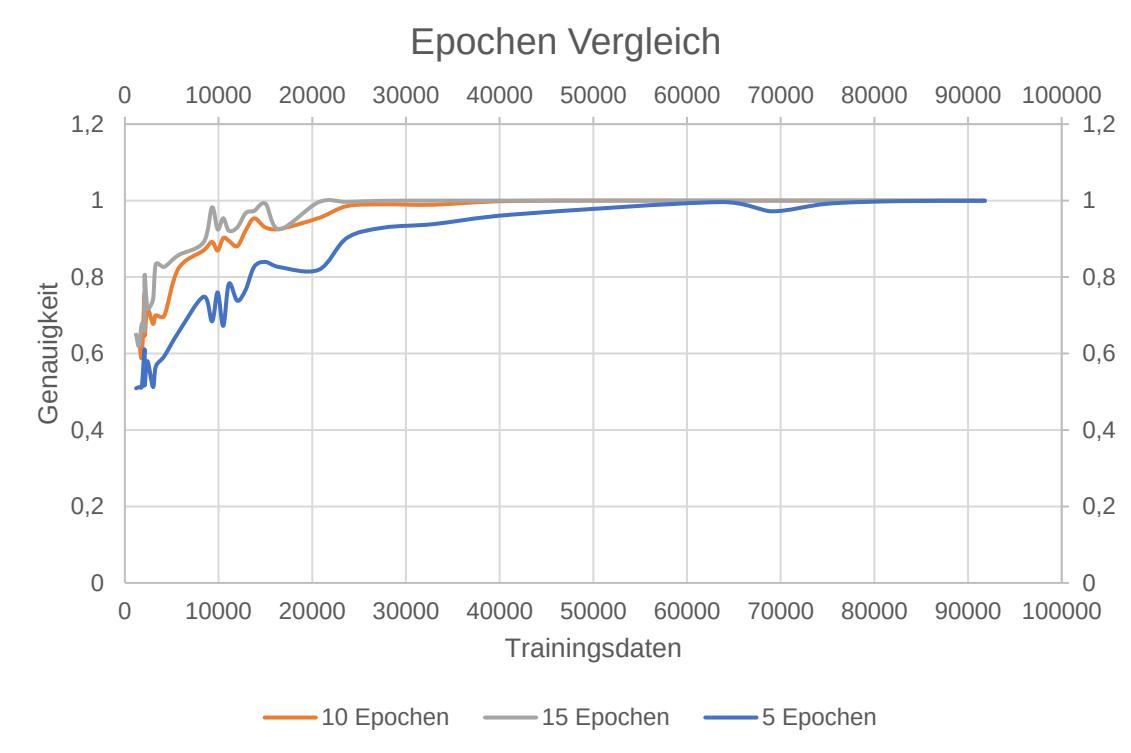


Abbildung 31 Trainingsdaten Epochen Vergleich

In Abbildung 31 ist die Genauigkeit, der Anzahl der vorhandenen Trainingsdaten gegenübergestellt. Untersucht wurde dabei ebenfalls, welche Auswirkung die Anzahl der Epochen auf das Ergebnis hat. Zu erkennen ist zunächst einmal, dass kein kontinuierlicher Verlauf der Kurve zu beobachten ist. Dies liegt, wie zuvor bereits erwähnt, an den

zufallsbasierten Startwerten und kann durch eine Vielzahl an Wiederholungen bereinigt werden. Dies ist jedoch gar nicht notwendig, da die Abweichungen von der Genauigkeit 1.0 deutlich zu erkennen sind. Bei 5 Epochen verliert das Netz bereits ab etwa 41400 Daten oder 110 Beispielen pro Variante[1] an Genauigkeit. Bei 10 Epochen ist eine Abweichung ab 20700 Daten oder 55 Beispielen zu erkennen und bei 15 Epochen sinkt die benötigte Anzahl auf 16500 Daten oder 44 Beispiele. Es lässt sich also ein deutlicher Trend erkennen. Mit steigender Epochenanzahl verringert sich die benötigte Zahl der Trainingsdaten. Dies lässt sich jedoch nicht unbegrenzt fortführen. Ab einer bestimmten Anzahl an Epochen kann, bei Realdaten, ein Overfitting auftreten und das Netz liefert schlechtere Werte bezüglich der Testdaten. Deshalb muss auch in diesem Fall ein Optimum zwischen Anzahl der Trainingsdaten, der Epochen und dem Dropout, welcher diesen Effekt herauszögern kann, gefunden werden.

Zuletzt wurde geprüft, ob das Netz eine tatsächliche Verbindung zwischen den personenbezogenen Daten und der Kurve herstellen konnte, oder nur eine Klassifikation anhand der separierten Kurve stattfindet. Hierzu wurden die personenspezifischen Daten verändert, während die Kurven gleichblieben. Die Kurvenwerte, welche verwendet wurden, stammen von einem Mann, mit einer Körpergröße von 1,65 Meter und einem Alter von 70 Jahren. Nun wurden bei einem Durchlauf die personenspezifischen Daten des Mannes verändert. Die Klassifizierung fand nun, für einen Mann mit einer Körpergröße von 1,95 Meter und einem Alter von 25 Jahren, statt. Die Grenzwerte der Daten wurden verwendet, um eine deutliche Änderung bei der Klassenzuordnung zu erreichen.

	Normal	Emphysem	Restriktion	Stenose	**Asthma**
1,65m, 70J	1,23e-23	3,54e-04	2,23e-21	3,01e-15	**9,99e-01**
1,95m, 25J	8,30e-22	**9,93e-01**	6,55e-14	1,49e-14	6,06e-03

Tabelle 1 Asthma

	Normal	Emphysem	Restriktion	Stenose	Asthma
1,65m, 70J	**9,99e-01**	4,65e-08	9,76e-06	1,08e-05	1,20e-31
1,95m, 25J	6,10e-03	1,38e-06	**9,93e-01**	3,81e-08	5,11e-35

Tabelle 2 Normal

[1] 41400 Daten / (5 Klassen * 75 Verschiebungen) ≈ 100 Beispiele

Wie zu sehen ist, wurde allein anhand der Veränderung der personenspezifischen Daten die Zuordnung der Klasse verändert. Die Asthmakurve wurde als ein Emphysem und der Normalfall als eine Restriktion klassifiziert. Dies zeigt zum einen, dass das Netz eine direkte Verbindung zwischen den Patientendaten und der Kurve hergestellt hat. Zum anderen zeigt es allerdings auch, welche gravierenden Auswirkungen eine Falscheingabe, und sei es nur ein Tippfehler bei dem Geburtsjahr oder der Körpergröße, haben kann.

9 Zusammenfassung

Die Basis der Arbeit bildete die Erstellung eines Datensatzes für das Training der neuronalen Netze. Da es im Rahmen der Arbeit nicht möglich war, ein bereits existentes Datenset oder sogar reale Patientendaten zu verwenden, musste das Datenset erzeugt werden. Hierzu wurden durch Literaturrecherche die möglichen medizinischen Parameter überprüft und der Volumenverlauf als adäquates Signal ausgewählt. Durch die Variation dieser Kurve wurden vier verschiedene Krankheitsbilder generiert. Zusammen mit der Normalform des Volumenverlaufes, wurden die Kurven durch personenspezifische Parameter verändert, um eine Klassifizierung unter Einbezug des Patienten zu ermöglichen. In der Erweiterung des Datensets wurden dann laufende Kurven hinzugefügt, also die Ursprungskurve der Abtastfrequenz entsprechend verschoben, um ein möglichst realistisches Bild zu erzeugen. Bei der Thematik Daten wurde sowohl auf die medizinischen und mathematischen Grundlagen eingegangen als auch auf den Umgang mit den Daten und die Programmierung des Datengenerators.

Im Weiteren wurden die theoretischen Grundlagen der Basisarchitekturen und Sonderfälle der neuronalen Netze sowie andere wichtige Parameter, welche die Erstellung und das Training der Netze betreffen, erläutert. Ein grundlegendes Verständnis von Trainingsverfahren, Optimierern, Verlust- und Aktivierungsfunktionen ist für die Auswahl und das darauf aufbauende Training der Netze notwendig.

Zur Bestimmung der besten Parameterkombinationen wurden verschiedene Hyperparameter des Netzes untersucht. Begonnen wurde mit der Struktur des Netzes. Zu Beginn wurde ein Netz mit zwei verdeckten Schichten ausgewählt und aufbauend auf diesem die Optimierer und Lernraten getestet. Nach einigen Revisionen wurde die Struktur auf eine verdeckte Schicht reduziert, was die Genauigkeit des Netzes deutlich erhöhte. Eine wichtige Thematik, welche sich durch die Arbeit zieht, ist der Vergleich zwischen den verschiedenen Kombinationen von Ausgangsschichten. Verglichen wurden hierbei eine Softmax-Schicht mit einem Kreuzentropie-Error und eine ReLU-Verlustfunktion auf der Ausgabeschicht in Zusammenspiel mit einer mittleren quadratischen Abweichung. Die Unterschiede sowie die Vor- und Nachteile der verschiedenen Schichten wurden ebenso aufgezeigt, wie die Auswirkung eines Dropouts auf die Performance des Netzes.

Für den Fall der laufenden Daten, lieferte folgende Konfiguration vielversprechende Ergebnisse:

- Feedforward Netzarchitektur
- Lernrate bei 0.001
- Eine verdeckte Schicht mit etwa 35 Neuronen
- Adam als Optimierer
- Kein Dropout
- Sowohl Softmax als auch ReLU möglich

Diese Konfiguration kann als Basis für weitere Test gesehen werden. Insbesondere im Zuge der Erstellung neuer Daten können die aufgezeigten Methoden und Vorgänge

verwendet werden, um schnell und einfach eine Vielzahl an Parametern durchzuspielen und ein neues Netz, passend zu den neuen Daten, generieren zu können.

Zusammenfassend kann man sagen, dass es mit einer Feedforward Netzarchitektur möglich ist, eine Klassifizierung von Zeitreihen, die von personenspezifischen Daten abhängen, vorzunehmen.

10 Ausblick

Wie bereits zuvor erwähnt, soll diese Arbeit als Proof-of-Concept und als Basis für weitere Entwicklungen in dieser Thematik dienen. Im Folgenden wird deshalb eine persönliche Einschätzung/Empfehlung bezüglich der möglichen nächsten Schritte gegeben.

Bereits in dieser Arbeit stellte der Datensatz einen essenziellen Teil des Gesamtsystems dar. In Zukunft ist es jedoch noch wichtiger ein möglichst umfassendes Datenset zu kreieren, welches nah an der Realität liegt. Die erstellten Daten basieren auf wissenschaftlichen Erkenntnissen, sollten aber nicht als einen Ersatz für reale Patientendaten gesehen werden. Es ist unerlässlich, entweder simulativ oder durch die tatsächliche Aufnahme an Personen, ein breites Datenspektrum zu generieren. Sollte die Beschaffung von Patientendaten aber aus verschiedenen Gründen weiterhin nicht möglich sein, empfiehlt sich eine Weiterentwicklung des Datengenerators. Beispielsweise können Variationen der Daten hinzugefügt werden. Bleibt man bei den Fluss-Volumen-Kurven, finden sich in der Literatur eine Vielzahl an unterschiedlichen Krankheitsbildern. Allerdings sollte dann auch über einen neuen, möglichst komplett automatisierten Weg der Generierung der Daten nachgedacht werden. Vielleicht sind Zugriffe auf Datenbanken mit passenden Informationen eine Option, um das Datenset zu erweitern. In der Realität ist es schwierig Diagnosen zu finden, die einer Krankheit einen numerischen Wert zuordnen, weshalb es schwierig werden könnte einen Datensatz zu entwerfen indem Verläufe und Zwischenwerte vorhanden sind. Sollte dies jedoch gelingen, bietet sich die Möglichkeit, direkt anhand der Ausgaben des Netzes eine Trenderkennung zu implementieren. Die Addition von Zufallswerten, um den Daten ein Rauschen hinzuzufügen, dürfte zum einen die Ausgaben resilienter machen und zum anderen bei der Generalisierung des Netzes helfen. Zusammengefasst kann man sagen: Mit der Qualität der vorhandenen Daten, steigen und fallen die Ergebnisse des neuronalen Netzes.

Mit der in dieser Arbeit gezeigten Vorgehensweise, kann das neuronale Netz schnell auf neue Daten angepasst werden, weshalb die Priorisierung der Entwicklung eines neuen Netzes wahrscheinlich hinter dem Ausbau und der Beschaffung der Daten zurückstehen sollte.

Eine Prüfung der messtechnischen Möglichkeiten des zu entwickelnden Beatmungsgerätes und darauffolgend eine dauerhafte Festlegung der Art der Daten. Dies bedeutet für den in dieser Arbeit behandelten Fall eine Entscheidung zwischen dem Fluss, dem Volumen oder einer Kombination aus beiden. Im Falle von anderen sich ergebenden Parametern können jedoch auch die Daten neu erzeugt werden. Die Erweiterung der Krankheitsbilder sowie deren Festlegung ist ebenso wichtig für das weitere Vorgehen, wie der Umgang mit Krankheitsbildern, welche nicht den vorgegebenen Klassen entsprechen.

Sollte die Datenbasis ausreichend und die beschriebenen Schritte und Schwierigkeiten überwunden sein, besteht die Möglichkeit das neuronale Netz in ein Gesamtdiagnosetool einzubinden. Zu den bisher erkannten Symptomen, können dann andere Messwerte und personenspezifische Daten hinzugefügt werden, um eine endgültige Diagnose zu ermöglichen.

11 Literatur

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, …, *Solving Rubik´s Cube with a Robot hand*. A Perprint, 2019. [Online]. Verfügbar unter: https://arxiv.org/pdf/1910.07113.pdf

[2] OpenAI, *MuseNet.* [Online]. Verfügbar unter: https://openai.com/blog/musenet/ (Zugriff am: 18. September 2020).

[3] Amin Dada, „Long Short-Term Memory zur Generierung von Musiksequenzen: Überarbeitete Version, 16.03.2018“. Bachelorarbeit, Ruhr-Universität Bochum, 2018. [Online]. Verfügbar unter: https://www.ini.rub.de/upload/file/1521461530_7126db755dc03bec85b1/dada-bsc.pdf

[4] Alexander Mordvintsev et al, *Inceptionism: Going Deeper into Neural Networks.* [Online]. Verfügbar unter: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html (Zugriff am: 18. September 2020).

[5] Roland Berger GmbH / VDMA, „Predictive Maintenance: Service der Zukunft - und wo er wirklich steht“, Apr. 2017.

[6] PWC, „Predictive Maintenance 4.0: Predict the unpredictable“, Juni 2017.

[7] Jonas Fausing Olesen, Hamiz Reza Shaker, „Predictive Maintenance for Pump Systems and Thermal Power Plants: State-of-the-Art Review, Trends and Challenges“, University of Southern Denmark, 2020.

[8] Yongyi Ran et al, „A Survey of Predictive Maintenance: Systems, Purposes and Approaches“ in *IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. XX, NO. XX*.

[9] Shang-Yi Chuang, Nilima Sahoo, Hung-Wei Lin, Yeong-Hwa Chang, „Predictive Maintenance with Sensor Data Analytics on a Raspberry Pi-Based Experimental Platform“, Department of Electrical Engineering, Chang Gung University, Lee-Ming Institute of Technology, Ming Chi University of Technology, 2020.

[10] Siemens Healthineers, *AI-Rad Companion: Providing multi-modality imaging decision support.* [Online]. Verfügbar unter: https://www.siemens-healthineers.com/digital-health-solutions/digital-solutions-overview/clinical-decision-support/ai-rad-companion#COVID-19 (Zugriff am: 17. September 2020).

[11] Mark Hughes, Irene Li, Toyotaro Suzumura, „Medical Text Classification using Convolutional Neural Networks“, IBM Research Lab, Japan Science and Technology Agency, IBM TJ Watson Research Center, 11. Sep. 2019.

[12] Eric Topol, Hg., *Deep medicine: How artificial intelligence can make healthcare human again*, 2019.

[13] G. D. Rey und K. F. Wender, *Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung,* 1. Aufl. Bern: Huber, 2008. [Online]. Verfügbar unter: http://sub-hh.ciando.com/book/?bok_id=15205

[14] Frank Ritter und Martin Döring, *Kurven und Loops in der Beatmung*.

[15] Thieme, *Kurzinfo zum Umgang mit COVID-19*.

[16] C.-P- Criée et al, *Leitlinie zur Spirometrie: Leitlinie der Deutschen Atemwegsliga, der Deutschen Gesellschaft für Pneumologie und Beatmungsmedizin und der Deutschen Gesellschaft für Arbeitsmedizin und Umweltmedizin zur Spirometrie*, 2015.

[17] Markus Gnädinger, Monika Curschellasb, Nadja Nattererc, Robert Thurnheer, „Praxis-Spirometrie“ in *Schweiz Med Forum*, S. 683–688.

[18] Mareike Müller, *Asthma.* [Online]. Verfügbar unter: https://www.netdoktor.de/krankheiten/asthma/ (Zugriff am: 23. September 2020).
[19] Deutsche Lungenstiftung e.V., *Was ist ein Lungenemphysem?* [Online]. Verfügbar unter: https://www.lungenaerzte-im-netz.de/krankheiten/lungenemphysem/was-ist-ein-lungenemphysem/ (Zugriff am: 23. September 2020).
[20] Python-Kurs, *Dictionaries.* [Online]. Verfügbar unter: https://www.python-kurs.eu/dictionaries.php (Zugriff am: 23. September 2020).
[21] Wikipedia, *Künstliches Neuron.* [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz#/media/Datei:NeuronModel_deutsch.svg (Zugriff am: 14. Juli 2020).
[22] Dipl.-Ing.(FH) Stefan Luber und Nico Litzel, *Was ist ein rekurrentes neuronales Netz (RNN)?* [Online]. Verfügbar unter: https://www.bigdata-insider.de/was-ist-ein-rekurrentes-neuronales-netz-rnn-a-843274/ (Zugriff am: 16. Juli 2020).
[23] N. Ketkar, *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017. [Online]. Verfügbar unter: http://www.springer.com
[24] Manuel Meyer, „Klassifizierung Multidimensionaler Zeitreihen mithilfe von Deep Learning“, Hamburg University of Applied Sciences.
[25] Hanna Bader, „Interpolation von Volumendaten mit Neuronalen Netzen“. Bachelorarbeit, Institut für Visualisierung und Interaktive Systeme, Universität Stuttgart, Stuttgart, 2018.
[26] J. Odenthal, „Convolutional Neural Networks: Machine Learning Seminar“, Universität Köln. [Online]. Verfügbar unter: http://www.mi.uni-koeln.de/wp-znikolic/wp-content/uploads/2019/06/11-Odenthal.pdf
[27] Stefan Selle, „Künstliche Neuronale Netzwerke und Deep Learning“, Fakultät für Wirtschaftswissenschaften, Hochschule für Technik undWirtschaft des Saarlandes, Saarbrücken, 2018.
[28] JAAI, *Convolutional Neural Networks - Aufbau, Funktion und Anwendungsgebiete.* [Online]. Verfügbar unter: https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/ (Zugriff am: 16. Juli 2020).
[29] S. Alla und S. K. Adari, *Beginning Anomaly detection using Python-based deep learning: With Keras and PyTorch*. Apress, 2019.
[30] Xiaofeng Yuan, Lin Li und Yalin Wang, „Nonlinear dynamic soft sensor modeling with supervised long short-term memory network“, Central South University, 2019.
[31] VDMA Software und Digitalisierung, „Quick Guide Machine Learning im Maschinen- und Anlagenbau“, Frankfurt am Main, 2018.
[32] HTW Dreseden, *Delta-Regel.* [Online]. Verfügbar unter: https://www2.htw-dresden.de/~boehme/Neuroinformatik/GNI_Prakt_TUI/ni_grundlagen_prak/deltalearning/deltalearning.html (Zugriff am: 9. Juli 2020).
[33] Anna A. E. Baumgard, „Algorithmen der Neuronalen Netze“, Hochschule für Technik und Wirtschaft Berlin, 2016.
[34] Technische Universität Ilmenau, *Delta-Regel.* [Online]. Verfügbar unter: https://www.tu-ilmenau.de/neurob/teaching/ss/ni-aktiv/aktivuebung-aufgabenserien-und-applets/delta-regel/ (Zugriff am: 9. Juli 2020).
[35] M. Rauscher, *Künstliche neuronale Netze zur Risikomessung bei Aktien und Renten: Am Beispiel deutscher Lebensversicherungsunternehmen*. Wiesbaden: Deutscher Universitätsverlag, 2004.
[36] A. Kathuria, „Intro to optimization in deep learning: Gradient Descent“, *Paperspace Blog*, 2. Juni 2018, 2018. [Online]. Verfügbar unter:

https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/. Zugriff am: 11. Juli 2020.
[37] Steffen Bucher, „MLP - Multi-Layer-Perzeptrone“, 2004.
[38] A. Nandy und M. Biswas, *Reinforcement learning: With Open AI, TensorFlow and Keras Using Python*. Berkeley, CA: Apress, 2018.
[39] Wikipedia, *Bestärkendes Lernen.* [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Best%C3%A4rkendes_Lernen (Zugriff am: 8. Juli 2020).
[40] A. Ng und K. Soo, *Data Science - was ist das eigentlich?!: Algorithmen des maschinellen Lernens verständlich erklärt*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018.
[41] *Loss Functions — ML Glossary documentation.* [Online]. Verfügbar unter: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html (Zugriff am: 23. Juli 2020).
[42] Sebastian Ruder, „An overview of gradient descent optimization algorithms“, Insight Centre for Data Analytics, NUI Galway, Aylien Ltd., Dublin, 2017.
[43] Wikipedia, *Stochastic gradient descent* (Zugriff am: 27. Juli 2020).
[44] C. Hansen, „Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent“, *Machine Learning From Scratch*, 16. Okt. 2019, 2019. Zugriff am: 27. Juli 2020.
[45] Aleksey Bilogur, „Keras optimizers“, *Kaggle*, 2. Dez. 2018, 2018. [Online]. Verfügbar unter: https://www.kaggle.com/residentmario/keras-optimizers. Zugriff am: 28. Juli 2020.
[46] J. Brownlee, „Dropout Regularization in Deep Learning Models With Keras“, *Machine Learning Mastery*, 19. Juni 2016, 2016. [Online]. Verfügbar unter: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/. Zugriff am: 29. September 2020.
[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“, Department of Computer Science, University of Toronto. [Online]. Verfügbar unter: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
[48] Fabian Beck, *Neuronale Netze - Eine Einführung - Aktivität.* [Online]. Verfügbar unter: http://www.neuronalesnetz.de/aktivitaet.html (Zugriff am: 28. Juli 2020).
[49] *Schwellenwertfunktion.* [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron#/media/Datei:Hard-limit-function.svg (Zugriff am: 28. Juli 2020).
[50] Martin Knöfel, „Künstliche Neuronale Netze: Hauptseminar“, Fakultät Informatik, Technische Universität Dresden, Dresden, 2017. [Online]. Verfügbar unter: https://tu-dresden.de/ing/informatik/ti/vlsi/ressourcen/dateien/dateien_studium/dateien_lehstuhlseminar/vortraege_lehrstuhlseminar/lehrstuhlseminar_ws17_18/KNN_Knoefel_Praesentation.pdf?lang=de

12 Abbildungsverzeichnis

13 Anhang

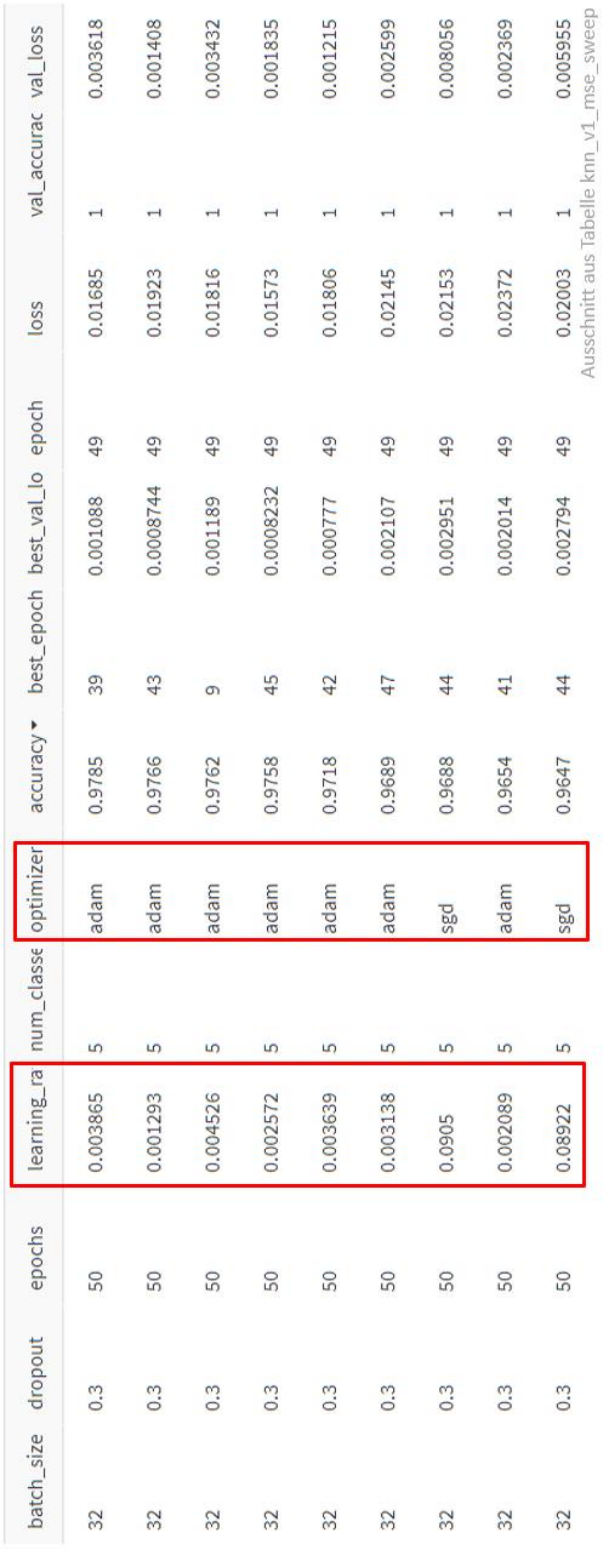
Trainingsergebnisse

Aufgrund der Größe und der Formate der Tabellen, können diese hier nicht aufgeführt werden. Zu finden sind die Tabellen entweder auf dem mitgelieferten Speichermedium in einer zusammengeführten Excel-Tabelle, mit dem Namen Runs_Tabelle oder unter folgenden Links, interaktiv auf der Weights&Biases-Webseite:

Tabellen-nummer	Projektname	Online-Link
1	KNN_v0	https://wandb.ai/auten-rieth/KNN_v0
2	KNN_v0_mse	https://wandb.ai/auten-rieth/KNN_v0_mse
3	KNN_v1_mse_sweep	https://wandb.ai/auten-rieth/knn_v1_mse_sweep
4	KNN_v2_mse_sweep_lr0035	https://wandb.ai/auten-rieth/knn_v2_mse_sweep_lr0035
5	KNN_v2_mse_sweep_lr001	https://wandb.ai/auten-rieth/knn_v2_mse_sweep_lr001
6	KNN_v2_mse_sweep_1layer	https://wandb.ai/auten-rieth/knn_v2_mse_sweep_1layer
7	KNN_v2_soft-max_sweep_1layer	https://wandb.ai/auten-rieth/knn_v2_soft-max_sweep_1layer
8	KNN_v3_soft-max_sweep_1layer	https://wandb.ai/auten-rieth/knn_v3_soft-max_sweep_1layer
9	KNN_v3_mse_sweep_1layer	https://wandb.ai/auten-rieth/knn_v3_mse_sweep_1layer
10	KNN_v3_soft-max_sweep_trainingsdata	https://wandb.ai/auten-rieth/knn_v3_softmax_sweep_trai-ningsdata

Anhang 1 Trainingsergebnisse

Ausschnitt aus Tabelle 2



batch_size	dropout	epochs	learning_ra	num_classe	optimizer	accuracy	best_epoch	best_val_lo	epoch	loss	val_accurac	val_loss
32	0.3	50	0.003865	5	adam	0.9785	39	0.001088	49	0.01685	1	0.003618
32	0.3	50	0.001293	5	adam	0.9766	43	0.0008744	49	0.01923	1	0.001408
32	0.3	50	0.004526	5	adam	0.9762	9	0.001189	49	0.01816	1	0.003432
32	0.3	50	0.002572	5	adam	0.9758	45	0.0008232	49	0.01573	1	0.001835
32	0.3	50	0.003639	5	adam	0.9718	42	0.000777	49	0.01806	1	0.001215
32	0.3	50	0.003138	5	adam	0.9689	47	0.002107	49	0.02145	1	0.002599
32	0.3	50	0.0905	5	sgd	0.9688	44	0.002951	49	0.02153	1	0.008056
32	0.3	50	0.002089	5	adam	0.9654	41	0.002014	49	0.02372	1	0.002369
32	0.3	50	0.08922	5	sgd	0.9647	44	0.002794	49	0.02003	1	0.005955

Ausschnitt aus Tabelle knn_v1_mse_sweep

Anhang 2 Ausschnitt aus Tabelle 2