

Smart Home Interior -

Erkennung von Interaktionen mit einem Sofa
auf Basis eines neuronalen Netzes

BACHELORARBEIT

ausgearbeitet von

Jan Schröder

zur Erlangung des akademischen Grades
BACHELOR OF SCIENCE (B.Sc.)

vorgelegt an der

TECHNISCHEN HOCHSCHULE KÖLN
CAMPUS GUMMERSBACH
FAKULTÄT FÜR INFORMATIK UND
INGENIEURWISSENSCHAFTEN

im Studiengang
INFORMATIK

Erster Prüfer/in: Prof. Dr. Matthias Böhmer
Technische Hochschule Köln

Zweiter Prüfer/in: Prof. Dr. rer. nat. Wolfgang Konen
Technische Hochschule Köln

Gummersbach, 21. Januar 2020

Adressen: Jan Schröder
 Wilhemstr. 12
 51643 Gummersbach
 jan.schroeder1@smail.th-koeln.de

Prof. Dr. Matthias Böhmer
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
matthias.boehmer@th-koeln.de

Prof. Dr. rer. nat. Wolfgang Konen
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
wolfgang.konen@th-koeln.de

Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation und Relevanz zum Thema Smart Homes	3
1.2 Idee zur Umsetzung vom Regelsystem zum neuronalen Netz	4
1.3 Personen an die das System gerichtet ist	4
1.4 Ziel der Bachelorarbeit	4
2 Grundlagen aus dem IoT für das Thema Smart Home Interior	6
2.1 Smart Home Aufbau und Funktion	6
2.2 Relevanz neuronaler Netze in Smart Homes	7
2.3 Smart Home Interior	7
2.4 Neuronale Netze zur Klassifizierung	8
2.5 MQTT zur Datenübertragung	13
2.6 Sensoren für die Interaktionen auf dem Sofa	14
2.7 Prototyp aus dem Praxisprojekt	15
3 Architektur und Aufbau des Prototyps	17
3.1 Anforderungen an die Architektur	17
3.2 Architektur des Sofaprototypen aus dem Praxisprojekt	18
3.3 Die Anpassung der Architektur für das neuronale Netz	19
3.4 Regelsystem zur Steuerung der Interaktionen	20
3.5 Verarbeitung der Daten im ESP und Raspberry Pi	21
3.6 Steuerungsarten eines Smart Homes	22
3.7 Kommunikation der Sensoren mit dem neuronalen Netz	22
3.8 Gegenüberstellung eines neuronalen Netzes mit TensorFlow und Numpy	24
3.9 Zusammenfassung und Ausblick	24
4 Umsetzung des neuronalen Netzes zur Erkennung von Interaktionen	26
4.1 Anforderungen an die Umsetzung	26
4.2 Umsetzung des neuronalen Netzes mit Numpy	26
4.3 Umsetzung des neuronalen Netzes mit Keras	28
4.4 Die Backend-Engine TensorFlow	30
4.5 Neue Funktionen durch das neuronale Netz	31
4.6 Einbindung des neuronalen Netzes in den Prototypen	32
4.7 Weiterführung der Entwicklung des neuronalen Netzes	32
4.8 Verbesserungen des Prototypen	32
5 Evaluation des Trainings durch Probanden	34
5.1 Umgebung zur Erkennung von Interaktionen	34
5.2 Soll-Ablauf der Situation	35
5.3 Tatsächlicher Ablauf aller Probanden	36

5.4	Aufbau des Datensatzes	36
5.5	Ergebnisse der Datensammlung	38
5.6	Ergebnisse und Evaluation des neuronalen Netzes	38
5.7	Zusammenfassung der Evaluation	40
5.8	Erkenntnisse aus der Evaluation zum Prototypen	40
6	Fazit und Aussicht für den Prototypen	42
6.1	Zusammenfassung der Bachelorarbeit	42
6.2	Zukunftsauussicht für das System	43
7	Anhang	45
	Abbildungsverzeichnis	50
	Tabellenverzeichnis	51
	Literaturverzeichnis	53

Vorwort zum Aufbau der Bachelorarbeit

Die Arbeit gliedert sich in sechs Teile. Der erste Teil widmet sich der Einleitung und einem Einblick in den Inhalt. Daraufhin werden im zweiten Teil die Grundlagen behandelt. Diese sollen ein Verständnis vermitteln, um die weiteren Kapitel der Bachelorarbeit zu verstehen.

Anschließend behandelt der Autor im dritten Teil der Arbeit die Architektur des Prototypen, welcher in der Bachelorarbeit benutzt wird. Zusätzlich behandelt dieser Teil die Architektur des Prototypen aus dem Praxisprojekt um die späteren Veränderungen zu veranschaulichen. Dies ist notwendig, da der Autor die Veränderungen in den weiteren Teilen der Bachelorarbeit beschreibt.

Auf der Grundlage aus dem zweiten Teil der Arbeit wird die Umsetzung des neuronalen Netzes im vierten Teil beschrieben. Außerdem wird ein Vergleich aus dem Regelsystem gezogen, welches im Rahmen des Praxisprojekts entwickelt wurde.

Weiterhin beschreibt das fünfte Kapitel die Evaluation aus der Sammlung der Trainingsdaten und der Ergebnisse aus dem neuronalen Netz.

Abschließend findet im sechsten Teil ein Fazit statt, welches die Endergebnisse der Arbeit veranschaulicht und wie der Prototyp in Zukunft weiterentwickelt werden kann. Als Anhang findet man Teile aus dem Programmcode der Programme, die für die Bachelorarbeit geschrieben wurden.

Abstract

Diese Bachelorarbeit behandelt ein System bestehend aus Sensoren, welche mit einem ESP32 und Raspberry Pi einen Sofaprototypen bilden. Der Prototyp zur Sammlung der Sensorwerte und ein neuronales Netz sind die beiden Kernkomponenten. Der aktuelle Prototyp hat Ultrasonic-Sensoren angeschlossen, welche zur Messung der Rückenlehne benutzt werden. Dieser Sensor ist nicht geeignet, da dieser zu auffällig ist und nur im Radius vor der Rückenlehne messen kann. Daher sollte dieser durch einen FSR-Sensor ersetzt werden. Das neuronale Netz kann noch nicht zu 100% Vorhersagen treffen, kommt mit einer Vorhersage von 99,7% aber schon nah dran. Diese Vorhersage tritt nur bei dem Datensatz auf, der alle Datenpunkte beinhaltet. Es ist also wichtig, dass der Input für das neuronale Netz so nah wie möglich an der Realität ist.

1 Einleitung

Der Bereich IoT gewinnt eine immer breitere Aufmerksamkeit. Smart Homes sind ein Bereich und können Menschen helfen, welche Probleme haben, ihren Alltag ohne Hilfe zu bewältigen. Da es verschiedene Arten von Smart Homes gibt, ist es auch möglich ein Smart Home für die Gesundheit von Menschen anzupassen. Dadurch können diese Menschen als Patienten trotzdem weiter Zuhause leben und sind nicht auf Dauerhafte Hilfe von anderen Menschen angewiesen. Das Beispiel Health Smart Home (Rialle u. a., 2002) beschreibt diese Art eines Smart Homes. Die verschiedenen Smart Homes haben mit der Zeit viele verschiedene Ideen zur Weiterentwicklung hervorgebracht. So werden mittlerweile auch Möbel zu smarten Geräten entwickelt.

Da Steuerungen für ein Smart Home weitestgehend über Tablets, Smartphones oder ähnlichen Geräten, genutzt werden, müssen die Personen so ein Gerät immer in der Nähe haben. Diese Arbeit diskutiert eine Steuerung über ein Sofa. Das zeigt, wie Sitzmöbel als smarte Steuerungen in ein Smart Home eingebunden werden. Dadurch braucht der Nutzer kein Tablet oder ähnliches Gerät zur Steuerung. Für Personen die ihren Alltag ohne Hilfe nicht mehr bewältigen können, ist dies eine Möglichkeit passiv die Automatisierungen im Smart Home auszuführen.

1.1 Motivation und Relevanz zum Thema Smart Homes

Im der vorliegenden Arbeit wird die Einbindung von Möbelstücken in ein Smart Home diskutiert. Als Beispiel dient ein Sofa, welches Interaktionen über ein neuronales Netz verwaltet. Die Motivation ist die Gewährleistung von Selbstständigkeit, Sicherheit und damit Unabhängigkeit für Personen, die körperlich oder geistig eingeschränkt sind. (Ramlee u. a., 2012) zeigt einen Prototypen für solche Menschen.

Mit dieser Arbeit zeigt der Autor die steigende Relevanz von Smart Homes. Es wird mit der Entwicklung einer Steuerung für smarte Geräte dargestellt. Darüber veranschaulicht der Autor wie Personen mit körperlichen oder geistigen Einschränkungen, sowie älteren Menschen mit dieser Komponenten für ein Smart Home geholfen wird. (Demeure u. a., 2014) beschreibt die Relevanz von End-User-Development basierten Smart Homes. Zudem spielt das Strom-Management eine Rolle in Smart Homes und steuert durch die Automatisierung die smarten Geräte, dass Strom von Geräten einspart, welche nicht benutzt werden. Ein Energy Management System wie in (Al-Ali u. a., 2017) ist eine Umsetzung zur Stromeinsparung.

1.2 Idee zur Umsetzung vom Regelsystem zum neuronalen Netz

Die Idee geht aus dem Praxisprojekt hervor. (Schröder, 2019) Dort wurde ein Prototyp entwickelt, welcher in der Bachelorarbeit weiter entwickelt wird. Ein Regelsystem verwaltet die Interaktionen von Personen auf dem Sofa. Die Person im Use Case löst Regeln aus um smarte Geräte zu steuern. Damit dies funktioniert, muss der Nutzer für die entsprechenden Geräte eine Regel auslösen. Dies ist jedoch nur für eine endliche Zahl an Regeln möglich. Um mehr Interaktionen zu verwalten, ersetzt eine Machine Learning Methode das Regelsystem. Für die Bachelorarbeit wird ein neuronales Netz entwickelt, um dies zu veranschaulichen.

Das macht die Verwaltung der Interaktionen dynamischer, damit die Nutzer das Sofa so benutzen wie sie es aus dem Alltag gewohnt sind. So wird gezeigt, wie Möbel in ein Smart Home eingebunden werden. Des weiteren veranschaulicht der Autor in dieser Arbeit den Unterschied zwischen einem statischen Regelsystem und einem neuronalen Netz.

1.3 Personen an die das System gerichtet ist

Vorzugsweise sollen Personen angesprochen werden, welche ihren Alltag nicht ohne Hilfe bewältigen können. Die Steuerung hilft diesen Menschen dabei, die Automatisierung der smarten Geräte im Haus zu steuern. Dies soll ihnen zur weiteren Selbstständigkeit im Alltag helfen. Außerdem kann dadurch das Smartphone oder ähnliche Geräte ersetzt werden. Dadurch müssen diese Personen es nicht dauerhaft bei sich tragen. Zudem bleibt ihnen mit dieser Verwaltung ihrer Interaktionen mehr Zeit, um sich um andere Dinge zu kümmern. Die gleiche Hilfe bekommen dadurch ältere Menschen, da ihnen damit beispielsweise der Gang zum Lichtschalter erspart wird. Sollten sie dies vergessen, kann das neuronale Netz ihre Interaktionen mit dem Sofa erkennen und schaltet das Licht ein.

Neben diesen Personengruppen sind auch Personen mit Smart Homes angesprochen, die keine weitere Hilfe benötigen. Für diese Menschen ist der Punkt der Zeitersparnis ein wichtiger Faktor, da diese Nutzer sich das entsperren und herunterladen von Apps auf dem Smartphone oder ähnlichen Geräten ersparen.

1.4 Ziel der Bachelorarbeit

Ziel dieser Arbeit ist eine Steuerung zur Verwaltung von Interaktionen mit einem Sofa zu entwickeln. Dazu wird ein Prototyp entwickelt, welches Sensorwerte erfasst und an ein neuronales Netz sendet. Probanden Testen dieses System und geben dadurch Verbesserungen an. Daraus soll dann identifiziert werden, welche Sensoren ausgetauscht werden müssen.

1 Einleitung

Das neuronale Netz klassifiziert die Interaktionen der Probanden und wird über die Sensordaten trainiert und getestet. Die Umgebung ist ein gleichbleibender Raum, in dem ein 3 Personen Sofa steht. Der Prototyp aus dem Praxisprojekt ist die Basis für die Verbesserungen in der Bachelorarbeit. Die empirische Methode und Verbesserungen sollen die Umsetzbarkeit zeigen.

In den Anfängen dieser Arbeit wurde ein neuronales Netz entwickelt, welches einfache Interaktionen erkennt. Dies erweist sich jedoch nicht als sinnvoll, da diese Interaktionen nicht die Realität wiederspiegeln haben.

2 Grundlagen aus dem IoT für das Thema Smart Home Interior

2.1 Smart Home Aufbau und Funktion

Ein Smart Home ist ein System, welches automatisch smarte Geräte im Haus steuert. Die smarten Geräte können zusätzlich durch den Nutzer im Haus gesteuert werden. Ein smartes Gerät ist smart, wenn es mit einem Computer ausgestattet ist und damit die Funktionalität des Gerätes erweitert. Dies ist durch verschiedene Kommunikationsmethoden möglich. Diese Kommunikation kann z.B. durch Bluetooth, WiFi oder auch Infrarot durch den Nutzer und der Smart Home Automatisierung eintreten. Für den Prototypen der Bachelorarbeit wird ein Wireless System verwendet. Dies bedeutet, dass kein Kabel für die Kommunikation nötig ist. Die Verwaltung der Kommunikation der sendenden und empfangenden Geräte führt das MQTT-Protokoll durch. Kapitel 2.5 erläutert den Aufbau und die Funktionen von MQTT.

Weiterhin ist es möglich durch die Automatisierung bestimmte Interaktionen im Haus nicht mehr manuell vom Bewohner im Haus ausführen zu müssen. Für jeden Raum wird in der Regel eine unterschiedliche Automatisierung erstellt, da zu unterschiedlichen Zeiten andere smarte Geräte im Raum benutzt werden. Dadurch sorgt das Smart Home für Einsparungen im Haus.

Ein Beispiel für eine Einsparung ist der Stromverbrauch. Dabei passiert es oft, dass Strom im Haus verbraucht wird. Dieser kommt von Geräten, die nicht durchgehend Strom brauchen. Ein Smart Home kann die smarten Geräte so verwalten, dass sie nur dann Strom verbrauchen, wenn sie auch wirklich vom Nutzer aktiv verwendet werden. Sollte der Nutzer ein Gerät nicht mehr brauchen, kann das Smart Home den Stromkreis automatisch unter festgelegten Bedingungen unterbrechen. Zusätzlich können damit gefährliche Situationen vermieden werden, wie z.B. ein Kurzschluss. (Sripan u. a., 2012)

Es ist auch wichtig hervorzuheben, dass an Smart Homes bestimmte Anforderungen gestellt sind, damit Kriterien wie Sicherheit, Komfort oder Mobilität bei Älteren und Menschen mit Behinderung gewährt sind. Diese Anforderungen für die genannten Personen werden in (Das u. a., 2015) dargestellt und ein Prototyp veranschaulicht eine mögliche Lösung, um diese Anforderungen zu erfüllen. Aus diesen Erkenntnissen ist es möglich, verschiedene Smart Home Systeme zu entwickeln. Diese können speziell an die Nutzer angepasst werden. So haben Personen, die ohne Hilfe ihren Alltag nicht mehr bewältigen können oder Personen, die medizinische Unterstützung brauchen, die Möglichkeit, trotzdem selbstständig zu leben. Außerdem bringt die Erweiterung zu einem Smart Home den positiven Aspekt der Zeiteinsparung mit sich. Durch die verschiedenen positiven Gründe steigt auch die Relevanz der Smart Homes immer weiter

an. Besonders die Einbindung von Machine Learning Methoden hebt die Relevanz weiter an.

2.2 Relevanz neuronaler Netze in Smart Homes

Ein neuronales Netz ist in der Lage das Verhalten von Menschen zu erkennen, um so in Smart Homes unter anderem das Strom Management zu verbessern. Anhand von Sensoren wird gemessen, wie viele Personen sich im Raum befinden und was sie wirklich an Strom brauchen. Anhand der Sensorwerte und dem Verhalten der Menschen entscheidet das neuronale Netz, welches Gerät ausgeschaltet werden kann. (Badlani u. Bhanot, 2011) Aus dieser wissenschaftlichen Arbeit mit dem genannten Beispiel zum Strom Management ist zu erkennen, dass ein neuronales Netz durch Interaktionen mit Sensoren in der Lage ist, die Steuerung der smarten Geräte zu verwalten. Deshalb ist es für den Autor wichtig, sich mit der Verbindung zwischen neuronalen Netzen und Smart Homes zu beschäftigen.

Weiterhin ist besonders beachtenswert, dass Smart Homes im Zusammenhang mit neuronalen Netzen auch beeinträchtigten Personen Unterstützung bieten. Diese sollen dadurch weiterhin selbstständig leben können, wie die fremde wissenschaftliche Arbeit (Hussein u. a., 2014) darstellt.

Anhand dieser wissenschaftlichen Arbeiten werden zudem weitere Beispiele gezeigt, wie man neuronale Netze in Smart Homes einsetzen kann. Der Autor will damit hervorheben, dass sich die Umsetzung des Prototypen in ein Smart Home integrieren lässt. Die verschiedenen Systeme, die für ein Smart Home entwickelt werden und von einem neuronalen Netz verwaltet werden, nutzen oft mehrere unterschiedliche neuronale Netze. Daraus stellt sich die Frage, ob für ein Smart Home ein oder mehrere neuronale Netze von Vorteil sind. Dies lässt sich anhand der wissenschaftlichen Arbeiten, die in diesem Kapitel ausgeführt wurden und dem Prototypen der Bachelorarbeit beantworten. Deshalb ist es besser für verschiedene Räume unterschiedliche neuronale Netze zu nutzen, da diese unterschiedliche Steuerungen im Haus damit übernehmen und die Aufgaben, damit übersichtlicher bleiben. Dies zeigt, dass neuronale Netze in Smart Homes eine große Rolle übernehmen. So kann dadurch auch z.B. die Möglichkeit Möbel in einem Smart Home zu digitalisieren in Betracht gezogen werden.

2.3 Smart Home Interior

Mit diesem Kapitel erläutert der Autor eines der beiden Hauptgebiete in dieser Bachelorarbeit. Es wird hauptsächlich die Frage geklärt, welche Rolle Möbel in einem Smart Home spielen, wenn man diese zu smarten Geräten weiterentwickelt. Zusätzlich werden verschiedene Möglichkeiten gezeigt, wie dies umgesetzt werden kann.

2.3.1 Die Funktion der Innenausstattung im Smart Home

Das Teilgebiet Smart Home Interior zeigt die Digitalisierung der Innenausstattung in einem Smart Home. So wird veranschaulicht, dass die Möbel im Haus oder der Wohnung als smarte Innenausstattung genutzt werden können. Um smarte Umgebungen zu

erschaffen, werden die bereits bestehenden Möbel zu smarten Geräten umgewandelt. Damit könne smarte Geräte die ein Smart Home definieren, bestimmte Aufgaben an Möbel abgeben oder ihnen weitere Informationen zur Automatisierung mitteilen. Dadurch entstehen weitere Steuerungen, die Smartphones oder andere Geräte ersetzen. Sitzmöbel werden z.B. als Steuerung anderer smarter Möbel oder Geräte eingesetzt. Der Sofaprootyp soll dies darstellen und zeigen, wie es die Aufgaben eines Smartphones oder ähnlichen Geräte ersetzt. Außerdem ist es möglich, für die Möbel eine zusätzliche Automatisierung zu entwickeln und in das Smart Home mit zu integrieren. Smarte Möbel dienen zur passiven Steuerung von smarten Geräten oder Möbelstücken ohne, dass der Nutzer bei der Interaktion selbst etwas davon äußerlich an der Innenausstattung mitbekommt. Der Nutzer muss nicht mehr aktiv ein Tablet oder ähnliches Gerät entsperren, die App raussuchen und darüber steuern, sondern braucht nur noch eine aktive Interaktion mit den Möbeln und steuert so die smarten Geräte. Dadurch hat er einen geringeren Einfluss auf die individuelle Steuerung der Geräte. So spart der Nutzer aber wiederum Zeit. Da die Steuerung durch die Interaktionen verwaltet werden muss, zeigt (Soares u. a., 2010) wie es möglich ist, dass ein Möbelstück mit einem neuronalen Netz verknüpft werden kann.

2.4 Neuronale Netze zur Klassifizierung

Das folgende Kapitel befasst sich mit Grundlagen zu neuronalen Netzen. Umfasst wird dabei der Aufbau, wie die Berechnungen auf den Layern stattfinden, wie klassifiziert wird und Allgemeinbegriffe werden geklärt. Da das Thema zu neuronalen Netzen sehr breit ist, wird hier nur das erläutert, was der Autor zur Umsetzung und Evaluation in den späteren Kapiteln benutzt. So sollen später benutzte Fachbegriffe und Methoden nicht in jedem Kapitel neu erklärt werden.

2.4.1 Aufbau eines neuronalen Netzes

Das neuronale Netz ist in Schichten, auch Layer L genannt, aufgebaut. Es besteht aus einem Input-, Hidden- und Output-Layer. Die Zahl der Hidden-Layer ist immer $L-2$. Der Input-Layer bekommt die Daten der Außenwelt in Zahlenwerten beschrieben. Jeder einzelne Wert wird an eine Input-Unit im Layer übergeben. Daraufhin werden die Daten weiterverarbeitet und an die Hidden-Units in den Hidden-Layern gesendet. Es können mehrere Hidden-Layer je nach Aufgabe des neuronalen Netzes benutzt werden. Von dort aus werden diese Werte mit den Gewichtungen zusammen gerechnet und an eine Aktivierungsfunktion übergeben. Die nächsten Schritte werden in den folgenden Kapiteln weiter erläutert. Nach den Hidden-Layern kommt als letztes der Output-Layer mit den Output-Units. Der Output-Layer gibt die Zahlenwerte aus, die die Vorhersage des neuronalen Netzes als Ergebnisse in Form von Zahlenwerten liefern. Für diesen in der Bachelorarbeit verwendeten Prototypen ist im neuronalen Netz jeder Layer mit dem vorherigen Layer verbunden.

2.4.2 Initialisierung der Layer und Inputwerte

Zu Beginn, wie schon erwähnt, liest das neuronale Netz die Inputwerte in den Input-layer ein. Zusätzlich sollte nach dem Input-Layer pro Layer auch noch der Bias-Wert als weitere Unit im Layer hinzugefügt werden, da sonst wie z.B. bei der Linearen-Aktivierungsfunktion der Achsenabschnitt nicht geändert wird und die Funktion damit nicht so stark verschoben werden kann, wie mit dem Bias-Wert.

Benutzt man Bibliotheken wie Keras, müssen die Aktivierungsfunktion und der Bias-Wert als optionale Parameter im Layer angegeben werden. Außerdem muss beim Dataset angegeben werden, wie viel Prozent der Datensätze als Testdaten verwendet werden sollen. Diese Parameter sind vom Autor schon in der Initialisierung angegeben, da dies bei der Ausführung des neuronalen Netzes schon angegeben sein muss.

Da für das Netz eigene Datensätze genutzt werden, werden diese noch normalisiert, um die Daten besser an das Netz anpassen zu können und das globale Minimum der Fehlerrate besser zu erreichen.

2.4.3 Feedforward und Backpropagation

Der Feedforward und die Backpropagation sind die Hauptausführungen des neuronalen Netzes. Im Feedforward Schritt werden aus den Inputwerten die Vorhersagen getroffen. In der Backpropagation wird die Fehlerrate dieser Ergebnisse angepasst, sodass die Vorhersagen im Endergebnis so nah wie möglich an den Soll-Werten liegen.

Berechnung der Inputwerte

Zwischen den einzelnen Layern sind Gewichtungen $x_1 \dots x_n$, welche willkürlich zu Beginn gewählt werden. Diese werden mit den Eingabewerten $a_1 \dots a_n$ multipliziert und als Summe in der Funktion h ausgerechnet. Abbildung 2.1 zeigt ein Perzeptron, welches die Verbindung zwischen den Input- und Output-Units einzeln darstellt. Ein Perzeptron ist das einfachste neuronale Netz mit nur einer Output-Unit. Die vorherigen Layer können jedoch mehr Units besitzen.

$$h(x) = \sum_{i=1}^n a_i x_i = (a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_{n-1} \cdot x_{n-1} + a_n \cdot x_n)$$

a_n : Inputwerte aus der Realwelt

x_n : Gewichte

n : Obergrenze der Inputwerte und Gewichte

In der Output-Unit rechnet die Funktion h das Ergebnis der Gewichtungen und Inputwerte aus und gibt diese an die Aktivierungsfunktion z innerhalb der Unit weiter. Eine Erläuterung für die Aktivierungsfunktion, welche in dem neuronalen Netz für diese Bachelorarbeit benutzt wird, findet in Kapitel 2.4.3 statt. Damit ist der Feedforward Schritt abgeschlossen. Danach berechnet die Cost-Funktion C , die im Fall des hier benutzten neuronalen Netzes der *mean squared error* ist, die Fehlerrate des Output-Layers. Die folgende Funktion zeigt diesen Schritt der Berechnung. Da der

Output-Layer ein Vektor ist und mehrere Ergebnisse präsentiert werden, müssen alle Werte entsprechend in der Cost-Funktion als Summe ausgerechnet werden.

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

Y_i : Ergebnis aus dem Feedforward Schritt

\hat{Y}_i : Soll-Wert

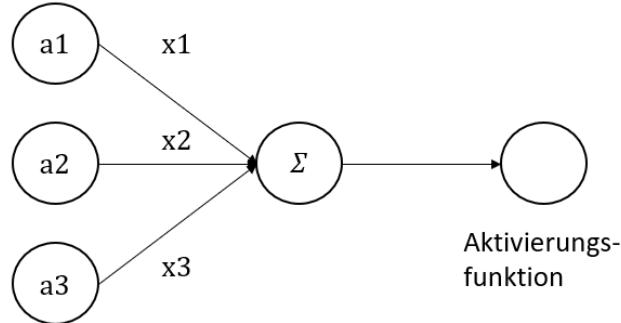


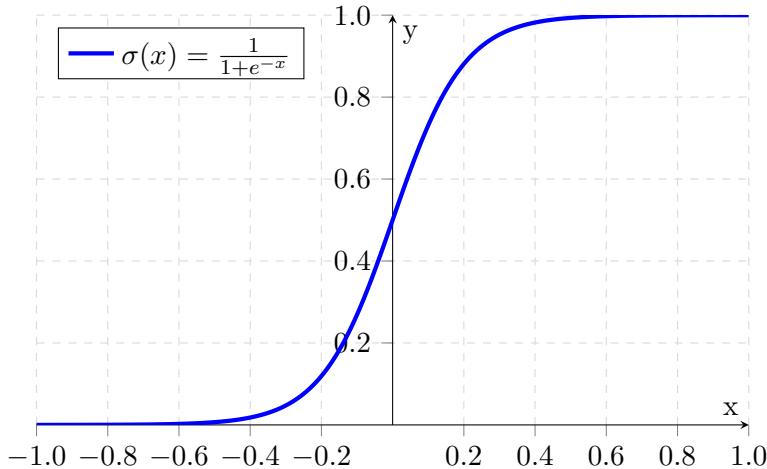
Abbildung 2.1: Darstellung eines Perzeptrons

Um die Inputwerte so einfach wie möglich darzustellen, wird eine $m \times n$ -Matrix verwendet. Die folgende Matrix soll ein allgemeines Beispiel dafür zeigen. Jede Zeile dieser Matrix steht für einen Datensatz und der erste Wert steht für den Bias-Wert.

$$m = \begin{pmatrix} a_0 & a_1 & \dots & \dots & a_n \\ b_0 & b_1 & \dots & \dots & b_n \\ \vdots & & & & \vdots \end{pmatrix}$$

Die sigmoid-Aktivitätsfunktion

Die sigmoid-Funktion ist die Aktivitätsfunktion der Layer, die im neuronale Netz aufgerufen wird. Da die Ergebnisse zwischen null und eins liegen, eignet sich diese Funktion für die Soll-Ergebnisse. (Wender u. Rey, 2011) Wichtig ist zu erwähnen, dass die Ergebnisse nicht den Wert null oder eins erreichen, sondern nur unter eins und über null liegen. Außerdem kann durch die möglichen Ergebnisse auch ein Intervall gewählt werden, bei dem die Klassifizierungen korrekt sind, auch wenn die Ergebnisse nicht exakt die gleichen sind. Damit ist der Raum der Ergebnismöglichkeiten größer und kann erweitert werden, sollten weitere Units im Output-Layer zu einem späteren Zeitpunkt dazu genommen werden.



Backpropagation

Bei der Backpropagation läuft das neuronale Netz nochmal rückwärts ab, um die Gewichte so zu verändern, dass die Fehlerrate auf ein lokales oder globales Minimum reduziert wird. Man muss also im Netz rückwärts gehen und die Gewichte und Biases anpassen. Die Anpassung findet über das Gradientenabstiegsverfahren statt. (Wender u. Rey, 2011) beschreibt das Gradientenabstiegsverfahren. Wie schon in Kapitel 2.4.3 genannt, verwendet das neuronale Netz die sigmoid-Funktion. Da hier aber die Backpropagation stattfindet, muss mit der partiellen Ableitung weiter gerechnet werden. Jede Gewichtung und jedes Bias der partiellen Ableitungen müssen im Gradienten Vektor gespeichert werden. Abschließen werden diese mit der Lernrate und der partiellen Ableitungen verrechnet. In Kapitel 4.2 wird die Umsetzung in der Programmiersprache Python dargestellt.

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Die gezeigte Funktion ist die Ableitung der sigmoid-Funktion. Mit der Backpropagation wird die Cost-Funktion minimiert und es werden die Gewichte und der Bias-Wert in jedem Layer des neuronalen Netzes vom Output-Layer aus rückwärts angepasst.

2.4.4 Ausgabe der Ergebnisse als Klassifizierung

Das neuronale Netz des Prototypen arbeitet mit der Vorhersage, die Ergebnisse wie in Abbildung 2.2 einzuteilen. Alle Punkte, die farblich eingeteilt sind, stehen für eine mögliche Klassifizierung. Dies bedeutet, dass ein Ergebnis aus dem neuronalen Netz einer dieser Klassen ist und die anderen Klassen einen Wert haben, welche nicht dem gewünschten Ergebnis entspricht. Also damit nicht die Klasse ist, in welches man das Ergebnis einteilt.

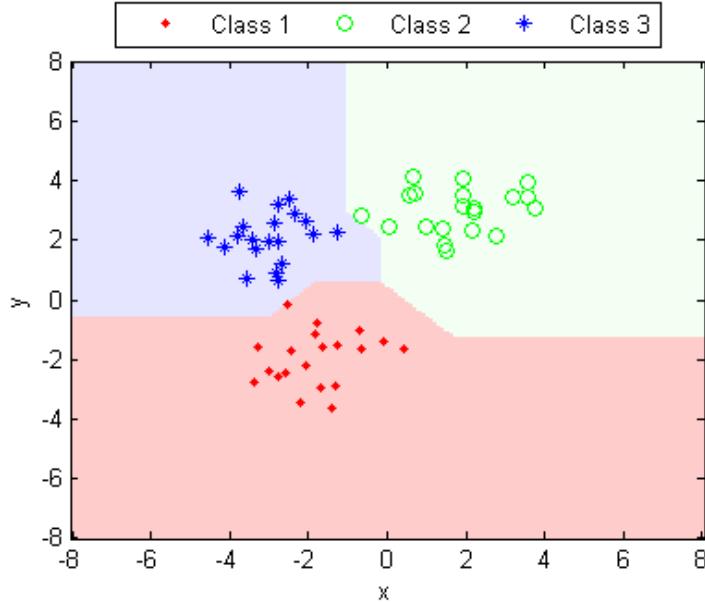


Abbildung 2.2: Beispiel einer Klassifizierung (Yu)

In dieser Arbeit geht es darum, dass die Ergebnisse des neuronalen Netzes in Gruppen eingeteilt werden. Die visuelle Klassifizierung in der Abbildung 2.2 ist nur ein Beispiel und entspricht nicht dem neuronalen Netz aus dieser Bachelorarbeit. Dennoch entsprechen diese drei Klasseneinteilungen den Ergebnissen dem neuronalen Netz des Prototypen. Weiterhin ist relevant, dass diese Darstellung auch in Form eines Vektors erfolgen kann. Dies funktioniert aber nur bei einem Output-Layer mit mindestens zwei Units. Der folgende Vektor v zeigt eine weitere Möglichkeit, um die klassifizierten Ergebnisse darzustellen.

$$v = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$$

2.4.5 Die Trainingsphase und Testphase mit Einlesen von Testdaten

Bei einem neuronalen Netz gibt es zwei Phasen - die Trainings- und die Testphase. Diese werden durch ein Dataset repräsentiert, welches Trainingsdaten und Testdaten beinhaltet. Während der Trainingsphase werden ein Teil dieses Datasets in das neuronale Netz als Inputwerte eingegeben und über Supervised Learning trainiert. Das Supervised Learning beschreibt eine Lernmethode, bei der die Trainingsdaten mit den Soll-Ergebnissen trainieren. Es wird schon beim Training das richtige Ergebnis mitgeteilt. Da die Gewichte, deren Aufgabe in Kapitel 2.4.3 näher erläutert wird, zu Beginn

noch willkürlich gewählt sind, werden diese in der Trainingsphase angepasst. Sind die Gewichte angepasst, erfolgt die Testphase. In der Testphase wird getestet, wie gut das neuronale Netz voraussagen kann, was das richtige Ergebnis ist. Es wird nicht mehr das Ergebnis wie beim Training vorgesagt. Ziel dieses Vorgangs ist danach weitere Echtzeitwerte in das neuronale Netz einzugeben und durch das Trainig und den Tests die richtige Klassifizierung vorherzusagen.

Das neuronale Netz muss so programmiert werden, dass es mit dem Dataset trainiert und getestet werden kann. Also klappt es nicht, wenn es z.B. eine Bildklassifizierung vorhersagen müsste, welches Vorhersagen beim Boston-Housing Dataset durchführt.

2.5 MQTT zur Datenübertragung

MQTT ist ein M2M-Protokoll, welches mit der Publish-Subscribe-Methode arbeitet. Mikrocontroller und Sensoren publishen dabei ihre Daten an den MQTT-Broker. Der Broker speichert diese Daten zwischen, bis Subscriber diese Nachrichten anfordern. Jeder Subscriber bekommt diese Nachrichten gleichzeitig je nach Quality of Service Stufe zugestellt. Diese Clients können ihre Daten wiederum auch publishen und wieder an neue Geräte die als Subscriber markiert sind, versenden. Als Clients sind intelligente Geräte gemeint, wie Mikrocontroller oder PCs. Ein Client ist Subscriber, Publisher oder beides zusammen. Die Anzahl der Clients, die aktiv als Subscriber oder Publisher dienen, ist nicht relevant. Wichtig ist dabei nur, dass der Broker funktioniert und verfügbar für die Clients ist. (Soni u. Makwana, 2017) Damit eignet sich dieses Protokoll für den IoT-Bereich besonders gut.

Wie schon gesagt, werden die Nachrichten je nach QoS Stufe versendet. Es gibt insgesamt drei Stufen des QoS. Diese werden in der Tabelle 2.1 dargestellt. Die Stufen dienen zur Sicherung der Nachrichtenübermittlung, auch wenn das Netz dabei gestört wird. So sollen die Nachrichten trotzdem weiterhin vermittelt werden.

Tabelle 2.1: Tabellarische Darstellung der QoS Level (Soni u. Makwana, 2017)

Stufen	Häufigkeit	Beschreibung
0	höchstens eine Nachricht	Es wird maximal eine Nachricht versendet, ohne eine Garantie, dass diese auch ankommt
1	mindestens eine Nachricht	Es ist möglich, dass eine Nachricht mehr als einmal versendet wird
2	genau eine Nachricht	Eine Nachricht wird mit einem Four-Way Handshake versendet

Für die Kommunikation ist ein TCP-Stack und das MQTT-Protokoll vorausgesetzt. Zusätzlich ist eine Verbindung zum MQTT-Broker erforderlich bei dem der Typ des Netzes aber unwichtig ist. Außerdem ist zu erwähnen, dass es nicht relevant ist, dass

sowohl Publisher als auch Subscriber gleichzeitig aktiv sein müssen. Der Broker kann die Nachrichten auch dann noch übermitteln, obwohl der Publisher nicht mehr aktiv ist.

Damit Nachrichten versendet werden, gibt es die sogenannten Topics. Bei diesen Topics ist es nicht wichtig, über welche Struktur die Nachricht aufgebaut ist. (Trojan, 2017) MQTT ist mit dieser Methode ein Protokoll, welches passend für diesen Prototypen ist, da es für das Senden von Sensordaten in einem Wireless Network ausgelegt ist.

2.6 Sensoren für die Interaktionen auf dem Sofa

Die Sensoren sind ein Hauptbestandteil des Prototyps. Diese sind mit Mikrocontrollern verbunden und ein Raspberry Pi sorgt für die Kommunikation zwischen den Geräten und den Mikrocontrollern. Neben der Erläuterung der genutzten Sensoren in dem aktuellen Prototyp wird hier auch zusätzlich der Mikrocontroller und Raspberry Pi beschrieben.

2.6.1 Erläuterung des Raspberry Pi 3 und des ESP32

Der Raspberry Pi 3 ist ein Einplatinencomputer, auf dem das Betriebssystem Raspbian installiert ist. Das Betriebssystem basiert auf Linux Debian und läuft in der aktuellen Version 10 - Buster. Neben den Möglichkeiten den Raspberry Pi für einfachere Anwendungen zu Verwenden, ist es auch möglich verschiedene Sensoren an dessen GPIO-Pins anzuschließen. Der Raspberry Pi 3 hat einen internen WiFi-Chip und kann damit in einem WLAN-Netzwerk eingebunden werden. Dies wird benötigt, da der Prototyp in einem lokalen Netzwerk ausgeführt wird.

Der ESP32 hingegen ist ein Mikrocontroller, welcher als Nachfolger vom ESP8266 weitere GPIO Pins besitzt, um mehr als einen Analogen Sensor anschließen zu können. Dies ist wichtig, da für den Prototypen mehrere Analog-Sensoren angeschlossen werden müssen. Ein ESP32 hat ein Live Betriebssystem und kann nur ein Programm ausführen, welches vorher auf den ESP geladen werden muss.

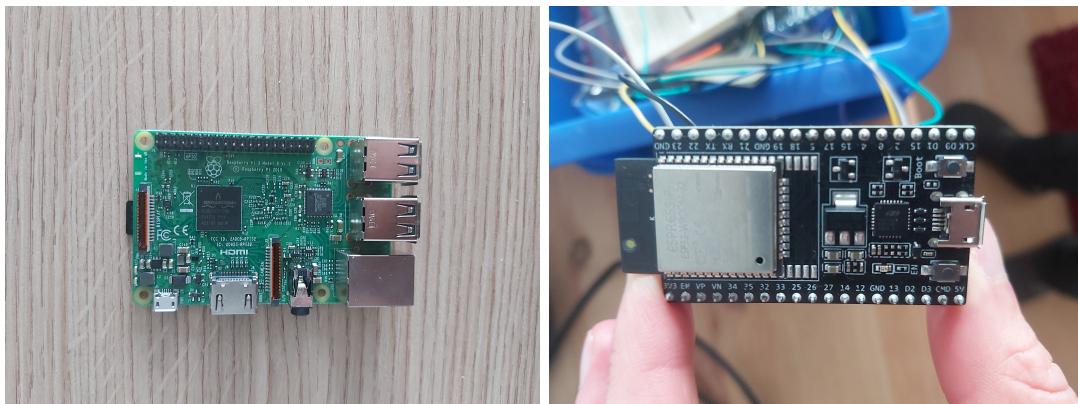


Abbildung 2.3: Die linke Abbildung zeigt den Raspberry Pi und die Rechte den ESP32

2.6.2 Sensoren, die für den Prototypen verwendet werden

Für den aktuellen Prototypen werden Ultrasonic-Sensoren und Force Sensitive Resistors verwendet. Diese reichen aus, um Sitz- und Liegepositionen der Nutzer zu erkennen. Dennoch zeigt Kapitel 5.1 die Vor- und Nachteile der Sensoren in der Realanwendung. (De Simone u. a., 2018) beschreibt die Funktionen des Ultrasonic-Sensors in der fremden wissenschaftlichen Arbeit. Wichtig ist dabei herauszulesen, dass der Sensor Distanzen zwischen 1 cm und 4 Meter messen kann. So ist dieser Sensor eine gute Möglichkeit, um auch Personen zu erkennen die sich anlehnen. Das Problem ist allerdings, dass beim Verwenden dieses Sensors im Prototypen dieser im Sofa verbaut sein muss. Dadurch können sich Personen so anlehnen, ohne den Sensor physisch zu treffen. Zudem ist der Radius, in dem die Wellen gesendet werden, zu gering, da die Sofalehnen mehr Platz haben als der Sensor messen kann.

Neben dem Ultrasonic-Sensor misst der FSR-Sensor anhand des elektrischen Widerstands den Druck, welcher zwischen 100 Gramm und 10 kg beträgt. Bei allen Gewichten über 10 kg zeigt der Sensor immer den höchsten Wert. (Flórez u. Velasquez, 2010) Der Sensor ist aus zwei Polymerschichten gebaut. Eine leitende Oberfläche und die zweite, welche an die erste Schicht mit Elektroden gebunden ist. (Hollinger u. Wanderley, 2006) Diese Sensoren werden verwendet, um zu erkennen, wann die Personen Druck auf sie ausüben indem sie ihn mit ihrem Gewicht auslösen. Als FSR-Sensoren werden die interlink Sensoren eingesetzt. Abbildung 2.4 zeigt die Sensoren, welche am ESP32 verbaut sind. Die beiden gezeigten FSR-Sensoren sind unterschiedlich groß, geben aber die gleichen Werte aus.



Abbildung 2.4: Links ist der Ultrasonic-Sensor, in der Mitte der Große und rechts der kleine FSR

2.7 Prototyp aus dem Praxisprojekt

Als letzten Punkt geht der Autor nochmal auf den Prototyp im Praxisprojekt ein. Der Prototyp, welcher die Interaktionen als Regelsystem mit sieben Regeln verwaltet, ist ein Projekt, welches im Rahmen des Praxisprojekts entwickelt wurde. Ein ESP32 misst die Sensordaten von zwei FSR-Sensoren und einem Ultrasonic-Sensor. Jeder Sensor wird als MQTT-Client deklariert und sendet Daten in Form von Codes, je nachdem mit welchen Sensoren der Nutzer interagiert. Zusätzlich messen zwei ESP8266 die Daten von jeweils einem Flex-Sensor, welche den Widerstand durch ihre Biegung messen. (Saggio u. a., 2015) Dies erweist sich aber als Nachteil, da die Biegung erst ab 45 Grad gemessen wird. Das kommt beim Test als Ergebnis heraus, durch Biegungen ohne auf

ihm zu liegen und beim Testen des kompletten Prototypen.

Ein Raspberry Pi 2 führt einen MQTT-Broker Service aus. Er kann Daten von den ESP-Sensoren empfangen, die über die Publish-Methode die Codes versenden. Auf dem Raspberry Pi 2 läuft zusätzlich ein Python-Programm, welches die Codes als Subscriber empfängt und im Regelsystem verwaltet. Die Regeln werden als Codes neu an den MQTT Broker gepublished. Daraufhin können die Codes dann von jedem beliebigen Subscriber empfangen werden. (Schröder, 2019)

Was bei der Nutzung des Regelsystems aber nicht beachtet wurde, sind die Ausreißer der FSR-Sensorwerte. Mit dem Regelsystem zielt man darauf ab, dass die FSR-Sensoren immer wenn sie Nichtbenutzt werden, den Wert 0 haben und wenn sie besetzt sind, den Wert 1024. Während des Tests sind diese Ausreißer immer wieder aufgefallen. Da wird der Fall ausgelöst, dass der FSR-Sensor eine Interaktion erkennt, obwohl keine stattfindet.

3 Architektur und Aufbau des Prototyps

In Kapitel 2 werden die Grundlagen zum MQTT-Protokoll und den Sensoren erklärt. Auf Basis dieser Themen wird die Architektur und der Aufbau des Prototyps im dritten Teil der Bachelorarbeit beschrieben. Des weiteren wird erklärt, welche Änderungen in der Architektur vom Prototypen des Praxisprojekts vorgenommen werden. Außerdem erklärt der Autor die Änderungen, welche beim Prototypen in der Bachelorarbeit auftreten.

3.1 Anforderungen an die Architektur

Der Einsatz eines Möbelstücks wie Sitzmöglichkeiten zur Steuerung von Interaktionen ist typischerweise an eine Umgebung gebunden. Dies bedeutet, dass bspw. ein Sofa also einen festen Platz im Raum hat und nicht bewegt wird. Damit der Nutzer den vollen Funktionsumfang dieser Steuerung nutzen kann, ist es wichtig, dass dabei die herkömmlichen Positionen auf dem Sofa als Klassifizierung angewendet werden. So muss der Anwender sich nicht an neue Positionen gewöhnen, sondern kann mit dem Sofa so interagieren, wie er es im Alltag auch ohne Steuerung würde.

Daraus wird abgeleitet, dass die Hardware-Architektur als finale Version nicht sichtbar sein darf und der Nutzer nichts von dieser mitbekommt. Zusätzlich ist noch wichtig zu erwähnen, dass durch seine Interaktionen erkannt werden muss, welche smarten Geräte er steuern möchte. Damit das neuronale Netz entsprechend steuert, welche smarten Geräte der Nutzer mit seinen Interaktionen ausführt, muss dies über vorherige Einrichtung geschehen oder einer automatischen Erkennung, welche erkennt was der Nutzer in bestimmten Interaktionen häufig nutzt. Dies ist aber nicht Teil dieser Bachelorarbeit und wird damit auch nicht weiter behandelt. Wenn diese Interaktionen erfolgreich klassifiziert werden, muss das neuronale Netz mit in das System eingebunden werden.

In der Abbildung 3.1 wird der Aufbau des Systems gezeigt und wie dieser Ablauf stattfinden soll. Der Prototyp umfasst die Aufnahme der Interaktionen und deren Verarbeitung von drei Positionen im neuronalen Netz. Kapitel 4.2 beschreibt die Umsetzung dieses neuronalen Netzes.

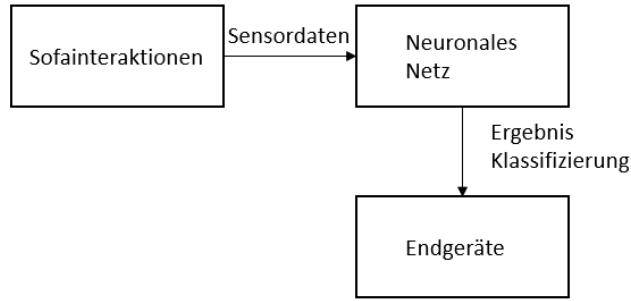


Abbildung 3.1: Visuelle Darstellung zum Aufbau des Systems

3.2 Architektur des Sofaprototypen aus dem Praxisprojekt

Der Prototyp ist ein im Rahmen des Praxisprojekts entwickeltes System, welches als verteilte Architektur aufgebaut ist. Während der Bachelorarbeit wird der Prototyp weiter ausgearbeitet und für die weiteren Versuche so optimiert, wie es in Kapitel 3.3 erläutert ist. In diesem Kapitel wird nur die Version aus dem Praxisprojekt zum Vergleich der aktuellen Version des Prototyps während der Bachelorarbeit erläutert.

Ein ESP32 misst die Sensordaten eines Ultrasonic-Sensors sowie eines FSR-Sensors. Zwei Flex-Sensoren sind separat an jeweils einem ESP8266 angeschlossen. So bietet der Prototyp nur die Möglichkeit für eine Person zu sitzen oder auf den Sitzflächen mit den Flex-Sensoren zu liegen. Diese Daten schicken sie an einen Raspberry Pi 2, welcher diese an ein Regelsystem weitergibt, dessen Regeln in 3.4 aufgeführt sind. Das Regelsystem ist in dieser Version des Prototypen die Hauptkomponenten, um die Interaktionen zu verwalten. Die verarbeiteten Daten, welche über *if-Abfragen* in Regeln umgewandelt sind, werden über den Broker an die smarten Geräte gesendet. In diesem Prototyp sind noch keine smarten Geräte integriert, da diese Einbindung kein primäres Ziel ist. Mit diesem Prototypen ist es möglich, dass Personen mit festen Interaktionen die smarten Geräte steuern ohne, dass der Nutzer Einfluss auf diesen Vorgang nimmt. Damit sind die Interaktionen davon abhängig, wie der Nutzer mit dem Sofa interagiert. Werden die Sensoren in einer anderen Art und Weise verwendet wie es nicht vom Regelsystem vorgesehen ist, so können die erwünschten smarten Geräte nicht gesteuert werden. Somit ist das Regelsystem ein erheblicher Nachteil für Personen, die das Sofa zum ersten Mal benutzen oder für Personen, welche körperlich oder geistig benachteiligt sind sowie Personen sehr hohen Alters.

Unter anderem wird aus diesem Grund auch das neuronale Netz entwickelt und ersetzt damit das Regelsystem. Zusätzlich werden die Flex-Sensoren nicht mehr für die Liegepositionen benötigt, da diese nicht dafür geeignet sind. Um die Liegeposition dennoch zu erkennen, übernehmen die Kombinationen der Drucksensoren auf den Sitzkissen diese Aufgabe. Es ist wichtig zu erwähnen, dass die Architektur aus dem Praxisprojekt so ausgelegt ist, dass nur eine Person auf einem Platz sitzen kann und die anderen Sitzkissen nur mit Flex-Sensoren verbaut sind. Auf den nicht besetzten Plätzen ist es jedoch nicht möglich, dass andere Personen dort zusätzlich Platz nehmen.

3.3 Die Anpassung der Architektur für das neuronale Netz

Dieses Kapitel beschreibt die Architektur der Bachelorarbeit und zeigt die Unterschiede aus Kapitel 3.2 zur aktuellen Architektur. Kapitel 3.7 beschreibt die Kommunikation mit MQTT, weshalb dies auch in diesem Kapitel nicht weiter erläutert wird.

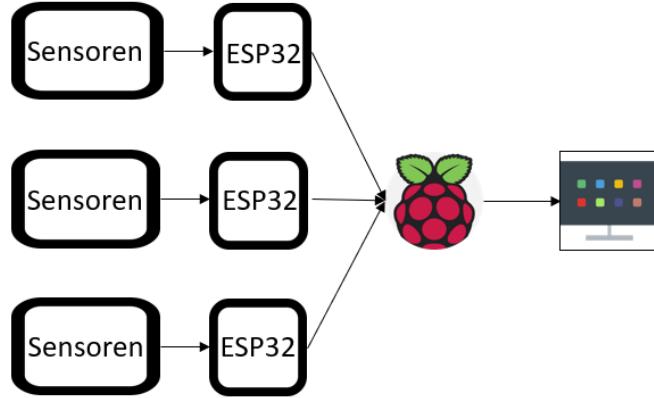


Abbildung 3.2: Abschließende Architektur mit einem Endgerät als Beispiel

Abbildung 3.2 zeigt die Architektur des überarbeiteten Prototypen. Als Sensoren werden Ultrasonic-Sensoren und Force Sensitive Resistors benutzt. Die Ultrasonic-Sensoren messen den Abstand um zu erkennen ob eine Person angelehnt ist. Die FSR-Sensoren messen anhand des Drucks ob eine Person sitzt, liegt oder mit den Armlehnen interagiert. Da dieser Prototyp für ein drei Personen Sofa ausgelegt ist, haben die beiden äußeren ESPs auch jeweils einen FSR für die Armlehnen angeschlossen. Der Raspberry Pi übernimmt die Aufgabe des MQTT-Brokers und der Aufnahme und Weitergabe der Sensorwerte. Der hier dargestellte Fernseher ist ein Beispiel für ein smartes Endgerät. Powerbanks sorgen für die Stromversorgung der Mikrocontroller und des Raspberry Pis. Alternativ ist auch das Anschließen eines Stecker für Steckdosen möglich. Als nächstes ist wichtig zu klären, ob der Prototyp als Cloud-Architektur oder lokales Netzwerk benutzt wird.

Hinsichtlich dieser Architektur reicht für das System ein lokales Netzwerk aus. Im Vergleich zu einer Cloud-Architektur ist es nicht möglich von außen auf das System zuzugreifen. Da die Steuerung sich nur innerhalb des Smart Homes befindet, reicht das lokale Netzwerk dafür aus. Eine Cloud-Architektur ergibt also nur Sinn, wenn Nutzer außerhalb des im Smart Homes auf die Geräte zugreifen wollen. Das kann der Nutzer aber nicht, da das Sofa sich immer innerhalb des Smart Homes befindet.

3.4 Regelsystem zur Steuerung der Interaktionen

Das neuronale Netz ersetzt in der aktuellen Version des Prototypen das Regelsystem. Das Regelsystem ist ein statisches System, welches feste Regeln beinhaltet. Dieses wird in diesem Abschnitt näher erläutert. Es sollen Vor- und Nachteile veranschaulicht werden und weshalb es durch das neuronale Netz ersetzt wird.

Das Regelsystem ist so ausgelegt, dass je nach Verwendung der Sensoren eine Regel bzw. ein Code erstellt wird. Der Nachteil dieses Systems ist der statische Aufbau. Wenn eine Regel ausgelöst wird, muss jeder Benutzer immer die gleichen Sensoren auslösen um eine Regel zu nutzen. So muss bei einer Erweiterung mit neuen Regeln der Python-Code immer angepasst werden. Das hat den Nachteil für Menschen, die das System nicht kennen und zum ersten Mal benutzen. Vor allem müssen sie den Code selbst erweitern.

Da sich jede Person anders hinsetzen kann, aber das gleiche Gerät steuern möchte, kann dies nicht vom Regelsystem erkannt werden. Daher hat der Autor die Architektur und damit auch die Weiterführung des Prototypen ersetzt. Die Tabelle 3.1 beschreibt dennoch noch einmal die Regeln und welche Kombination von Sensoren ausgelöst werden muss, damit man die Regeln an ein smartes Gerät sendet. Die Aufzählungen der Regeln sind auch gleichzeitig die Codes, die gepublished werden, um die smarten Geräte zu steuern.

Mit dem Regelsystem gibt es drei FSR-Sensoren und einen Ultrasonic-Sensor für die Sitzpositionen im Prototypen. Für die Liegeposition werden zwei Flex-Sensoren benutzt. Die Flex-Sensoren fallen mit dem einsetzen des neuronalen Netzes weg, da auf den entsprechenden Plätzen andere Sensoren verwendet werden. Dies passiert, weil die Sensoren eine Biegung von mindestens 45 Grad brauchen, damit sie erkennen, dass jemand auf ihnen liegt. Daher werden nun FSR-Sensoren für die Liegepositionen eingebaut. Diese erkennen, ob eine Person sitzt oder liegt. Somit müssen dann nicht mehr zwei Sensoren in die Sitzkissen eingebaut werden. Das positive an diesem Regelsystem ist die Erweiterbarkeit der Regeln für den Entwickler. Dabei wird im Programm nur die neue Regel mit einer *if-Abfrage* erweitert.

Tabelle 3.1: Aufzählung der Regeln des Regelsystems

Regeln	Beschreibung
1	Alle FSR-Sensoren und der Ultrasonic-Sensor messen Daten
2	Nur der FSR-Sensor auf dem Sitzkissen misst Daten und der Ultrasonic-Sensor
3	Beide FSR-Sensoren messen Daten
4	Der FSR-Sensor auf dem Sitzkissen misst Daten
5	Der FSR-Sensor auf dem Sitzkissen und die beiden Flex-Sensoren messen Daten
6	Beide Flex-Sensoren messen Sensordaten
7	Ein Flex-Sensor und der FSR-Sensor auf dem Sitzkissen messen Daten

3.5 Verarbeitung der Daten im ESP und Raspberry Pi

Kapitel 3.3 zeigt die Verwendung mehrerer ESP32 und einem Raspberry Pi in der Architektur. Die Mikrocontroller und der Einplatinencomputer sind die Hauptkomponenten zur Verarbeitung der Sensordaten. An einem ESP sind mindestens ein FSR-Sensor und maximal ein Ultrasonic-Sensor angeschlossen. Die Sensoren schicken ihre Sensorwerte an den entsprechenden ESP. Die Sensordaten werden als Binary-Werte vom ESP erfasst und an den MQTT-Broker gepublished. Das Python-Programm soll diese Daten auf dem Raspberry Pi als Subscriber empfangen und verarbeiten. Damit das Python-Programm die Sensordaten unterscheiden kann, werden die Sensordaten in einem Array gespeichert. Um unterscheiden zu können, welcher Wert zu welchem Sensor gehört, hat ein Sensor einen eigenen Index in einer Liste. Das Programm wird als Subscriber vom MQTT-Broker erfasst und schickt ihm die Sensorwerte. Die Daten werden in einer CSV-Datei gespeichert. Vorher müssen sie noch encoded werden, damit sie richtig dargestellt werden. Wenn dies nicht passiert, werden die Werte mit dem falschen Datentyp in der CSV-Datei gespeichert. Über eine Funktion aus der MQTT-Bibliothek kann zur besseren Darstellung der Binary-Wert zu Utf-8 encoded werden. Da das neuronale Netz Integer-Werte einliest, muss das Encoding stattfinden, denn die Binary-Werte sind in Python sonst einzelne Char-Datentypen. Dadurch würden die Daten nicht pro Zeile als eine Position gespeichert werden, sondern jeder Wert in einer neuen Zeile. Diese Liste speichert mit den Sensorwerten aktuell noch eine unbekannte Position des Nutzers, da diese Werte noch nicht verarbeitet wurden. Daher werden die Sensorwerte zum trainieren für das neuronale Netz in die CSV-Datei übergeben.

Sobald das neuronale Netz die Vorhersagen richtig angibt, besteht die Möglichkeit das Netz in das Python-Programm zu integrieren, welches aber nicht von Vorteil ist wie 4.6 zeigt. Zudem wird dann auch nicht mehr jeder Sensorwert in die CSV-Datei

übergeben, sondern die Sensorwerte werden einzeln an das neuronale Netz geschickt und dieses versucht die Positionen sofort zu klassifizieren.

3.6 Steuerungsarten eines Smart Homes

Diese Bachelorarbeit beschäftigt sich mit der Verwaltung von Interaktion zur Steuerung smarter Geräte durch Möbel. Da dies nur eine Möglichkeit zur Steuerung ist, diskutiert der Autor in diesem Kapitel Alternativen zu der Steuerung, welches Thema dieser Bachelorarbeit ist und deren Unterschiede zum aktuell entwickelten Prototypen.

Der aktuelle Prototyp, welcher im Praxisprojekt entwickelt wurde und in der Bachelorarbeit weiter entwickelt wird, interagiert passiv mit dem Nutzer. So muss der Nutzer nur seine gewohnten Positionen auf dem Sofa einnehmen und steuert damit die smarten Geräte. Dies spart Zeit, die der Nutzer normalerweise mit einem Tablet oder ähnlichem Gerät beötigen würde um das Smart Home zu steuern. Als weiteren Punkt sollte nicht unerwähnt bleiben, dass mit der Sofasteuerung Nutzungsmöglichkeiten mit den smarten Geräten entfallen. Dies ist ein Nachteil, welcher wiederum durch die zusätzliche Hinzuziehung weiterer Steuerungsmöglichkeiten gelöst werden kann. Das ist jedoch nicht sinnvoll, wenn der Aspekt der Zeitersparnis bestehen bleiben soll. Die folgende wissenschaftliche Arbeit (Piyare, 2013) erläutert die Umsetzung einer Steuerung in einem Smart Home über Tablet und Smartphone, welche mit Android umgesetzt wurde. Achtet man nicht auf Zeitersparnis und will die individuelle Nutzung der smarten Geräte beibehalten, haben Mobilgeräte und ähnliche Steuerungsgeräte den Vorteil, smarte Geräte in einem breiteren Spektrum zu nutzen. Dafür muss aber entsprechend auch mehr Zeit genutzt werden, da die Geräte oft erst entsperrt werden müssen und unterschiedliche Apps für die smarten Geräte heruntergeladen sein müssen. Also fällt mit dem Nutzen einer Sofasteuerung das Entsperren und die Benutzung verschiedener Apps weg. Zudem sind die Komponenten des Systems an Powerbanks oder direkt an einer Steckdose angeschlossen. So müssen diese Komponenten nicht so oft oder gar nicht wie ein Smartphone oder ähnliches Gerät aufgeladen werden.

3.7 Kommunikation der Sensoren mit dem neuronalen Netz

Wie in den vorherigen Kapiteln zur Architektur erwähnt, findet die Kommunikation im Prototypen mit MQTT statt. Hier erläutert der Autor dessen Umsetzung und Funktion im Prototypen.

Um MQTT-Mosquitto auf dem Raspberry Pi zu installieren, müssen dafür bestimmte Commands im Terminal eingegeben werden. Zudem muss die entsprechende Bibliothek in Python installiert werden, damit das Python-Programm mit dem Broker kommunizieren kann. Über

```
sudo apt install -y mosquitto mosquitto-clients
```

wird Mosquitto installiert und daraufhin wird Service mit

3 Architektur und Aufbau des Prototyps

`sudo systemctl enable mosquitto.service` aktiviert.

Je nach Betriebssystem und dessen Version weichen die Befehle voneinander ab oder es müssen weitere hinzugefügt werden.

Ist der Service gestartet, kann darüber kommuniziert werden. Zu Beginn schicken die ESP32 die Sensorwerte mit der Publish-Funktion als eine Byte-Nachricht zum MQTT-Broker. Der MQTT-Broker empfängt die Daten vom Publisher und speichert sie für die Subscriber zwischen. Alle Subscriber empfangen die Daten gleichzeitig und können sie unabhängig voneinander verarbeiten. Wie schon in Kapitel 2.5 erwähnt, ist es auch möglich, dass die Subscriber die Nachrichten auch empfangen können, wenn die Publisher nicht mit dem MQTT-Broker verbunden sind. Dies ist ein Vorteil, wenn ein Publisher ausfallen sollte, aber der Subscriber die Sensorwerte noch nicht rechtzeitig verarbeiten konnte. Als Subscriber dient in diesem Projekt das Python-Programm, welches diese Daten für das neuronale Netz empfängt. Das Programm befindet sich im ersten Anhang. Bevor das neuronale Netz in den Prototypen eingebunden wird, muss es aber korrekt klassifizieren. Damit das neuronale Netz dies trainieren kann, werden die im Prototyp kommunizierten Daten als CSV-Datei in das neuronale Netz eingelesen.

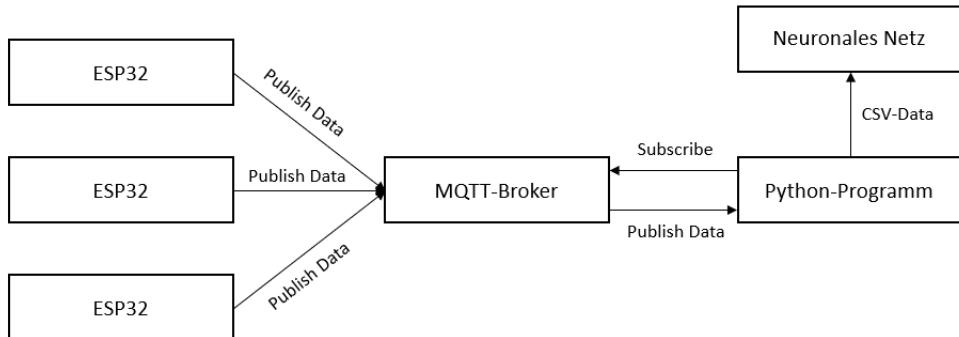


Abbildung 3.3: Die Kommunikationsarchitektur ohne Endgerät als Beispiel

Das Python-Programm, welches die Daten in einer CSV-Datei speichert, published die Ergebnisse des neuronalen Netzes. Diese Ergebnisse nutzen die smarten Geräte als Steuerung. Abbildung 3.3 zeigt die Kommunikationsarchitektur ohne die Anbindung an ein smartes Endgerät, da diese Kommunikation noch nicht hergestellt wird. Da die Ergebnisse am Ende gepublished werden, ist die Kommunikation offen, um mit smarten Geräten oder Möbeln zu kommunizieren. Sollte das neuronale Netz weiterhin als eigene Komponenten agieren, so müssen die Ergebnisse auch dementsprechend darüber an den Broker mitgeteilt werden und nicht über das Python-Programm, welches die Sensordaten erfasst. Dies passiert erst, sobald das neuronale Netz vollständig korrekt klassifizieren kann.

3.8 Gegenüberstellung eines neuronalen Netzes mit TensorFlow und Numpy

Um ein neuronales Netz zu programmieren gibt es verschiedene Möglichkeiten. In diesem Kapitel wird die Umsetzung mit der Numpy-Bibliothek und TensorFlow diskutiert. Die entsprechende Umsetzung beider Möglichkeiten findet in Kapitel 4.2 statt. Da in dieser Arbeit beide Möglichkeiten umgesetzt wurden, befinden sich auch entsprechend beide Erläuterungen im genannten Kapitel.

Da Numpy schon eine integrierte Standardbibliothek in Python ist, muss man sich keine weiteren Gedanken machen, eine externe Bibliothek in Python einzubinden. Der Aufbau eines neuronalen Netzes mit dieser Bibliothek muss komplett selbst gestaltet und entwickelt werden. So müssen also Aktivitätsfunktionen, Feedforward- und Backpropagation-Schritte erstellt werden, welche nicht aus der Bibliothek wie in TensorFlow bzw. Keras zur komfortableren Entwicklung mit entsprechenden Funktionen schon enthalten sind. Daher besteht der Vorteil mit TensorFlow unterschiedliche Kombinationen für ein neuronales Netz schneller Umsetzen und Ausprobieren zu können. Wird bei der Entwicklung nur Numpy verwendet, lässt sich die sigmoid-Funktion zum Beispiel einfach Umsetzen, da Numpy eine mathematische Bibliothek von Python ist. Anders als mit Keras, bei der im entsprechenden Layer einfach der Name der Aktivitätsfunktion angegeben werden kann. Durch die mathematischen Vorteile mit Numpy kann ein einfaches neuronales Netz sehr simpel aufgebaut werden. Durch die einfache Umsetzung eines simplen neuronalen Netzes mit Numpy hat sich der Autor zu Beginn erst dafür entschieden, weil TensorFlow zu Beginn der Bachelorarbeit zu mächtig für kleine neuronale Netze ist und Keras dafür dem Autor nicht bekannt war. Weiterhin ist bei der Einschätzung aufgefallen, dass TensorFlow eher für Deep Neural Networks geeignet ist. Also für neuronale Netze mit mehr als zwei Hidden-Layer. Da das neuronale Netz nicht die gewünschten Ergebnisse liefert und zu einfach für die Anforderungen gestaltet ist, ist ein Umstieg auf die Einbindung von Keras als Frontend-Engine mit TensorFlow als Backend-Engine von Vorteil. Durch den Umstieg ist es möglich, einfach Layer hinzuzufügen und besser entscheiden zu können, welche Daten aus den Datensätzen für das Training und für den Test benutzt werden. Um einfach kleinere Netze zu programmieren reicht es aus, Numpy für die Umsetzung zu verwenden und Keras für Netze, die größer sind.

3.9 Zusammenfassung und Ausblick

Mit der beschriebenen Architektur zum Prototypen und dem neuronalen Netz möchte der Autor zeigen, dass es möglich ist, die Anforderungen und das Ziel aus Kapitel 1.4 zum Prototypen umzusetzen. Des weiteren besteht damit die Möglichkeit dieses Gesamtsystem über den Prototypen hinaus weiterzuentwickeln, um diesen als finale Steuerung in ein Smart Home einzubinden.

Die Gegenüberstellung der Architektur aus dem Praxisprojekt zum System in der Bachelorarbeit, zeigt, dass es besser ist einen Prototypen zu verwenden, welcher eine Komponente pro Sitzplatz auf dem Sofa verwendet. So ist es möglich, ein Sofa als

3 Architektur und Aufbau des Prototyps

Steuerung in größerem Umfang zu nutzen, wenn z.B. die Steuerung abhängig von der Sitzplatzzahl auf dem Sofa ist. Zusätzlich ist es mit dem Prototyp auch möglich, dass mehrere Personen gleichzeitig auf dem Sofa sitzen können und dies als Klassifizierung hinzuge nommen werden kann. Dies wird allerdings nicht vom neuronalen Netz in dieser Arbeit mit einbezogen. Das zeigt die Möglichkeiten, welche in Zukunft implementiert werden können. So ist es wichtig, dass die Sensoren nicht beim Einbau in einem Sofa auffallen. Die Anzahl oder Größe der Sensoren spielt auch eine Rolle, damit der Nutzer das Sofa in vollem Umfang zur Steuerung benutzen kann. Daher sollte bei der Auswahl der Sensoren die Größe der jeweils zu messenden Fläche beachtet werden.

Beim neuronalen Netz ist es von Vorteil bei größeren Umsetzungen auf Bibliotheken wie TensorFlow zurückzugreifen. Für einfache Netze reicht Numpy aus, da man dort individuell ohne Bindung an Funktionen, die nicht unbedingt die gewünschten Ergebnisse bringen, die Umsetzung realisieren kann.

Wenn TensorFlow und Keras genutzt wird, können sehr einfach und schnell große neuronale Netze realisiert werden. Dies zeigt auch (Frochte, 2018) bei der Umsetzung eines neuronalen Netzes mit Keras und TensorFlow. In Zukunft ist es wichtig, dass die Sensoren weiter angepasst werden. Zusätzliche Klassifizierungen erweitern dann auch die Interaktionsmöglichkeiten mit dem Sofaprototypen.

4 Umsetzung des neuronalen Netzes zur Erkennung von Interaktionen

Im Rahmen des Praxisprojekts entstand ein Prototyp, welcher ein Sofa darstellt, das als Verwaltung von Interaktionen und Steuerung für smarte Endgeräte dient. Die Verwaltung erfolgt durch ein Regelsystem, welches durch ein ersetzt neuronales Netz werden soll. Der vierte Teil erläutert die Umsetzung von der Aufgabe eines neuronalen Netzes im Smart Home bis zu den Funktionen des neuronalen Netz zur Verwaltung der Interaktionen.

4.1 Anforderungen an die Umsetzung

Zu Beginn ist sehr wichtig zu erwähnen, dass die Echtheit der Werte aus der Außenwelt für ein neuronales Netz garantiert sein müssen. Kapitel 5.6 zeigt den Grund für diese Aussage. Dies wurde zu Beginn der Bachelorarbeit noch nicht bedacht, weshalb in Kapitel 4.2 auch noch andere Werte zur Verarbeitung verwendet werden. Daher ist es wichtig, dass diese Anforderung bei der Entwicklung erfüllt wird. Um Supervised Learning einzusetzen, muss das neuronale Netz einen Datensatz mit den Sensorwerten und einen weiteren Datensatz mit den entsprechenden Positionen einlesen. Weiterhin ist es wichtig, dass die Ergebnisse den Soll-Werten vollständig entsprechen, was mit dem neuronalen Netz in Kapitel 4.3 aber noch nicht komplett der Fall ist. Um dies umsetzen zu können, braucht das neuronale Netz mehr Zeit für die Entwicklung und Verbesserung der Vorhersagen. Damit der Autor die Ergebnisse auswerten kann, ist es wichtig im Programm die Verbesserungen visualisiert werden, bei der Ausführung des Programms. Außerdem muss die Fehlerrate als Wert vorhanden sein, damit Differenzen bei unterschiedlichen Datensätzen zu sehen sind. Zuletzt ist es wichtig, dass das neuronale Netz am Ende so entwickelt ist, dass die Daten aus der Architektur aus Kapitel 3.3 eingelesen werden können. (Frochte, 2018) bietet ein schon fertiges neuronales Netz, welches den Anforderungen zur erfolgreichen Ausführung des Netzes bietet. Das Netz muss entsprechend an den Datensatz angepasst werden, damit es die richtigen Ergebnisse liefert.

4.2 Umsetzung des neuronalen Netzes mit Numpy

Das folgende Kapitel beschreibt die Umsetzung des neuronalen Netzes ohne eine Machine Learning Bibliothek. Die Vorgehensweise eine Python-Bibliothek für Machine Learning zu nutzen, kam erst als spätere Erkenntnis nachdem die Sensordaten über die empirische Methode gesammelt wurden. Da der Datensatz sehr groß ist und die Nutzung einer schon vorhanden Bibliothek ein Vorteil ist, um einfache Änderungen

zur besseren Vorhersage vornehmen zu können, entsteht das neuronale Netz mit Keras erst ab diesem Zeitpunkt. Der Aufbau eines solchen neuronalen Netzes und einer Erläuterung der einzelnen Funktionen und Komponenten, befinden sich in den Grundlagen in Kapitel 2.4 bei den Grundlagen. Mit dem entsprechenden Programmcode erläutert der Autor hier nur die Umsetzung des neuronalen Netzes mit Numpy.

Zu Beginn dieser Umsetzung wurde eine Funktion geschrieben, welche die sigmoid-Funktion beinhaltet. Diese wird in einer Trainingsfunktion aufgerufen. Die Trainingsfunktion ist der Hauptteil dieses neuronalen Netzes.

Die nachfolgende Erklärung des Python-Codes beschreibt die Vorhersage mit drei Units im Input- und Output-Layer. Das komplette selbstgeschriebene neuronale Netz hat insgesamt drei Inputwerte und drei Outputwerte.

```
def train():
    with open("values.txt") as f:
        dataset = [[int(x) for x in line.split(",")]
                    for line in f]
    print(dataset)
```

Als erstes gilt in dieser Funktion das Einlesen und Aufteilen der Textdatei. Danach folgt das Zusammenrechnen der Inputwerte mit den Gewichtungen und die daraus errechnete Summe für jede Unit. An dieser Stelle bekommt das neuronale Netz jeweils einen Inputwert pro Unit im Inputlayer. Zusätzlich wird an dieser Stelle noch der Bias-Wert dazu gerechnet.

```
x = point[0] * w1 + point[1] * w2 + point[2] * w3 + b1
```

Ist die Summe ausgerechnet, werden die Ergebnisse in die sigmoid-Funktion übergeben und die Aktivierungsfunktion wird ausgelöst.

```
pred1 = sigmoid(x)
```

Anschließend rechnet die Fehlerfunktion aus den Ergebnissen und dem gewünschten Ergebnis den mittleren quadratischen Fehler aus. Die Variable *target* beschreibt das gewünschte Ergebnis.

```
cost1 = np.square(pred1 - target1)
```

Das Ergebnis dieser Fehlerfunktion zeigt die Fehlerrate für die Vorhersagen mit den unveränderten Gewichtungen an. Als nächstes wird die Fehlerfunktion und sigmoid-Funktion abgeleitet. Das *d* vor den Namen steht im Code für Derived also abgeleitet.

```
dcost_dpred1 = 2 * (pred1 - target1)
dpred_dz1 = sigmoid_p(x)
```

Diese beiden Funktionen müssen nun anschließend multipliziert werden. Die dadurch entstehenden Ergebnisse müssen durch multiplizieren mit dem gewünschten Ergebnis zusammengerechnet werden. Um nun die Gewichte anzupassen, subtrahiert man das Gewicht mit der Lernrate von 0,1 und multipliziert diese mit der zuvor errechneten *cost* Variable.

```

dcost_dz1 = dcost_dpred1 * dpred_dz1
dcost_dw1 = dcost_dz1 * dz_dw1
w1 = w1 - learning_rate * dcost_dw1

```

Als weitere Erläuterung des Codes wird noch hinzugefügt, dass dieser Vorgang insgesamt 1000 Mal in einer *for-Schleife* durchgeführt wird.

Für sehr kleine und einfach gehaltene Inputwerte kann dieses neuronale Netz verwendet werden. Da für das System aber acht Inputwerte genommen werden, müssten entsprechend auch acht Gewichte benutzt und die Layer angepasst werden. Daher wird im nächsten Kapitel das neuronale Netz vorgestellt, welches mit Keras ausgeführt wird. Durch die Nutzung von Keras und TensorFlow sind größere neuronale Netze wesentlich simpler umzusetzen wie es in Kapitel 4.3 vorgestellt wird.

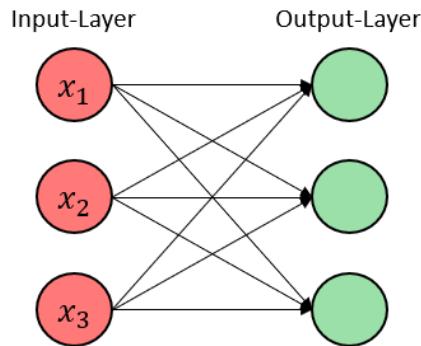


Abbildung 4.1: Vereinfachtes neuronales Netz umgesetzt mit Numpy

Die Abbildung 4.1 zeigt die Architektur des neuronalen Netzes, welches mit Numpy umgesetzt ist. Die farblichen Markierungen zeigen die Units der Layer. Der Input-Layer ist rot und der Output-Layer ist grün. Wie schon erläutert werden hier nur drei Inputwerte eingegeben und vom Netz verarbeitet. Dies kann in der Architektur auch nochmal beobachtet werden.

4.3 Umsetzung des neuronalen Netzes mit Keras

In Kapitel 4.2 bildet sich die Erkenntnis, ein neuronales Netz mit Keras zu entwickeln, da die Art der Umsetzung mit Numpy eher für simplere Netze geeignet ist. Daher ist es wichtig, wie das neuronale Netz für den aktuellen Prototypen umgesetzt ist.

Das neuronale Netz verwendet das Frontend Keras. Dies ist eine unterstützte Bibliothek von TensorFlow. TensorFlow ist das Backend und führt alle Berechnungen und Vorhersagen durch. Kapitel 4.4 beschreibt dessen Aufbau und Funktion sowie die Verbindung zwischen Keras und TensorFlow. Da Keras eine Bibliothek für Python ist, ist das neuronale Netz auch entsprechend damit umgesetzt.

Als Ergänzung zu der Erklärung aus den Grundlagen 2.4 ist noch erweitert zu erläutern, dass das neuronale Netz zum Training in diesem Projekt Datensätze aus CSV-Dateien

einliest. Die Testdaten werden aus den Datasets als zufällige Anzahl ausgewählt. Insgeamt gibt es pro Proband immer eine CSV-Datei für seine Interaktionen auf dem Sofa und eine zweite CSV-Datei für die entsprechenden Vorhersagen. Kapitel 5.4 beschreibt den Aufbau der CSV-Dateien. Da für das neuronale Netz Datensätze benutzt werden, welche vom Prototypen erstellt werden, normalisiert das neuronale Netz die Daten. An diesem Punkt hat Keras auch eine eigene Funktion, die dafür aufgerufen wird.

Das Fully-connected Neural Network ist untereinander mit allen vorherigen Schichten verbunden. Zu Beginn muss ein Sequential-Objekt in Python deklariert werden, damit das neuronale Netz Schicht für Schicht aufgebaut werden kann. Im nachfolgenden Code-Abschnitt werden die Layer des neuronalen Netzes deklariert. Die Funktionsaufrufe *myANN.add(Dense())* sind die einzelnen Layer und mit der *BatchNormalization()* werden die Ergebnisse aus dem vorherigen Layer normalisiert. In Keras wird dieser Aufruf zur Normalisierung wie ein Layer dargestellt. Die Aktivierungsfunktion des Hidden-Layers und Output-Layers ist die sigmoid-Funktion. Außerdem wird der optionale Parameter angegeben, dass die Bias-Unit benutzt werden soll. Der Hidden-Layer ist zu Beginn mit der Rectified Linear Units-Funktion entstanden. Mit dieser Funktion entstand eine Fehlerrate von 72%, was mit der sigmoid-Funktion nicht der Fall ist. Diese Funktion gibt bei Ergebnissen kleiner Null den Wert Null aus und alle Werte darüber werden als dessen Werte ausgegeben. Die weiteren Ergebnisse sind in Kapitel 5.6 aufgeführt und werden dort diskutiert.

```
myANN.add(Dense(8, input_dim=8,
                kernel_initializer='normal'))
myANN.add(Dense(8, kernel_initializer='normal',
                activation='sigmoid', use_bias=True))
myANN.add(BatchNormalization())
myANN.add(Dense(3, kernel_initializer='normal',
                activation='sigmoid', use_bias=True))
```

Als Fehlerfunktion wird Mean Squared Error verwendet und Adam als Optimizer. Aus den Versuchen, die Fehlerrate so gering wie möglich zu halten, stellt sich für den Mean Squared Error und die Learning Rate vom Adam Optimizer ein Wert von 0,001 als beste Wahl heraus. Die Umsetzung des neuronalen Netzes stammt aus einer Vorlage aus dem Buch (Frochte, 2018) und wurde auf den Trainingsdatensatz vom Sofaprototypen angepasst. Das Python-Programm befindet sich in Kapitel 7 als Anhang. Zuletzt wird im neuronalen Netz die *matplotlib.pyplot* importiert. So werden die Veränderungen während des Trainings zur Fehlerfunktion in einem Koordinatensystem dargestellt.

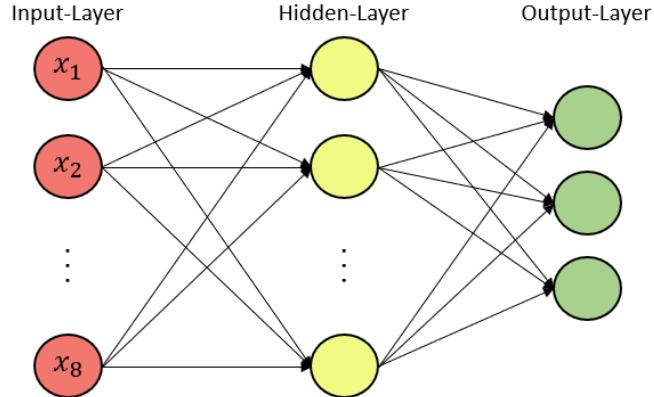


Abbildung 4.2: Darstellung des neuronalen Netzes zur Verwaltung der Interaktionen

Abbildung 4.2 zeigt die Architektur des neuronalen Netzes, welches mit Keras entwickelt wurde. Insgesamt hat der Input- und Hidden-Layer acht Units und der Output-Layer drei Units. Es werden insgesamt acht Inputwerte in Form von Sensordaten im Input-Layer eingegeben und verarbeitet. Die entsprechenden Layer sind zur besseren Unterscheidung farblich markiert. Die Units im Input-Layer sind rot, die Units im Hidden-Layer sind gelb und die Units Output-Layer sind grün. Die Bias-Unit wird hier nicht dargestellt, wird aber trotzdem im Hidden- und Output-Layer als zusätzlicher Wert dazu gerechnet. Wie Keras und TensorFlow funktionieren und in Beziehung zueinander stehen, hat der Autor dies im nächsten Kapitel erläutert.

4.4 Die Backend-Engine TensorFlow

TensorFlow ist ein Open-Source Projekt, welches sich auf Deep Neural Networks spezialisiert. Das neuronale Netz verwendet TensorFlow 2 und Keras 2.3.0. TensorFlow ist eins der möglichen Backend Bibliotheken neben Microsoft Cognitive Toolkit und Theano. (Goldsborough, 2016) erläutert den Aufbau von Tensorflow und beschreibt dessen Elemente. Da TensorFlow für sehr große neuronale Netze ausgelegt ist, nutzt der Autor mit Keras den Vorteil, dass man nicht direkt mit TensorFlow arbeitet. So wird die Backend Bibliothek sehr einfach gehalten und ist von dessen Möglichkeiten übersichtlicher. Um die Unterschiede zu zeigen, wie mit TensorFlow direkt gearbeitet wird, zeigt (Pattanayak, 2017) in Kapitel 2. Wie Keras genutzt wird, zeigt die Umsetzung im Anhang. Abbildung 4.3 zeigt nochmal den Aufbau und die Beziehung zwischen Keras und TensorFlow. Um TensorFlow zu benutzen, muss CUDA/cuDNN auch zusätzlich installiert werden.

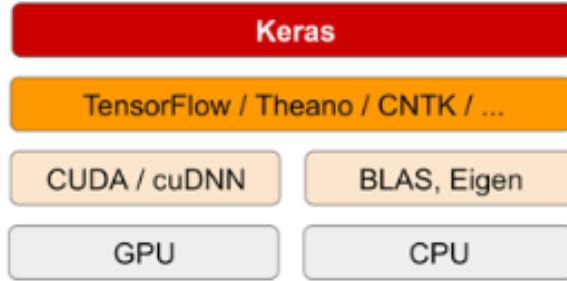


Abbildung 4.3: Der Software und Hardware Stack von Keras (Chollet, 2018)

4.5 Neue Funktionen durch das neuronale Netz

Die Hauptmerkmale zur Erkennung der Interaktionen verändern sich nicht vom Regelsystem aus gesehen mit dem neuronalen Netz. Das Regelsystem bestimmt anhand der Sensorwerte, welche Position der Nutzer auf dem Sofa einnimmt. Dies funktioniert nur, wenn die Personen immer für die von ihnen gewollte Steuerung die gleichen Sensoren besetzen.

Diese statische Steuerung wird durch das neuronale Netz zu einer dynamischen Verwaltung umgestaltet. Durch die Trainingsdaten wird im neuronalen Netz immer eingelesen wann eine Person sitzt, liegt oder das Sofa leer ist, egal wie sie auf dem Sofa diese Positionen ausführt. Das hat den Effekt, dass jede Person selber entscheiden kann, wie sie sich auf das Sofa setzen möchte. Dadurch kann trotzdem immer das gleiche smarte Gerät oder die entsprechende Automatisierung ausgelöst werden. Zudem muss bei neuen Liegepositionen und Sitzpositionen keine Regel aktualisiert oder hinzugefügt werden, da das neuronale Netz auch neue Interaktionen durch neue Nutzer erkennt.

Für die Liegepositionen werden mit dem Regelsystem auch keine eigenen Sensoren verwendet, da die verschiedenen Liegepositionen vom neuronalen Netz erkannt werden und sie von Sitzpositionen unterscheiden. Ein weiterer Vorteil ist das Wegfallen des Flex-Sensors, da das neuronale Netz auch anhand der FSR-Sensoren erkennt, ob ein Nutzer liegt. Auf den Sitzkissen wird nur noch ein Sensor benötigt. Zudem ist es mit Flex-Sensoren sehr schwer zu erkennen, ob eine Person mit ihm interagiert, da dieser erst den Widerstand ab einer 45-Grad-Biegung messen kann. Durch das neuronale Netz fällt der Flex-Sensor weg und die Interaktionen können dynamisch verwaltet werden. Durch das Entfernen des Flex-Sensors wird der ESP8266 nicht mehr gebraucht, was die Erkennung der Sensordaten einheitlich gestalten lässt. Denn es wird nur noch der ESP32 benutzt. Wenn der Prototyp nur noch FSR-Sensoren besitzt, müssen unterschiedliche Sensorwerte nicht mehr verwaltet werden, was die Messung neuer Sensorwerte stark vereinfacht.

4.6 Einbindung des neuronalen Netzes in den Prototypen

Die Erkennung von Interaktionen findet über die Klassifizierung des neuronalen Netzes statt. Anhand der Ergebnisse aus dem Feedforward-Schritt und der darauf folgenden Backpropagation klassifiziert das neuronale Netz wie der Nutzer mit dem Sofa interagiert. Ist diese Aufgabe abgeschlossen und liefert die gewünschten Ergebnisse, besteht die Möglichkeit es in das Python-Programm einzubinden. Das Python-Programm ist im Anhang zu finden. Damit werden das Netz und das Python-Programm zur Erfassung der Sensorwerte auf dem Raspberry Pi zusammen mit dem MQTT-Broker ausgeführt. Wenn diese Klassifizierung mit den Testdaten immer richtig liegt, soll das neuronale Netz in das System des Sofaprototypen eingebunden werden. Durch die Zusammenführung entsteht jedoch der Punkt, dass ein Programm mehrere Aufgaben übernimmt. Deswegen will der Autor nicht unerwähnt lassen, dass es eher von Vorteil ist, wenn das neuronale Netz und das Python-Programm separat ausgeführt werden, damit jede Komponente nur eine Aufgabe hat.

Um zu prüfen, wie das Netz die Vorhersagen genauer treffen kann, beschreibt Kapitel 5.5 die Ergebnisse der Datensätze und welche für das Lernen am besten geeignet sind. Bevor diese Einbindung in den Prototypen stattfindet, muss das neuronale Netz erst noch weiter verbessert werden. Dies wird im folgenden Kapitel erläutert.

4.7 Weiterführung der Entwicklung des neuronalen Netzes

Aus Zeitgründen kann das neuronale Netz nicht so weiter entwickelt werden, dass dessen Vorhersagen immer stimmen. Dieses Kapitel soll in der Theorie beschreiben, was noch implementiert oder geändert werden muss, damit die Vorhersagen besser werden.

Neben der sigmoid- und ReLU-Funktion bietet TensorFlow viele weitere Aktivierungsfunktionen. Dabei muss immer beachtet werden, dass die Aktivierungsfunktion am Ende Werte ausgeben muss, welche Richtung null oder eins gehen. Dies muss aus dem Grund beachtet werden, weil im Ergebnisvektor pro Wert maximal eins oder null steht. Da die Lernrate vom größeren zum kleineren Wert gemessen wird, ist es möglich auch diese noch weiter anzupassen. Weiterhin gibt es mehrere Fehlerfunktionen, welche noch nicht getestet wurden, womit es aber auch möglich ist, das neuronale Netz weiter anzupassen.

4.8 Verbesserungen des Prototypen

In Kapitel 3 sind die Kernkomponenten des Prototypen aus dem Praxisprojekt erläutert. Zudem wird dort die Architektur des weiterentwickelten Prototypen visualisiert und erläutert. Kapitel 4.5 beschreibt daraufhin die mit dem neuronalen Netz einhergehenden Funktionen. Die daraus resultierenden Verbesserungen sind Möglichkeiten zur Verwaltung der Interaktionen. Außerdem werden Interaktionen besser erkannt und können auf verschiedene Arten die smarten Geräte steuern. Der Autor möchte hervorheben, dass das neuronale Netz für eine solche Steuerung die bessere Wahl ist als ein statisches Regelsystem, welches keine veränderten Interaktionen mit der Steuerung

4 Umsetzung des neuronalen Netzes zur Erkennung von Interaktionen

erlaubt. Zudem besteht die Möglichkeit die Interaktionen mit dem Sofa vielfältiger zu gestalten, um mehr Geräte und Abläufe zu Automatisieren.

Die Kommunikation zwischen den Geräten bleibt auch bei der Nutzung eines neuronalen Netzes gleich. Die weitreichendsten Unterschiede sind die Ergebnisse zur Steuerung der smarten Endgeräte und das Einbinden einer CSV-Datei, um das neuronale Netz trainieren und testen zu können. Hier wird das neuronale Netz auch nicht sofort in das Python-Programm eingebunden, so wie beim früheren Regelsystem. Es wird stattdessen als eigene Komponente ausgeführt und kommuniziert mit dem Python-Programm.

5 Evaluation des Trainings durch Probanden

Mit der Evaluation wird die Auswertung der empirischen Methode zur Erfassung der Interaktionen mit dem Sofa erläutert. Weiterhin möchte der Autor die Ergebnisse und Erkenntnisse des neuronalen Netzes und des Gesamtsystems zeigen. Zudem finden in den folgenden Kapiteln auch die Beschreibungen des Aufbaus vom Prototypen auf einem realen Sofa statt. Der Autor möchte also eine Zusammenfassung der Ergebnisse und der empirischen Methode veranschaulichen, um daraus in Teil Sechs der Bachelorarbeit ein Fazit zu ziehen.

5.1 Umgebung zur Erkennung von Interaktionen

Der Prototyp ist so aufgebaut, dass er beliebig um Sitzflächen auf einem Sofa erweitert werden kann. Für die empirische Methode wurde sich dafür entschieden, die Anzahl auf drei Sitzplätze zu beschränken. So kann das Sofa aus Abbildung 5.1 benutzt werden. Jede zusammengesetzte Komponente aus Sensoren am ESP32 steht für einen Platz auf dem Sofa. Zwei ESPs haben einen FSR-Sensor mehr, da diese an den äußeren Sitzplätzen mit einer Armlehne ausgestattet sind. Für den akutellen Prototypen ist es nicht wichtig in welchem Raum, welche Möbel zusätzlich und welche Geräte vorhanden sind. Außerdem gibt es zwei unterschiedlich große FSR-Sensoren, die die gleichen Messwerte liefern und somit keine unterschiedliche Auswirkung auf die Sensorwerte im Datensatz haben.



Abbildung 5.1: Realer Aufbau des Prototyps

Abbildung 5.1 zeigt den Prototypen, welcher auf einem Sofa aufgebaut ist. Die beiden grünen äußeren Kissen stellen die Armlehnen dar. Das wird gemacht, da dieses Sofa keine Armlehnen angebaut hat. Da der Prototyp für ein drei Personen Sofa ausgelegt ist, werden die Sitzplätze durch drei oben liegende Kissen getrennt. Auf der linken Seite ist der Raspberry Pi, mit dem darauf ausgeführten MQTT-Broker und Python-Programm zur Erfassung der Sensorwerte. Die Abbildung macht sichtbar, dass die Verwendung von Ultrasonic-Sensoren nicht positiv ist. Man kann also zum Schluss kommen, dass statt Ultrasonic-Sensoren, dünnerne Sensoren die weniger auffallen, eingebaut werden müssen. Wenn Ultrasonic-Sensoren verwendet werden, dann muss immer eine Fläche in den Sofalehnen zusätzlich frei sein. Der Sensor kann dann dementsprechend den Abstand messen und der Nutzer muss sich immer im Radius der Wellen zur Messung des Abstands an die Rückenlehne anlehnen. Eine Möglichkeit zur Lösung des Problems kann durch Austauschen der Ultrasonic-Sensoren erfolgen, indem Sensoren verwendet werden, welche mehr Platz von der Rückenlehnen einnehmen. Alternativ können mehrere kleine Sensoren verwendet werden.

5.2 Soll-Ablauf der Situation

Dieses Kapitel befasst sich mit der Beschreibung eines Ablaufs zur Sammlung der Daten indem die Probanden mit dem Sofa interagieren. Die folgende Beschreibung der Vorgehensweise zeigt, wie ein Proband in der Theorie mit dem Sofa interagiert, um die bestmögliche Erfassung der Sensordaten zu bekommen.

Um die Klassifizierung richtig vorhersehen zu können, erstellen Probanden durch Interaktionen mit Sofa einen Datensatz für die Trainings- und die Testphase wie sie in Kapitel 2.4.5 beschrieben ist. Ein Proband befindet sich immer alleine im Raum mit dem Sofa. So kann kein anderer Proband durch seine Interaktionen beeinflusst werden. Dort nimmt er verschiedene Positionen auf dem Sofa ein. Hat dieser seine Position festgelegt, werden die Sensorwerte in die CSV-Datei hinzugefügt. Die Position muss immer eine andere sein, damit die Variation mit den Sensoren sich so oft wie möglich unterscheidet. Der Proband soll sich immer hinlegen, hinsetzen oder das Sofa leer lassen. Der Ablauf ist mit jedem Probanden gleich und unterscheidet sich nur von den Interaktionen auf dem Sofa. Sobald 12 bis 15 Probanden diese Situation absolviert haben, wird das neuronale Netz mit diesem Datensatz trainiert und angepasst. Die Probanden sehen außerdem auf dem Monitor, welche Position sie eingenommen haben in Form von den Sensordaten durch die Sensoren die sie benutzen.

Durch das Sammeln der Daten für den Datensatz soll außerdem getestet werden, wie die Probanden mit dem Sofaprotypen umgehen und wie diese damit zureckkommen. So wird dann gezeigt, welche Änderungen an dem Prototypen vorgenommen werden müssen, damit die Probanden die bestmögliche Interaktion mit dem Sofa ausführen können.

5.3 Tatsächlicher Ablauf aller Probanden

Der Autor führt aus Kapitel 5.2 an, wie eine Datensammlung in der Theorie mit einem Probanden ablaufen soll. Der tatsächliche Ablauf mit allen Probanden erweist sich aber als Differenz zur Theorie. Insgesamt wurden mit 17 Probanden Daten gesammelt. In manchen Fällen befanden sich mehrere Probanden im Raum, was den Interaktionen geholfen hat. Denn nicht jeder Proband kann sich sofort vorstellen, was mit den Interaktionen auf dem Sofa gemeint ist.

Jeder Proband nimmt verschiedene Positionen ein und auch unterschiedlich viele, da verschiedene Interaktionen mehrfach auftreten. Je mehr Probanden mit dem Sofa interagieren, desto häufiger treten damit die gleichen Positionen auf. Die Anzahl an Positionen nimmt mit mehr Probanden über die Zeit ab. Damit alle Probanden die gleiche Situation haben, ändert sich der Raum und das Sofa nicht. So hat jeder Proband die gleichen Voraussetzungen. Für diese empirische Methode ist auch kein smartes Gerät oder Möbelstück vorhanden. Da hier nur Sensordaten erfasst werden und das neuronale Netz noch keine entsprechende Vorhersage machen kann die immer korrekt ist.

Teile des Prototyps müssen zwischen den Interaktionen der Probanden auch hin und wieder neu zusammen gebaut werden, da es passieren kann, dass beim Anlehnen an die jeweilige Rückenlehne ein Kabel von den Ultrasonic-Sensoren durch den Probanden ohne Absicht getrennt wird. Die FSR-Sensoren sind zum Großteil nicht davon betroffen, da die Interaktionen nicht so stark auf die Kabel einwirken wie bei den Ultrasonic-Sensoren. Abgesehen davon gibt es mit den Interaktionen keine weiteren Probleme, außer das nicht jeder Proband immer selbstständig weiß, welche Positionen er machen muss. Dies ist aber unwichtig, da es bei dieser Datensammlung auch darum geht zu Testen, welche Änderungen am Prototyp gemacht werden müssen, damit der Proband besser mit dem System zurecht kommt.

5.4 Aufbau des Datensatzes

In diesem Kapitel soll gezeigt werden, wie die Sensordaten für die Trainings- und Testphase gespeichert werden. Eine Position des Probanden ist eine Liste, die als eine Zeile in einer CSV-Datei liegt. Eine weitere CSV-Datei enthält die Positionen, die dem neuronalen Netz mit eingegeben werden. Kapitel 5.6 diskutiert die Ergebnisse aus dem neuronalen Netz und welche Optimierungen im weiteren vorgenommen werden, um die Vorhersagen mit den Testdaten genauer darzustellen. Die Tabelle 5.1 stellt in abwärtiger Reihenfolge die festen Positionen der Sensoren in einem Array dar. Das folgende Beispiel der Daten einer CSV-Datei, zeigt die Positionen als Beispiel.

0, 0, 134, 0, 3095, 0, 0, 717
 4095, 4095, 426, 84, 417, 580, 41, 456
 0, 718, 136, 80, 3102, 4095, 530, 0

Der erste Datensatz zeigt ein leeres Sofa, der zweite eine sitzende Person und der dritte auch, nur auf der anderen Seite des Sofas. Was also aus dem Aufbau zu erkennen ist, dass es wichtig ist wie auch bei dem Regelsystem, wie der Datensatz aufgebaut ist, da sowohl das Regelsystem als auch das neuronale Netz erkennen muss, welcher Wert zu welchem Sensor zugewiesen ist.

Tabelle 5.1: Aufbau eines Arrays des Datensatzes

Sensor	Beschreibung	Werte
Sitzkissen Links	Ein FSR-Sensor erkennt die Sitz- bzw. Liegeposition eines Nutzers	0 bis maximal 4095
Armlehne Links	Der FSR-Sensor erkennt den Kopf, Arm oder Fuß eines Nutzers	0 bis maximal 4095
Rückenlehne Links	Anhand des Abstands wird erkannt, ob ein Nutzer sich anlehnt	ca. 3000 bis maximal 0
Sitzkissen Mitte	Ein FSR-Sensor erkennt die Sitz- bzw. Liegeposition eines Nutzers	0 bis maximal 4095
Rückenlehne Mitte	Anhand des Abstands wird erkannt, ob ein Nutzer sich anlehnt	ca. 3000 bis maximal 0
Sitzkissen Rechts	Ein FSR-Sensor erkennt die Sitz- bzw. Liegeposition eines Nutzers	0 bis maximal 4095
Armlehne Rechts	Der FSR-Sensor erkennt den Kopf, Arm oder Fuß eines Nutzers	0 bis maximal 4095
Rückenlehne Rechts	Anhand des Abstands wird erkannt, ob ein Nutzer sich anlehnt	ca. 3000 bis maximal 0

5.5 Ergebnisse der Datensammlung

Es lässt sich anhand der Ergebnisse aus der Datensammlung zeigen, dass die FSR-Sensoren auch Werte größer 0 ausgeben und unterhalb des Maximalwerts liegen. Obwohl die Probanden keinen direkten Druck auf diese ausüben. Daraus lässt sich schlussfolgern, dass das neuronale Netz auch diese Datenpunkte erkennen muss. Damit wird bezweckt, dass die Ergebnisse trotzdem zur richtigen Vorhersage beitragen. Um zu überprüfen wie die Datenpunkte sich auf die Klassifizierung auswirken, testet der Autor den Datensatz sowohl mit und ohne Filterung der Datenpunkte.

Weiterhin sei auch zu erwähnen, dass die Abstandserkennung durch die Ultrasonic-Sensoren bei geringen Abständen sehr genau arbeitet. Wenn sich ein Proband anlehnt kann es auch vorkommen, dass ein Kabel von einem Ultrasonic-Sensor entfernt wird. Tritt dieser Fall auf, wird in der Datensammlung der Wert 0 gespeichert. Da ein Kabel immer dann abgerissen wird, wenn ein Nutzer sich anlehnt, ist es also korrekt, dass der Sensor den Wert 0 übergibt. Mit diesen Ergebnissen steht es außer Frage, dass die Datensammlung für das neuronale Netz geeignet ist.

5.6 Ergebnisse und Evaluation des neuronalen Netzes

Die folgenden Kapitel sollen die Evaluation des neuronalen Netzes näher erläutern. Es werden die Datensätze analysiert und deren Ergebnisse präsentiert. Weiterhin sollen Erkenntnisse aus den Ergebnissen gezogen werden. Das Kapitel zeigt außerdem die Unterschiede des Datensatzes wenn dessen Datenpunkte gefiltert in die Vorhersage eingenommen werden und wie die Klassifizierung ohne die Filterung der Datenpunkte aussieht.

Für das neuronale Netz werden zwei verschiedene Datensätze benutzt. Die Datensätze entstehen aus den gleichen Rohdaten, zeigen aber unterschiedliche Ergebnisse. Als erstes werden die Daten der Probanden benutzt ohne, dass sie verändert wurden. Damit will der Autor zeigen, wie die Ergebnisse des neuronalen Netzes aussehen, wenn alle Datenpunkte gespeichert sind. Denn eine Position tritt mehrfach in der CSV-Datei hintereinander auf. Mit der Abbildung 5.3 soll gezeigt werden, wie das Lernverhalten des neuronalen Netzes mit allen Datenpunkten ausfällt. Dies bedeutet, dass eine Position mehrfach in die CSV gespeichert wird und damit können bei einer Position geringe Veränderungen auftauchen. Mit der Abbildung 5.2 wird dementsprechend gezeigt, wie das Lernverhalten ohne das Filtern der Datenpunkte aussieht.

5.6.1 Datensatz mit dem Filtern von Datenpunkten

Als ersten Datensatz werden die Positionen der Probanden verwendet, dass sie einmal in das neuronale Netz eingelesen werden. Dies bedeutet, dass jeder Datensatz nicht mehrfach auftaucht und so Veränderungen bei einer Position nicht gespeichert werden. So fällt das mehrfache Auftreten der Positionen weg und damit sieht auch das Ergebnis anders aus. Die folgende Abbildung 5.2 zeigt das Lernverhalten des neuronalen Netzes mit den gefilterten Datenpunkten. Anhand dieser Abbildung, zeigt der Autor auch den Unterschied zum unveränderten Datensatz.

5 Evaluation des Trainings durch Probanden

Die Fehlerrate beim ersten Datensatz liegt bei 0,16. Damit liegt das neuronale Netz zu 84% richtig bei den Vorhersagen. Für das Training- und die Testphase werden insgesamt 245 Positionen beim Filtern der Datenpunkte von allen 17 Probanden gespeichert. Der Unterschied zum Datensatz mit allen Datenpunkten ist sehr groß, da dieser aus 2934 Datensätzen besteht.

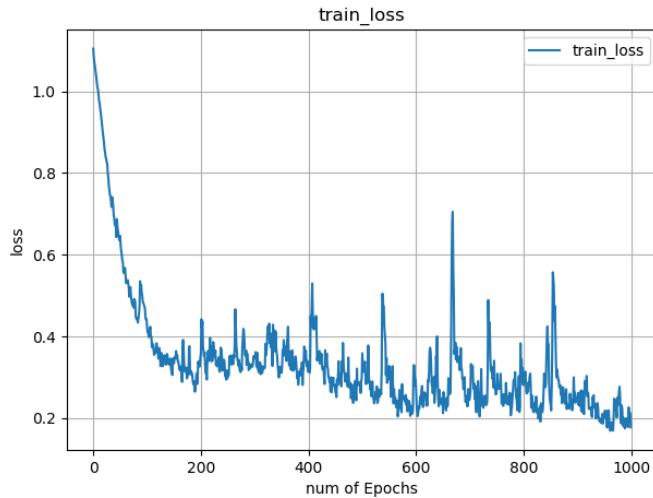


Abbildung 5.2: Abbildung der Fehlerrate des neuronalen Netzes mit Filterung der Datenpunkte

5.6.2 Datensatz ohne Filterung der Datenpunkte

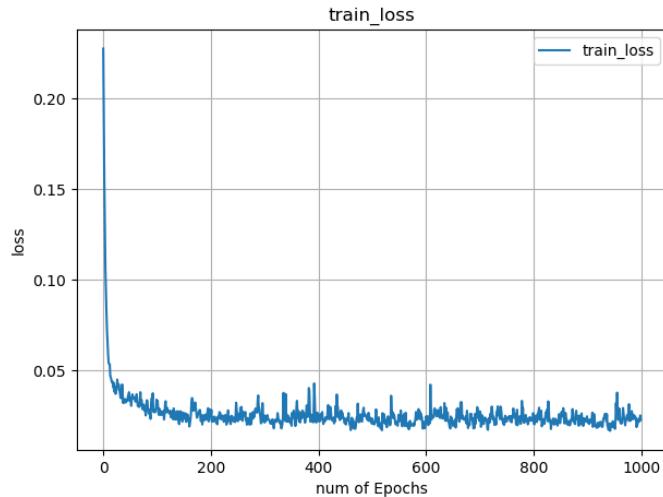


Abbildung 5.3: Abbildung der Fehlerrate des neuronalen Netzes mit allen Datenpunkten

Werden alle Datenpunkte in den Datensätzen gelassen, so liegt die Fehlerrate bei 0,03. Somit klassifiziert das neuronale Netz zu 99,7% richtig. Damit ist es schon sehr nah an den 100% und kann schon fast in den Prototypen mit eingebunden werden.

5.7 Zusammenfassung der Evaluation

Aus den vorherigen Kapiteln der Evaluation wird gezeigt, wie verschiedene Probanden mit dem Prototypen umgehen. So ist also nochmal zu sagen, dass der Prototyp durchersetzen der Ultrasonic-Sensoren eine bessere Möglichkeit bietet um zu erkennen, wann ein Nutzer die Rückenlehne des Sofas verwendet. Wenn diese Sensoren ersetzt werden, kann die Rückenlehne des Sofas auch besser mit in die Verwaltung der Interaktionen einfließen.

Aus 5.6 ist zu erkennen, dass es besser ist ein neuronales Netz mit Daten zu trainieren, die so nah wie möglich an der Realität sind. Zudem verläuft das Training ohne die gefilterten Daten wesentlich besser ab. Es ist nicht wichtig wie viele Daten benutzt werden, sondern wie viele verschiedene Kombinationen dem neuronalen Netz zum trainieren gegeben wird. Durch die geringere Anzahl an Interaktionen die der Datensatz mit Filterung der Datenpunkte vorgibt, ist die Fehlerrate auch dementsprechend höher.

5.8 Erkenntnisse aus der Evaluation zum Prototypen

Aus folgender Überlegung aus dem Kapitel 5.3, dass Probanden nach einer Zeit wiederholt die gleiche Sitzposition einnehmen, werden in diesem Kapitel die Schlüsse und Erkenntnisse daraus gezogen. Diese Erkenntnisse sind so beschrieben, dass bestimmte Punkte die Erkenntnis aus der Evaluation bilden.

So deuten diese Ergebnisse darauf hin, dass es von Vorteil ist, mehr Sensoren zur Erkennung der Interaktionen auf dem Sofa zu verbauen. So können weitere Werte aus den Realwerten gemessen werden, um so noch besser die Interaktionen im neuronalen Netz vorhersagen zu können. Als weitere Möglichkeit geht daraus hervor, nur FSR-Sensoren zu verwenden. Alternativ können auch Sensoren verwendet werden, welche die komplette Fläche einnehmen. Da die Probanden durch ihre Interaktionen auf dem Sofa Druck auf die Oberflächen ausüben, ist dies ein weiterer Grund nur FSR-Sensoren zu benutzen. Zusammengefasst kann also erwähnt werden, dass der Prototyp im aktuellen Zustand noch überarbeitet werden muss, da die Sensoren wie der Ultrasonic-Sensor für die Erkennung nicht von Vorteil ist. Also ist eine weitere Erkenntnis aus der Evaluation, dass für einen Prototypen mit den Interaktionen aus Sitz- und Liegepositionen erfasst werden, Sensoren zu verwenden, welche ausschließlich den Druck oder die Kraft messen.

5 Evaluation des Trainings durch Probanden

Anhand des letzten Kapitels wird gezeigt wie Vorhersagen ausfallen, wenn ein Datensatz mit allen Datenpunkten verwendet wird. Zusätzlich wird der Datensatz aus den Sensorwerten erstellt bei dem die Position nur einmal gespeichert wird. Zusammengefasst liegt die Fehlerrate bei dem Datensatz mit den gefilterten Datenpunkten bei 0,16. Dies ist im Vergleich zur Fehlerrate von 0,03 eine großer Unterschied. Dies zeigt, dass das Trainieren mit allen Datenpunkten besser ist.

6 Fazit und Aussicht für den Prototypen

Der Prototyp ist mit dem neuronalen Netz verbunden. Zur Erkennung und Verwaltung von Interaktionen, ist dies eine gute Möglichkeit, um Steuerungen wie Smartphones oder ähnliche Geräten zum Teil zu ersetzen. Da kein Einfluss auf die Automatisierung oder Steuerung einzelner Geräte genommen werden kann, sind Smartphones oder ähnliche Geräte eine optionale Steuerungsmöglichkeit für kleinere Veränderungen der smarten Geräte oder Möbelstücke. Dies hat den Nachteil für eingeschränkte Personen. Personen die körperlich oder geistig eingeschränkt sind, haben mit der Sofasteuerung den Vorteil, sich nicht um die smarten Geräte durch ein Smartphone oder ähnlichen Geräten kümmern zu müssen.

Dem Autor ist bei der Datensammlung mit den Sensoren besonders aufgefallen, dass das Sofa wie ein herkömmliches Sofa genutzt werden muss, damit die Nutzer am besten damit klar kommen. Wichtig ist, dass die Nutzer bei der Einrichtung eines solchen Systems das Sofa aktiv nutzen, bevor das neuronale Netz die Interaktionen verwaltet. So kann spezifisch darauf eingegangen werden, wie der Nutzer das Sofa im Alltag benutzt.

Der Autor möchte auch nochmal betonen, dass die Steuerung besonders für Personen geeignet ist, welche ihren Alltag nicht ohne Hilfe bewältigen können. Die Interaktionen mit dieser Steuerung führen zu einer passiven Automatisierung bzw. Steuerung smarter Endgeräte oder Möbelstücke, wodurch Nutzer sich nicht mehr darum kümmern müssen.

6.1 Zusammenfassung der Bachelorarbeit

Mit dieser Bachelorarbeit will der Autor zeigen wie es möglich ist, die Innenausstattung in das Smart Home zu integrieren. Anhand eines Beispiels an einem Sofa zeigt der Autor dessen Umsetzung. Der Prototyp besteht aus mehreren ESP32 und einem Raspberry Pi 3 sowie verschiedenen Sensoren, die an den ESPs angeschlossen sind. Die Evaluation der empirischen Methode zeigt, dass der Ultrasonic-Sensor gegen FSR-Sensoren ausgetauscht werden muss. Da die Messung an den Sofalehnen dadurch beeinträchtigt wird. Dennoch lässt sich aus dem dritten Teil der Arbeit zur Architektur sagen, dass diese ohne Probleme als Prototyp umgesetzt wird.

Weiterhin sollen die Interaktionen mit diesem Prototypen verwaltet werden. Diese Aufgabe übernimmt ein neuronales Netz. Das neuronale Netz unterteilt die Ergebnisse am Ende in drei Klassen. Es lernt diese Klassifizierungen richtig vorzusagen, indem zwei Datensätze basierend auf den gleichen Sensorwerten trainiert werden. Ein Datensatz behält alle Datenpunkte und ein zweiter Datensatz bekommt nur die gefilterten Interaktionen. Das Ergebnis zeigt deutlich, dass der Datensatz mit allen Datenpunkten eine Fehlerrate von 0,03 hat und damit wesentlich besser klassifiziert. Im Vergleich

6 Fazit und Aussicht für den Prototypen

dazu hat der zweite Datensatz mit einer Fehlerrate 0,16 schlechter abgeschnitten. Also wird nochmal gezeigt, dass es sehr wichtig ist, dass die Sensorwerte so genau wie möglich sind.

Für das neuronale Netz ist es auch sehr wichtig, wie es aufgebaut ist. Es macht einen Unterschied welche Aktivitätsfunktionen benutzt werden und ob der Bias-Wert hinzugezogen wird. Zusätzlich muss die Lernrate angepasst werden indem bei beim höchsten Wert angefangen wird und die Lernrate immer weiter runter geht. Die Normalisierung ist wichtig, wenn bei TensorFlow ein eigener Datensatz benutzt wird. TensorFlow bietet daneben auch schon vorbereitete, skalierte Datensätze an.

Der Prototyp hat von der Sicht des Autors bei Weiterentwicklung gute Aussichten auf eine reale Einbindung in ein Smart Home. Damit zieht der Autor das Fazit, dass Möbel sich in ein Smart Home als smarte Erweiterung einbinden lassen. Wenn das Smart Home bestimmte Möbel, wie Sitzmöglichkeiten zur Steuerung nutzt, dann ist es außerdem ein Vorteil, wenn dafür Machine Learning Methoden eingesetzt werden. Dies zeigt die Verbindung des Sofas, mit der Verwaltung durch das neuronale Netz. Die Grundmerkmale aus dieser Arbeit sind also ein Prototyp zur Erfassung der Sensordaten aus den Interaktionen mit dem Sofa und ein neuronales Netz zur Verwaltung dieser Interaktionen. Somit ist es also immer wichtig in einer Steuerung, eine Messung von Daten zu implementieren und eine Methode diese zu verarbeiten. Somit hat durch die komplette Arbeit hinweg den Autor immer wieder beschäftigt, wie diese Aufgaben richtig aufgeteilt werden.

6.2 Zukunftsauussicht für das System

Da der Prototyp noch nicht vollkommen für ein Smart Home bereit ist, gibt es für die Zukunft weitere Punkte die zur Verbesserung beitragen. Wichtig dabei ist, dass jede Komponente weiterhin nur für eine Aufgabe zuständig ist. Zudem wird des öfteren in der Arbeit erwähnt, dass der Ultrasonic-Sensor nicht die ideale Lösung ist. Daher ist es besser, wenn entweder nur FSR-Sensoren eingebaut sind oder Sensoren benutzt werden, die die komplette Fläche eines Sofakissens oder Armlehne einnehmen. Der Vorteil beim implementieren mehrerer Sensoren auf einer Fläche ist, die damit höhere Genauigkeit der Interaktionen. Um herauszufinden welche Sensoren weiterhin benutzt oder ersetzt werden, muss der Prototyp weiter von Probanden getestet werden. Um die Situationen bei diesen Tests realistischer zu gestalten, bietet die Implementierung von smarten Endgeräten in den Prototypen ein genauereres Umfeld in Smart Homes. Weiterhin sollten Probanden hinzugezogen werden, an die dieser Prototyp gerichtet ist. Im ersten Teil in Kapitel 1.3 beschreibt der Autor diese Personen.

6 Fazit und Aussicht für den Prototypen

Das neuronale Netz muss in Zukunft dann auch weiter angepasst werden. Da sich die Sensorwerte dann noch ändern und erweitern werden. So kann noch genauer klassifiziert werden. Dadurch werden außerdem die Möglichkeiten größer im Output-Layer mehr Ergebnisse zur Verwaltung zu benutzen. Also kann die Automatisierung und Steuerung der smarten Geräte individuell gestaltet werden. Für Personen die körperlich oder geistig eingeschränkt sind, ist die Selbstständigkeit und Sicherheit dadurch weiterhin gewährleistet und wird dann immer besser an die Person angepasst. In Zukunft ist es also weiterhin ein System, welches sich an die genannten Personen richtet.

Als abschließende Bemerkung vom Autor, zeigt die Bachelorarbeit, dass der Prototyp darstellt wie die Innenausstattung mit in ein Smart Home implementiert werden kann und dies hat besonders für eingeschränkte Personen die Hilfe benötigen einen Vorteil zur Selbstständigkeit und Sicherheit.

7 Anhang

```
import paho.mqtt.client as mqttClient
import time
import sys

sensors = [0, 0, 0, 0, 0, 0, 0, 0]
pubMsg = ""

def on_connect(client, userdata, flags, rc):

    if rc == 0:

        print("Connected to broker")

        global Connected
        Connected = True

    else:

        print("Connection failed")

def on_message(client, userdata, message):

    if message.topic == "fsr1":
        print("Message received 1: " +
              message.payload.decode("utf-8"))
        sensors[0] = message.payload.decode("utf-8")

    if message.topic == "fsr2":
        print("Message received 2: " +
              message.payload.decode("utf-8"))
        sensors[1] = message.payload.decode("utf-8")

    if message.topic == "dis":
        print("Message received 3: " +
              message.payload.decode("utf-8"))
        sensors[2] = message.payload.decode("utf-8")

    if message.topic == "fsr3":
        print("Message received 4: " +
```

```

        message.payload.decode("utf-8"))
sensors[3] = message.payload.decode("utf-8")

if message.topic == "dist2":
    print("Message received 5: " +
          message.payload.decode("utf-8"))
    sensors[4] = message.payload.decode("utf-8")

if message.topic == "fsr4":
    print("Message received 6: " +
          message.payload.decode("utf-8"))
    sensors[5] = message.payload.decode("utf-8")

if message.topic == "fsr5":
    print("Message received 7: " +
          message.payload.decode("utf-8"))
    sensors[6] = message.payload.decode("utf-8")

if message.topic == "dist3":
    print("Message received 8: " +
          message.payload.decode("utf-8"))
    sensors[7] = message.payload.decode("utf-8")

print(sensors)
file2write=open("dataset.txt",'a')
file2write.write(str(sensors) + "\n")
file2write.close()

def on_publish(client, userdata, result):
    print("Data published \n")
    pass

Connected = False

broker_adress = "IP Adress of the broker"
port=1883

client = mqttClient.Client("mosq/E4hpAcdlgjIy5b74cE")
client.on_connect= on_connect
client.on_message= on_message
client.connect(broker_adress, port=port)
client.loop_start()

while Connected != True:
    time.sleep(0.1)

```

```
client.subscribe("fsr1")
client.subscribe("fsr2")
client.subscribe("fsr3")
client.subscribe("fsr4")
client.subscribe("fsr5")
client.subscribe("dis")
client.subscribe("dist2")
client.subscribe("dist3")

try:
    while True:
        time.sleep(1)

except KeyboardInterrupt:
    print("exiting")
    client.disconnect()
    client.loop_stop()

client1 = mqttClient.Client("controller1")
client1.on_publish = on_publish
client1.connect(broker_adress, port=port)
ret = client.publish("living_room", pubMsg)
```

```

import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import BatchNormalization
import keras
import matplotlib.pyplot as plt
import tensorflow
from time import time

np.random.seed(42)

#train_samples
fFloat = open("path\\to\\my\\
                csv\\sensordata","r")
X = np.loadtxt(fFloat, delimiter=",",
               dtype=float); fFloat.close()
#train_labels
fFloat = open("path\\to\\my\\
                csv\\positions","r")
Y = np.loadtxt(fFloat, delimiter=",",
               dtype=float); fFloat.close()

yMin = Y.min(axis=0); yMax = Y.max(axis=0)
Y = (Y - yMin) / (yMax - yMin)

TrainSet = np.random.choice(X.shape[0], int(X.shape[0]*0.80),
                           replace=False)
XTrain = X[TrainSet,:]
YTrain = Y[TrainSet]
TestSet = np.delete(np.arange(0, len(Y)), TrainSet)
XTest = X[TestSet,:]
YTest = Y[TestSet]

myANN = Sequential()

myANN.add(Dense(8, input_dim=8,
                 kernel_initializer='normal'))
myANN.add(Dense(8, kernel_initializer='normal',
                 activation='sigmoid', use_bias=True))
myANN.add(BatchNormalization())
myANN.add(Dense(3, kernel_initializer='normal',
                 activation='sigmoid', use_bias=True))
myANN.compile(loss='mean_squared_error', optimizer='adam')

```

```
history = myANN.fit(XTrain, YTrain,
                     epochs=1000, verbose=False)
print(history)
yp = myANN.predict(XTest)
yp = yp.reshape(yp.shape[0], -1)
errorT = (yMax - yMin) * (yp - YTest)
#print(np.mean(np.abs(errorT)))
for i in range(len(XTest)):
    print(str(XTest[i]) + ";" + str(yp[i]))

print("Error: " + str(np.mean(np.abs(errorT)))))

train_loss=history.history['loss']
xc=range(1000)

plt.figure(1, figsize=(7,5))
plt.plot(xc, train_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss')
plt.grid(True)
plt.legend(['train_loss'])
plt.style.use(['classic'])
plt.show()
```

Abbildungsverzeichnis

2.1	Darstellung eines Perzeptrons	10
2.2	Beispiel einer Klassifizierung (Yu)	12
2.3	Die linke Abbildung zeigt den Raspberry Pi und die Rechte den ESP32	14
2.4	Links ist der Ultrasonic-Sensor, in der Mitte der Große und rechts der kleine FSR	15
3.1	Visuelle Darstellung zum Aufbau des Systems	18
3.2	Abschließende Architektur mit einem Endgerät als Beispiel	19
3.3	Die Kommunikationsarchitektur ohne Endgerät als Beispiel	23
4.1	Vereinfachtes neuronales Netz umgesetzt mit Numpy	28
4.2	Darstellung des neuronalen Netzes zur Verwaltung der Interaktionen . .	30
4.3	Der Software und Hardware Stack von Keras (Chollet, 2018)	31
5.1	Realer Aufbau des Prototyps	34
5.2	Abbildung der Fehlerrate des neuronalen Netzes mit Filterung der Datenpunkte	39
5.3	Abbildung der Fehlerrate des neuronalen Netzes mit allen Datenpunkten	39

Tabellenverzeichnis

2.1	Tabellarische Darstellung der QoS Level	13
3.1	Aufzählung der Regeln des Regelsystems	21
5.1	Aufbau eines Arrays des Datensatzes	37

Literaturverzeichnis

- [Al-Ali u. a. 2017] AL-ALI, Abdul-Rahman ; ZUALKERNAN, Imran A. ; RASHID, Mohammed ; GUPTA, Ragini ; ALIKARAR, Mazin: A smart home energy management system using IoT and big data analytics approach. In: *IEEE Transactions on Consumer Electronics* 63 (2017), Nr. 4, S. 426–434
- [Badlani u. Bhanot 2011] BADLANI, Amit ; BHANOT, Surekha: Smart home system design based on artificial neural networks. In: *Proceedings of the World Congress on Engineering and Computer Science* Bd. 1, 2011, S. 146–164
- [Chollet 2018] CHOLLET, François: *Deep Learning with Python*. Manning Publications Co., 2018. – 62 S. – ISBN 978-1-617-29443-3
- [Das u. a. 2015] DAS, Resul ; TUNA, Gurkan ; TUNA, Ayse: Design and Implementation of a Smart Home for the Elderly and Disabled. In: *environment* 1 (2015), S. 3
- [De Simone u. a. 2018] DE SIMONE, Marco ; RIVERA, Zandra ; GUIDA, Domenico: Obstacle avoidance system for unmanned ground vehicles by using ultrasonic sensors. In: *Machines* 6 (2018), Nr. 2, S. 18
- [Demeure u. a. 2014] DEMEURE, Alexandre ; CAFFIAU, Sybille ; COUTAZ, Joëlle: Activity based End-User-Development for Smart Homes: Relevance and Challenges. In: *Intelligent Environments (Workshops)*, 2014, S. 141–152
- [Flórez u. Velasquez 2010] FLÓREZ, JA ; VELASQUEZ, A: Calibration of force sensing resistors (fsr) for static and dynamic applications. In: *2010 IEEE ANDESCON* IEEE, 2010, S. 1–6
- [Frochte 2018] FROCHTE, Jörg: *Grundlagen und Algorithmen in Python : mit 146 Abbildungen, 22 Tabellen und zahlreichen Beispielen*. Hanser, 2018. – 210–250 S. – ISBN 978-3-446-45705-8
- [Goldsborough 2016] GOLDSBOROUGH, Peter: A tour of tensorflow. In: *arXiv preprint arXiv:1610.01178* (2016)
- [Hollinger u. Wanderley 2006] HOLLINGER, Avrum ; WANDERLEY, Marcelo M.: Evaluation of commercial force-sensing resistors. In: *Proceedings of the International Conference on New Interfaces for Musical Expression, Paris, France*, 2006, S. 4–8
- [Hussein u. a. 2014] HUSSEIN, Ali ; ADDA, Mehdi ; ATIEH, Mirna ; FAHS, Walid: Smart home design for disabled people based on neural networks. In: *Procedia Computer Science* 37 (2014), S. 117–126

Literaturverzeichnis

- [Pattanayak 2017] PATTANAYAK, Santanu: *Pro Deep Learning with TensorFlow*. Santanu Pattanayak, 2017. – 118–123 S. – ISBN 978-1-4842-3095-4
- [Piyare 2013] PIYARE, Rajeev: Internet of things: ubiquitous home control and monitoring system using android based smart phone. In: *International journal of Internet of Things* 2 (2013), Nr. 1, S. 5–11
- [Ramlee u. a. 2012] RAMLEE, RA ; TANG, DHZ ; ISMAIL, MM: Smart home system for disabled people via wireless bluetooth. In: *2012 International Conference on System Engineering and Technology (ICSET)* IEEE, 2012, S. 1–4
- [Rialle u. a. 2002] RIALLE, Vincent ; DUCHENE, Florence ; NOURY, Norbert ; BAJOLLE, Lionel ; DEMONGEOT, Jacques: Healthsmart”home: information technology for patients at home. In: *Telemedicine Journal and E-Health* 8 (2002), Nr. 4, S. 395–409
- [Saggio u. a. 2015] SAGGIO, Giovanni ; RIILLO, Francesco ; SBERNINI, Laura ; QUITADAMO, Lucia R.: Resistive flex sensors: a survey. In: *Smart Materials and Structures* 25 (2015), Nr. 1, S. 013001
- [Schröder 2019] SCHRÖDER, Jan: Smart Home Interior. (2019). – Last accessed 10 October 2019
- [Soares u. a. 2010] SOARES, Symone G. ; DA ROCHA, Adson F. ; BARBOSA, Tales Marcelo G de A. ; MATOS ARAÚJO, Rui A.: Embedding a Neural Network into WSN furniture. In: *2010 10th International Conference on Hybrid Intelligent Systems* IEEE, 2010, S. 219–222
- [Soni u. Makwana 2017] SONI, Dipa ; MAKWANA, Ashwin: A survey on mqtt: a protocol of internet of things (iot). In: *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, 2017
- [Sripan u. a. 2012] SRIPAN, Meensika ; LIN, Xuanxia ; PETCHLORLEAN, Ponchan ; KETCHAM, Mahasak: Research and thinking of smart home technology. In: *International Conference on Systems and Electronic Engineering (ICSEE'2012)*, 2012, S. 61–63
- [Trojan 2017] TROJAN, Walter: *Das MQTT-Praxisbuch*. Elektor-Verlag GmbH, 2017. – 12–17 S. – ISBN 978-3-89576-324-3
- [Wender u. Rey 2011] WENDER, Karl F. ; REY, Günter D.: *Neuronale Netze Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung*. Verlag Hans Huber, 2011. – 14–31 S. – ISBN 978-3-456-84881-5
- [Yu] YU, Peter: *Visualize classifier decision boundaries in MATLAB*. – http://www.peteryu.ca/tutorials/matlab/visualize_decision_boundaries

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 21. Januar 2020

A handwritten signature in blue ink, appearing to read "Jan Schröder".

Jan Schröder