Untersuchung zur Eignung mehrlagiger Perzeptren für die Indoor-Lokalisierung durch Bluetooth-Beacons

PRAXISBERICHT I

T1000

des Studienganges Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Maximilian Hengl

05.09.2022

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	4907952, STG-TINF21E
Dualer Partner	telent GmbH, Backnang
Betreuer, E-Mail	Christoph Tonnier christoph.tonnier@telent.de

telent
service • commitment • value

DHBW
Duale Hochschule
Baden-Württemberg

Inhaltsverzeichnis

	Abbildungsverzeichnis	
	Abkürzungsverzeichnis	
	Ehrenwörtliche Erklärung	
1	Einleitung	1
2	Aufgabenstellung	4
3	Künstliche Neuronale Netze in der Theorie	5
4	Bibliotheken für Programmierschnittstellen	13
5	Probeaufgaben	16
6	Anwendung eines Perzeptrons zur Positionsbestimmung	22
7	Ausblick	32
8	Fazit	33
	Literaturverzeichnis	
	Anhang	

Abbildungsverzeichnis

Abb. 1	Vergleich der Funktion ReLU mit der logistischen Funktion in einem Liniendiagramm (S. 7)
Abb. 2	grafische Darstellung eines 2-Ebenen Netzes, [4, S. 529] (S. 8)
Abb. 3	kartographische Darstellung der Messpunktverteilung innerhalb der Halle (S. 23)
Abb. 4, Abb. 5	Vergleiche zwischen Hypotenusenlängen und den Summen aus den Kathetenlängen (S. 24)
Abb. 6	Codeausschnitt des Regressionsnetzes zur Ortung (S. 27)
Abb. 7	Codeausschnitt des Genauigkeitstests für die Regression (S. 28)
Abb. 8	Abweichungen des Regressionsnetzes bei Trainingsdaten (S. 28)
Abb. 9	Aufteilung der Trainingshalle zur Klassifizierung (S. 29)
Abb. 10	Codeausschnitt des Klassifizierungsnetzes zur Ortung (S. 30)
Abb. 11	Codeausschnitt des Genauigkeitstests für die Klassifizierung (S. 31)
Abb. 12	Konsolenausgabe über die Genauigkeit der Klassifizierung (S.31)

Abkürzungsverzeichnis

KNN	Künstliches Neuronales Netz
LoRaWAN	Long Range Wide Area Network (Netzprotokoll auf Vermittlungsschicht)
LTE	Long Term Evolution (Mobilfunkstandard)
ReLU	rectified linear unit (Aktivierungsfunktion)
MSE	mean squared error (mittlere quadratische Abweichung)
RMSProp	Root Mean Square Propagation (Gradientenabstiegsverfahren)
Adam	Adaptive Movement Estimation (Gradientenabstiegsverfahren)
RSSI	Received Signal Strength Indicator (Indikator für Signalstärke)
CSV	Comma-separated values (Dateiformat für Tabellen oder Listen)

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema „Untersuchung zur Eignung mehrlagiger Perzeptren für die Indoor-Lokalisierung durch Bluetooth-Beacons“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Backnang, den 02.09.2022	
Ort Datum	Unterschrift

1 Einleitung

Seit ihren Anfängen war es immer Ziel der IT, Arbeit möglichst effizient zu erledigen. Ein großer Fortschritt für Reisende war zum Beispiel der Wechsel vom Atlas zu einem Navigationssystem. Nun musste man nicht mehr selbst herausfinden, wo man gerade ist und wie man zu dem Ziel auf seiner Karte kommt. Das erspart Zeit und Mühe.

Doch ein solches System zur Positionsbestimmung arbeitet mit Strahlung. Dadurch gibt es viele Störfaktoren: Strahlung kann reflektiert, gebrochen oder absorbiert werden. Signale sollten also im Optimalfall aus mehreren Richtungen gesendet werden. Wenn ein Strahlungsweg gestört ist, können Signale so noch von anderen Seiten empfangen werden. Eine höhere Anzahl an Sendern macht allerdings auch den Algorithmus zur Positionsbestimmung komplexer.

Wenn ein Ortungssystem für einen speziellen Ort voller Störfaktoren entwickelt wird, ist es unabdingbar, die Störfaktoren durch auf sie zugeschnittene Formeln im Algorithmus auszugleichen. Der Arbeitsaufwand, ein passendes Modell zu finden, ist also sehr hoch. Doch die IT hat bereits etwas hervorgebracht, was effizient komplexe Algorithmen erzeugen kann: Künstliche Neuronale Netze (KNN).

Von Sprachassistenten, die es Sehbehinderten ermöglichen, das Potential des Internets zu nutzen, bis hin zu Algorithmen, die verschiedene Spezies auf Bildern identifizieren können, ist heute dank KNN vieles möglich, was vor wenigen Jahrzehnten noch für Science-Fiction gehalten wurde. Da diese Technik so ein hohes Potential hat, ist es also naheliegend, dass sie auch zur Eruierung eines Algorithmus für die zuletzt beschriebene Positionsbestimmung fähig ist.

Im Umfang dieser Arbeit werden sowohl die Funktionsweise, als auch die Anwendung einer Art des Neuronalen Netzes, des mehrlagigen Perzeptrons, beleuchtet. Anschließend wird betrachtet, wie gut sich ein KNN zur Schaffung eines spezifischen Ortungsalgorithmus eignet.

1.1 Einsatzort

Das Ortungssystem, das betrachtet wird, soll in einer Fabrikhalle, in der Kräne hergestellt werden, das Finden von Einzelteilen erleichtern. Die meisten Einzelteile haben einen ungefähren Durchmesser von vier Metern, es gibt aber auch kleinere Teile. Die erwünschte Genauigkeit beträgt in etwa drei Meter. Das heißt die durchschnittliche Distanz zwischen der tatsächlichen und der vom Ortungssystem errechneten Position soll bei eben dieser Größe liegen. Das System wird in einer wesentlich kleineren Fabrikhalle getestet.

1.2 Eingesetzte Technik

Da in Innenräumen Signale von außen nur schlecht empfangen werden, sind Bluetooth Sendegeräte, sogenannte Beacons, an der Decke der Testhalle angebracht. Diese senden ihre Identifikationsnummer an ein Empfangsgerät, den Tag, das später an den Fertigungsteilen des Kranherstellers angebracht werden soll. Jenes misst die Signalstärke und leitet diese mit der zugehörigen Identifikationsnummer an ein Gateway weiter. Das Gateway sendet die erhaltenen Informationen zusammen mit der zugehörigen Uhrzeit an einen Server, der dann über Lateration die Position des Tags berechnet.

Lateration ist ein Verfahren, bei dem anhand von Entfernungen der Standort eines Punktes berechnet wird. Betrachtet man die Distanz zwischen dem Tag und einem Beacon als Radius eines Kreises, so befindet sich der Tag auf der Kreislinie und das Beacon im Mittelpunkt des Kreises. Da der Tag von mehreren Beacons umgeben ist und man die Entfernung zu diesen genauso betrachten kann, befindet sich der Tag also genau an dem Punkt, an dem sich alle Kreislinien schneiden [14, S. 7 f.].

Die Beacons senden erst dann ihre Identifikationsnummer an den Tag, wenn sie auf Anfrage von ihm ein Signal erhalten haben, dass er bewegt wurde. So haben die Beacons eine lange Batterielaufzeit. Der Tag und das Gateway nutzen das Long Range Wide Area Network (LoRaWAN). So können beide aufgrund der niedrigen Sendefrequenz, die LoRaWAN nutzt, über große Distanzen hinweg energiesparsam miteinander

kommunizieren. Das Gateway nutzt zur Übermittlung der Daten an den Server ein gängiges LTE-Netz.

1.3 Problematik

Um den Standort eines Tags per Lateration zu bestimmen, muss die Position der Beacons und die jeweilige Distanz zu ihnen bekannt sein. Da die Beacons einen festen Standort haben, kann dieser direkt zur Berechnung genutzt werden. Die Entfernung eines Beacons zum Tag hingegen muss aus der Signalstärke errechnet werden. Da die Signalstärke aber sehr von den bereits erwähnten Störfaktoren abhängt, ist eine Berechnung der Distanz nur unter Miteinbeziehung dieser Faktoren möglich.

Sowohl in der Testhalle als auch in der Halle des Auftraggebers stören vor allem Fertigungsanlagen und Gerüste die Verbindung zwischen den Beacons und dem Tag. Weil diese Objekte einen festen Standort haben, ist es allerdings möglich, die Störungen zu minimieren. Dafür müssen die Signalstärken aller Beacons, die der Tag im gleichen Zeitraum gemessen hat, verglichen werden, um festzustellen, ob sich eines dieser Objekte zwischen Beacon und Tag befindet. Einen passenden Algorithmus dafür zu finden, ist als Mensch kaum möglich.

Bisher wurden schon mehrere manuell erstellte Algorithmen unter großem Zeitaufwand für die Ortung innerhalb der Testhalle geprüft, jedoch hat keiner bisher die gewünschte Genauigkeit erreicht. Zudem wurden nun leistungsstärkere Beacons installiert, weshalb die Algorithmen neu angepasst werden müssen.

2 Aufgabenstellung

Die Aufgabenstellung für diese Praxisarbeit, ist es, herauszuarbeiten, inwiefern sich mehrlagige Perzeptren zur Positionsbestimmung anhand von Signalstärken eignen und wie sie funktionieren.

Zunächst gilt es, sich in Künstliche Neuronale Netze (KNN) einzuarbeiten und ihre einzelnen Bestandteile zu erläutern. Da es Ziel ist, ein mehrlagiges Perzeptron möglichst präzise zu erklären, wird eine umfassende Recherche zu der Funktionsweise gemacht, wobei die populärsten Algorithmen, die eingesetzt werden, genauer beleuchtet werden. Anschließend sollen noch kurz die Hardware-Anforderungen betrachtet werden, um die Herausforderungen, die mit der Anwendung eines KNNs kommen, einordnen zu können.

Die nächste Teilaufgabe besteht darin verschiedene Bibliotheken für Programmierschnittstellen zu vergleichen. Ziel ist es, die Bibliothek, die sich am besten für diese Aufgabe eignet, ausfindig zu machen, da unnötiger Arbeitsaufwand vermieden werden soll, um das Projekt nicht zu umfangreich werden zu lassen. Hierbei wird viel Wert auf das leichte Verständnis der Bibliothek und auf die Aktualität gelegt.

Anschließend soll die Funktionalität eines KNNs, das mit der ausgewählten Bibliothek erstellt wird, anhand von verschiedenen Aufgabentypen getestet werden. Dabei werden hauptsächlich Regressionsaufgaben geprüft, da diese für die Positionsbestimmung aufgrund der geringen Komplexität der Ausgaben des KNNs präferiert werden. Auch eine Klassifizierungsaufgabe wird getestet, um diese der Regression gegenüber zu stellen. Hieraus sollen Schlüsse für die Datenverarbeitung bei der Hauptaufgabe, der Positionsbestimmung, gezogen werden.

Des Weiteren sollen ein Klassifizierungsnetz und ein Regressionstest mit der ausgewählten Bibliothek erstellt und die Genauigkeit ihrer Ausgaben getestet werden. Bei der Genauigkeitsfeststellung werden nur die Ausgaben für den Datensatz, der zum Training genutzt wird, betrachtet.

Anschließend sollen noch die Folgeschritte in einem Ausblick dargelegt werden und es sollen die Erkenntnisse über die Eignung eines Perzeptrons zur Indoor-Lokalisierung in einem Fazit geschildert werden.

3 Künstliche Neuronale Netze in der Theorie

Künstliche Neuronale Netze (KNNs) sind ein Themengebiet des Deep Learnings und bezeichnen Algorithmen, die einer Vorstellung von der Funktionsweise des menschlichen Gehirns nachgeahmt sind.

Sie können mit Datensätzen trainiert werden und anschließend Schätzungen zu neuen Daten zurückgeben. Beispielsweise kann man ihnen Katzen- und Hundebilder mit einer jeweiligen Kennung, um welches Tier es sich handelt, übergeben. Danach werden sie im Optimalfall selbst schätzen können, welches der beiden Tiere auf bisher unbekannten Bildern zu sehen ist [1, S. 27].

Im Folgenden werden die einzelnen Bestandteile eines KNNs erläutert, um schrittweise ihr Zusammenwirken betrachten zu können. Es handelt sich hierbei ausschließlich um mehrlagige Perzeptren, da diese die Grundlage für andere Netze bilden. Rekurrente Netze sowie Faltungsnetze werden in dieser Arbeit nicht thematisiert, da diese für Text- und Spracherkennung beziehungsweise Bilderkennung bestimmt sind und Perzeptren für die Aufgabenstellung ausreichen.

3.1 Tensoren

Unter einem Tensor wird eine nach Achsen und entsprechenden Plätzen auf den Achsen geordnete endliche Menge verstanden. Diese Ordnung ermöglicht es, Tensoren auf beliebige Art und Weise miteinander zu verrechnen. So kann jede Achse, ähnlich zu einem Koordinatensystem, für eine bestimmte Eigenschaft verwendet werden.

Beispielsweise kann ein vierachsiger Tensor zur Beschreibung eines Bildes angelegt werden. Die Achsen dafür würden folgende Eigenschaften der Pixel des Bildes beschreiben: horizontale Position, vertikale Position, Farbraum und Farbtiefe [1, S. 59].

In Programmiersprachen werden Tensoren durch ein- oder mehrdimensionale Arrays dargestellt. So kann jede beliebige Rechenoperation zwischen Elementen der Tensoren ausgeführt werden. Ihr Ergebnis kann anschließend wieder als Skalar in einen Tensor eingetragen werden.

Tensoren haben eine universelle Eigenschaft. Bei einem Wechsel des Koordinatensystems, in dem sie sich befinden, verändert sich nicht die Summe ihrer Elemente [2, S. 37 f.].

3.2 Künstliche Neuronen

Das menschliche Gehirn besteht aus vielen Neuronen, die miteinander über Synapsen kommunizieren. Ähnlich ist ein KNN aufgebaut.

Ein künstliches Neuron ist ein Objekt mit den Attributen Gewichtung W und Schwellwert b. Wenn das Neuron einen Tensor I übergeben bekommt, wird zunächst jedes Element des Tensors gewichtet. Dies geschieht, indem das jeweilige Element i mit dem entsprechenden Element w aus dem Gewichtungstensor multipliziert wird. Die Summe aller so entstandenen Produkte wird nun mit dem Schwellwert b addiert und bildet ein Skalar x [1, S.99]:

$$x = b + \sum_{k=1}^{n} (w_k \cdot i_k), \qquad \forall i: i \in I, \qquad \forall w: w \in W$$

Das Neuron gibt den Wert einer nicht-linearen Aktivierungsfunktion $a(x)$ zurück, die auf die gewichtete Summe x angewandt wurde. Ohne diese Funktion kann das KNN nur lineare Zusammenhänge erkennen, was die Möglichkeiten des Netzes stark beschränken würde [1, S. 102]. Zu den beliebtesten Aktivierungsfunktionen gehören die logistische Funktion $sig(x)$ und der Rectifier ReLU $\max(0, x)$ [3, S. 102].

ReLU gibt die gewichtete Summe x nur zurück, wenn sie im positiven Zahlenbereich liegt. Sonst wird die Zahl Null zurückgegeben [3, S. 104].

Die logistische Funktion ist eine Sigmoidfunktion, weshalb sie oft auch nur Sigmoid genannt wird. Der Graph der Funktion sieht folglich dem Buchstaben „S“ ähnlich. Sie kann Zahlen auf Werte zwischen Null und Eins abbilden, weshalb sie sich gut für stochastische Algorithmen eignet. Die logistische Funktion ist eine Modifikation des exponentiellen Wachstums und nutzt in diesem Sinne die Eulersche Zahl e, um ihre Ableitung $sig'(x)$ simpel zu halten [3, S. 103]:

$$sig(x) = \frac{1}{1 + e^{-x}}, \qquad sig'(x) = sig(x) \cdot (1 - sig(x))$$

Bei der gezeigten Funktion handelt es sich nur um die logistische Funktion, die als Aktivierungsfunktion verwendet wird, es gibt aber auch noch andere logistische Funktionen in der Mathematik.

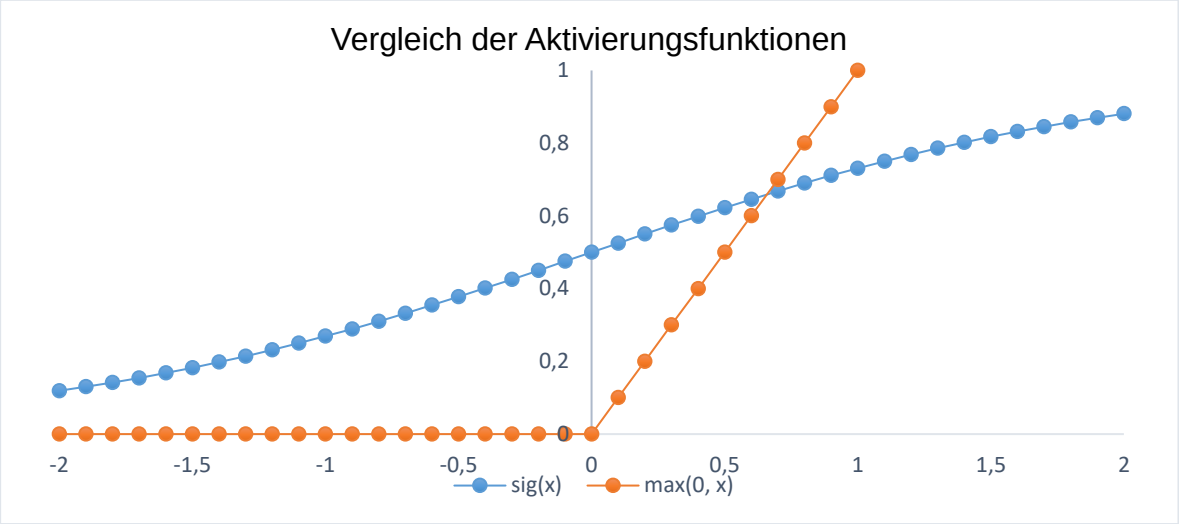

Abb. 1

Eine weitere nützliche Aktivierungsfunktion, die jedoch ausschließlich für die Ausgabeschicht verwendet wird, ist Softmax $sm(x_j)$. Sie ermöglicht eine vollwertige Wahrscheinlichkeitsverteilung, da sie die gewichtete Summe eines Neurons mit denen der anderen Neuronen in Verhältnis stellt [1, S.112]. Softmax verteilt die Wahrscheinlichkeiten auf einer Skala zwischen Null und Eins:

$$sm(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{n}(e^{x_k})}$$

Sie wird auch als Verallgemeinerung der logistischen Funktion gesehen.

Zusammengefasst sind Neuronen also komplexe Funktionen, die einen Tensor auf ein Skalar abbilden und sich über die Attribute Gewichtung und Schwellwert regeln lassen.

3.3 Schichten

Mehrere Neuronen können eine sogenannte Schicht bilden, wenn sie nicht miteinander verbunden sind. Eine Verbindung besteht dann nur zu Neuronen aus anderen Schichten. Da jedes Neuron einen Skalar zurückgibt, können alle Skalare, die eine Schicht

zurückgibt, zu einem Tensor zusammengefasst werden. In dieser Form können sie wieder an Neuronen in der folgenden Schicht übergeben werden. Der Prozess dieser Übergabe wird bildlich als synaptische Verbindung gesehen.

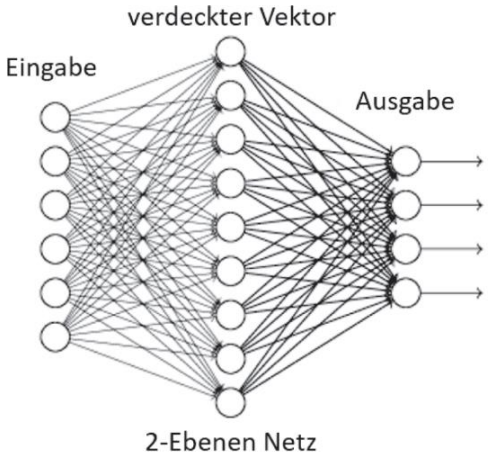

Abb. 2 [4, S. 529]

Die Schicht, in der der ursprüngliche Tensor übergeben wird, heißt Eingabeschicht. Sie dient lediglich der Parameterübergabe und verändert nichts an den ihr übergebenen Werten. Die Schicht, die keinen Tensor mehr an eine weitere Schicht gibt, wird Ausgabeschicht genannt. Alle dazwischenliegenden Schichten werden versteckte Schichten genannt.

Schichten werden aufgrund ihrer vektorähnlichen Struktur auch als Vektoren bezeichnet. Da sie jedoch keine bloße Anordnung von Skalaren sind, ist diese Bezeichnung irreführend.

Mehrere miteinander verknüpfte Schichten bilden bereits ein KNN (siehe Abb. 2). Damit es lernfähig ist, fehlen allerdings noch wichtige Elemente.

3.4 Verlustfunktionen

Bevor das KNN trainiert wird, werden all seine Gewichtungen und Schwellwerte mit Zufallszahlen initialisiert. Wenn ihm jetzt ein Tensor übergeben wird, gibt es einen vollkommen willkürlichen Tensor zurück.

Damit es lernen kann, muss ausgemacht werden, wie stark sich seine Parameter verändern sollen. Hierfür wird eine sogenannte Verlustfunktion $L(\hat{p}, p)$ auf die Ausgabeschicht angewandt. Die Verlustfunktion berechnet aus den prognostizierten Istwerten $\hat{p}$ und den bekannten Sollwerten p einen sogenannten Verlust. Ziel des Trainings ist es nun, dass dieser möglichst nah an die Zahl Null herankommt.

Zu den meistverwendeten dieser Funktionen zählen die mittlere quadratische Abweichung (MSE) und die Kreuzentropie [1, S. 89]. Bevor ihre Funktionsweise betrachtet wird, ist es wichtig, zu verstehen, für welche Fälle sie benötigt werden.

Die mittlere quadratische Abweichung eignet sich gut zur Regression. Sie wird also zur Erkennung von Zusammenhängen zwischen mehreren Variablen verwendet. So kann sie auch zur Berechnung von zweidimensionalen Koordinaten anhand von Signalstärken genutzt werden.

Die Kreuzentropie ist für Klassifizierungsaufgaben gut geeignet. Wird eine Karte in Quadrate aufgeteilt, so kann ein KNN darauf trainiert werden, das Quadrat zu erkennen, in dem sich ein Objekt befindet. Hierfür muss in der Ausgabeschicht die One-Hot-Kodierung verwendet werden. Außerdem soll sich die Aktivierungsfunktion in der Ausgabeschicht für Wahrscheinlichkeiten eignen.

Mathematisch definiert ist die mittlere quadratische Abweichung $MSE(\hat{p}, p)$ als der Mittelwert aller quadrierten Differenzen zwischen Istwert $\hat{p}$ und Sollwert p [5, S. 104]. Die Kreuzentropie $CE(\hat{p}, p)$ hingegen ist als die negative Summe aller Produkte, die aus dem Sollwert p eines Neurons und dem natürlichen Logarithmus des Istwerts $\hat{p}$ entstehen, definiert [5, S. 135].

$$MSE(\hat{p}, p) = \frac{1}{n} \cdot \sum_{k=1}^{n} (p_k - \hat{p}_k)^2 \qquad CE(p, \hat{p}) = -\sum_{k=1}^{n} (p_k \cdot \ln(\hat{p}_k))$$

Zusammengefasst definieren Verlustfunktionen also einen Maßstab, wie sehr sich die Gewichtungen innerhalb eines KNNs ändern sollen. Gleichzeitig kann an ihnen abgelesen werden, wie weit die Ergebnisse des Algorithmus noch vom Zielergebnis abweichen. Nun fehlt nur noch ein Anwendungsverfahren zur tatsächlichen Anpassung der Gewichtungen.

3.5 Optimierer

Als Optimierer werden Algorithmen bezeichnet, die die Gewichtungen W und Schwellwerte b des KNNs verändern. Hierbei handelt es sich um Gradientenabstiegsverfahren. Anders als die Verlustfunktionen $L(\hat{p}, p)$ beziehen sie sich nicht auf die ganze Schicht, sondern auf die einzelnen Elemente in den Attributen der Neuronen. Die geläufigsten Optimierer sind RMSProp (Root Mean Square Propagation) und Adam (Adaptive Movement Estimation) [6, S. 350 f.].

Die Gewichtungselemente w und Schwellwerte b werden verallgemeinert als Parameter θ dargestellt. Bevor sich den beiden Optimierern gewidmet wird, ist es noch wichtig, den Gradienten g_θ eines Neurons zu betrachten:

Der Gradient g_θ ist als partielle Ableitung von der Verlustfunktion $L(\hat{p}, p)$ nach einem Parameter θ definiert. Dies ist möglich, da der prognostizierte Istwert $\hat{p}$ als Aktivierungsfunktion $a(x)$ der gewichteten Summe x aller Eingabeelemente i des Neurons verstanden werden kann. Die Eingabeelemente i können ebenfalls als solche betrachtet werden. So kann der Optimierer bis zur Eingabeschicht, deren Elemente nicht als Aktivierungsfunktion $a(x)$ einer Summe betrachtet werden können, zurückpropagieren.

$$g_\theta = L'_\theta(\hat{p}, p) = L'_\theta(a(x), p) = L'_\theta\left(a\left(b + \sum_{k=1}^{n}(w_k \cdot i_k)\right), p\right)$$

$$\theta \in W \ \vee \ \theta = b$$

Nun wird die Funktionsweise von RMSProp betrachtet:

Als erstes wird der gleitende Mittelwert E der bisherigen quadrierten Gradienten g_θ^2 aktualisiert. Die Konstante β ist eine beliebige Zahl ohne weiterer Bedeutung, meist $0{,}9$.

$$E_\theta \coloneqq \beta \cdot E_\theta + (1-\beta) \cdot g_\theta^2, \qquad 0{,}9 \leq \beta < 1$$

Dann wird der entsprechende Parameter θ aktualisiert. Ziel ist es, den Verlust $L(p, \hat{p})$ zu minimieren. Zum Feinjustieren wird hierbei die Lernrate η benutzt. Je kleiner sie eingestellt ist, desto höher kann die erreichbare Genauigkeit des KNN werden. Je größer sie ist, desto schneller lernt das KNN.

$$\theta \coloneqq \theta - \frac{\eta}{\sqrt{E_\theta}} \cdot g_\theta$$

Da RMSProp zwar aufgrund seiner allgemeinen Anwendbarkeit weit verbreitet ist, jedoch nicht besonders effizient ist, wird nun das ebenfalls universelle Adam betrachtet [7, S. 2]:

Zunächst werden der gleitende Mittelwert $E_{\theta,1}$ der Gradienten g_θ und der gleitende Mittelwert $E_{\theta,2}$ der quadrierten Gradienten g_θ^2 aktualisiert.

$$E_{\theta,1} \coloneqq \beta_1 \cdot E_{\theta,1} + (1-\beta_1) \cdot g_\theta, \quad 0{,}9 \leq \beta_1 < 1$$

$$E_{\theta,2} \coloneqq \beta_2 \cdot E_{\theta,2} + (1-\beta_2) \cdot g_\theta^2, \quad 0{,}99 \leq \beta_2 < 1$$

Anschließend werden die soeben berechneten Werte bereinigt. Die Variable t steht hierbei für die Anzahl an Gradienten, die bereits in die gleitenden Mittelwerte einberechnet wurden.

$$\hat{E}_{\theta,1} = \frac{E_{\theta,1}}{1-\beta_1^t}, \qquad \hat{E}_{\theta,2} = \frac{E_{\theta,2}}{1-\beta_2^t}$$

Zuletzt wird der Parameter θ aktualisiert. Die Konstante ε ist hierbei eine sehr kleine Zahl, deren Zweck darin besteht, die Nullteilerfreiheit zu gewährleisten.

$$\theta \coloneqq \theta - \frac{\eta \cdot \hat{E}_{\theta,1}}{\sqrt{\hat{E}_{\theta,2} + \varepsilon}}$$

Zusammengefasst sind Optimierer Gradientenabstiegsverfahren, die das Lernen ermöglichen. Sie passen die Parameter eines Neurons entgegengesetzt zum Gradienten an, um den Verlust zu minimieren. Hierbei fangen sie in der Ausgabeschicht an und arbeiten sich dann bis zur Eingabeschicht vor. Moderne Optimierer, wie RMSProp und Adam nutzen zudem einen gleitenden Mittelwert vorheriger Gradienten, um die Anzahl an Optimierungsschritten zu minimieren.

3.6 Gesamtbild

Als Künstliches Neuronales Netz wird eine komplexe Funktion bezeichnet, die einen Eingabetensor auf einen Ausgabetensor abbildet. Zudem ist für die Optimierung der Funktion ein ebenso komplexer Algorithmus vorgesehen, der dafür sorgt, dass sich der erzeugte Ausgabetensor dem erwünschten Ausgabetensor annähert.

Als Mensch ist es aufgrund der Masse an Rechnungen nur schwer möglich, die für ein KNN notwendige Rechenleistung zu erbringen. Doch auch für einen Hauptprozessor wird das mit zunehmender Komplexität zur Herausforderung. Darum wird zu solchen Zwecken eine ursprünglich für die Videospielbranche entwickelte Grafikkarte verwendet. Diese kann Rechenoperationen parallel ausführen und ist somit dem Hauptprozessor in der Rechengeschwindigkeit überlegen. Wesentlich leistungsfähiger als die Grafikkarte ist die

2016 speziell für Tensoroperationen entwickelte TPU (Tensor Processing Unit) [1, S. 43 f.]. Ein TPU-Pod kann hundert Billiarden float32-Rechenoperationen pro Sekunde ausführen [8, S. 127].

In diesem Zusammenhang ist es wichtig, die Komplexität des KNNs und die Größe der zum Training eingesetzten Datenmenge abzuschätzen, um die kosteneffizienteste Hardware-Lösung zu ermitteln.

4 Bibliotheken für Programmierschnittstellen

Da das Problem möglichst effizient gelöst werden soll, werden bereits existierende Programmierschnittstellen für Künstliche Neuronale Netze (KNN) verwendet. Um die passende Bibliothek dafür zu finden, werden im Nachfolgenden die Vor- und Nachteile einzelner Bibliotheken verglichen. Die letzten Updates beziehen sich auf den 31.03.2021.

4.1 OpenNN

OpenNN ist eine von der Polytechnischen Universität Katalonien entwickelte Open-Source-Bibliothek für KNN. Geschrieben ist sie für die Programmiersprache C++, wodurch eine vergleichsweise geringe Laufzeit bei der Ausführung eines KNNs gewährleistet werden kann [9, Homepage]. Sie ist sowohl in Form einer ZIP-Datei, als auch in Form einer GZIP-Datei kostenlos erhältlich. Dadurch kann sie auf Unix-artigen Betriebssystemen und Windows benutzt werden. Die neueste Version der Bibliothek wurde am 14.01.2021 veröffentlicht [9, GitHub, Releases] und beinhaltet eine Klasse zum Anlegen eines KNNs und zudem auch verschiedene Optimierer und Verlustfunktionen. Sie kann für Regressions- und Klassifizierungsaufgaben verwendet werden. Es ist mit OpenNN nicht möglich, eine Aktivierungsfunktion auszuwählen [9, Tutorials, OpenNN in 6 steps].

4.2 PyTorch

PyTorch ist ein von Facebook entwickeltes Open-Source-Framework. Es ist für Linux Distributionen, Mac und Windows frei erhältlich. Für die Sprachen Python, C++ und Java stellt es Programmierschnittstellen zur Verfügung. Außerdem kann eine NVIDIA-Grafikkarte zur Ausführung der KNN genutzt werden. In einer seiner Bibliotheken, PyG, sind Schnittstellen speziell für den Umgang mit geometrischen Daten vorhanden [10, Homepage]. Des Weiteren beinhaltet es viele moderne Optimierer, Aktivierungs-und Verlustfunktionen sowie verschiedene Netzstrukturen [10, Tutorials, Build Model]. PyTorch profitiert vor allem von seinen vielen Nutzern. Über zweitausend seiner

Benutzer tragen laut GitHub-Statistiken zur Weiterentwicklung bei. So kam das letzte Update am 10.03.2022 [10, GitHub].

4.3 Caffe

Das von der Universität Kalifornien in Berkeley entwickelte Open-Source-Framework Caffe stellt Schnittstellen für die Programmiersprachen MATLAB, Python und C++ bereit. Auch Caffe ist für Unix-artige Betriebssysteme und Windows erhältlich. Die neueste Version wurde am 18.04.2017 veröffentlicht [11, View On GitHub]. Es enthält viele moderne Optimierer, Verlust- und Aktivierungsfunktionen sowie verschiedene Typen von Schichten. Außerdem ist seine Anwendungsstruktur einfacher, als die von OpenNN oder PyTorch [11, Tutorial Documentation]. Ebenso kann es durch die Kompilierung, wenn C++ genutzt wird, geringe Laufzeiten gewährleisten.

4.4 Keras

Die Open-Source-Bibliothek Keras wurde von François Chollet mit der Absicht entwickelt, maschinelles Lernen für jeden zugänglich zu machen. Sie ist in Python geschrieben und für Linux, Windows und Mac frei erhältlich. Das letzte Mal, als die Keras-Bibliothek gepatcht wurde, war am 03.02.2022. Sie enthält Schnittstellen zum Aufbau eines KNN, zu einzelnen Schichten, Aktivierungs- und Verlustfunktionen, aber auch moderne Optimierer und verschiedene Ausgabefunktionen. Des Weiteren nutzt sie ein Backend namens TensorFlow, welches von Google entwickelt wurde. Sie hat eine sehr intuitiv zu verstehende Anwendungsstruktur. Da zwangsläufig der Python-Interpreter genutzt werden muss, ist jedoch die Laufzeit zum Training eines KNNs vergleichsweise hoch [12].

4.5 Entscheidungsfindung

Stand: 31.03.2022	OpenNN	PyTorch	Caffe	Keras / TensorFlow
letztes Update	14.01.2021	10.03.2022	18.04.2017	03.02.2022
GitHub Sterne	879	55,5k	32,5k	55k / 164k
Mitwirkende	24	2,2k	269	1k / 3,1k
Programmiersprache	C++	Python, C++, Java	Python, C++, MATLAB	Python
Verfügbarkeit	Windows, Unix-Artige	Windows, Unix-Artige, Mac	Windows, Unix-Artige	Windows, Unix-Artige, Mac
bekannte Nutzer	Airbus, Philips, Seat, Thales, Universität Washington	Meta, Amazon, Universität Stanford	Universität Berkeley, Yahoo!	Google, Intel, Twitter, SAP, NVIDIA, NASA, CERN
intuitives Verständnis	-	-	+	+
Umfang	-	+ +	+	+

Da nur 617 Messungen für das Ortungssystem vorliegen, ist die Laufzeit, die das Training des KNNs beansprucht, nicht von hoher Relevanz. Einfach zu verstehende Programmierschnittstellen werden präferiert, um nicht notwendigen Aufwand zu vermeiden. Da Caffe seit dem Erfolg von PyTorch nicht mehr aktualisiert wurde, ist es nicht zukunftsorientiert, den Umgang mit Caffe zu erlernen. PyTorch ist hauptsächlich auf Python ausgelegt. Wenn es jedoch in Python verwendet wird, hat es aufgrund des fehlenden Backend eine höhere Laufzeit als Keras. Die niedrigste Laufzeit hat OpenNN. Da dies jedoch kaum relevant ist und es dafür bei OpenNN an der Auswahl von Aktivierungsfunktionen fehlt, wird im Folgenden die Bibliothek Keras verwendet. Sie ist die einsteigerfreundlichste von den genannten Bibliotheken und sowohl ihre Laufzeit als auch ihr Umfang sind für das Problem angemessen. Außerdem hat sie zusammen mit TensorFlow die meisten Nutzer weltweit.

5 Probeaufgaben

Im Folgenden werden verschiedene Aufgabentypen getestet, um festzustellen, wofür sich ein Künstliches Neuronales Netz (KNN) gut eignet und wofür es nicht geeignet ist. Hieraus werden in einem späteren Kapitel Rückschlüsse gezogen, wie das KNN für die spätere Ortungsaufgabe eingestellt sein muss und wie die Eingabedaten verarbeitet werden müssen. Alle Probeaufgaben werden mit verschiedenen Netzarchitekturen getestet und die jeweils effizienteste Architektur wird genauer beschrieben.

5.1 Das Vielfache einer Zahl

Zunächst wird ein Trainingsarray stellvertretend für einen Tensor mit hundert Zufallswerten zwischen Null und Hundert initialisiert. Das zugehörige Kennungsarray wird mit dem jeweils Doppelten der Werte initialisiert. Als Optimierer wird Adam bei einer Lernrate von 0,001 verwendet. Die Verlustfunktion ist die mittlere quadratische Abweichung. Das KNN besteht aus vier vollständig verbundenen Schichten inklusive der Eingabeschicht, wobei die versteckten Schichten jeweils vier Neuronen und Eingabe- und Ausgabeschicht jeweils ein Neuron besitzen. Es werden keine Aktivierungsfunktionen genutzt, da diese noch nicht nötig sind. Das Netz durchläuft eintausend Optimierungen.

Beim Training kommt das KNN so auf einen Verlust von 0,7. Anschließend wird das trainierte Netz mit fünfzig Zufallszahlen zwischen Eins und Fünfhundert getestet. Die Abweichung zwischen dem vom KNN berechneten Ergebnis und dem Doppelten der Eingabezahl beträgt jeweils etwa ein Prozent des Ergebnisses.

Daraus kann geschlussfolgert werden, dass bereits kleine Neuronale Netze gut das Vielfache einer Zahl berechnen können.

5.2 Addition mit einer unbekannten Konstante

Es wird dieselbe Netzarchitektur wie im vorherigen Test verwendet. Da die Ergebnisse mit eintausend Optimierungen bei diesem Versuch nicht zufriedenstellend sind, wird die Optimierung zweitausendmal wiederholt. Das Trainingsarray bleibt dasselbe, während

das Kennungsarray nun aus den Summen der jeweiligen Zufallswerte mit der Zahl Zehn besteht.

So kommt ein Verlust von 0,0000003 zustande. Im Test liegt die Differenz zwischen den vom KNN berechneten Ergebnissen und den entsprechenden Summen jeweils bei zirka 0,15% der Summe.

Dem ist zu entnehmen, dass ein KNN auch diese Art von Zusammenhängen problemlos lösen kann.

5.3 Quadratzahlen

Das Trainingsarray wird mit eintausend Zufallswerten zwischen Null und Hundert initialisiert. Das Kennungsarray bekommt jeweils die dem Zufallswert entsprechende Quadratzahl zugewiesen. Das Netz besteht aus sieben Schichten, von denen die ersten vier die logistische Funktion zur Aktivierung nutzen. Die erste Schicht nach der Eingabeschicht hat zweiunddreißig, ihre drei Nachfolgeschichten jeweils sechzehn Neuronen. Die sechste Schicht hat wieder zweiunddreißig Neuronen und nutzt die Exponentialfunktion. In der Ausgabeschicht befindet sich ein Neuron mit ReLU-Aktivierung. Diese Netzarchitektur wird gewählt, da die Aufgabe in Versuchen mit anderen Netzwerkarchitekturen nicht lösbar war. Als Lernrate ist 0,0005 eingestellt. Als Optimierer wird Adam mit der mittleren absoluten Abweichung als Verlust gewählt. Sie ist der Mittelwert der Beträge aller Differenzen zwischen Ist- und Sollwert. Das Netz wird in eintausend Durchläufen trainiert.

So kann ein Verlust von 2,8 erreicht werden. Bei Werten innerhalb des Bereichs der Trainingsdaten können im Test gute Ergebnisse mit einer durchschnittlichen Abweichung von 2,4% zu den Quadratzahlen erzielt werden. Die Ergebnisse für Probewerte außerhalb des Trainingsbereichs haben allerdings nichts mit Quadratzahlen gemein.

Da in den vorherigen Versuchen mit anderen Netzarchitekturen auch für Zahlen innerhalb des Trainingsbereichs keine zufriedenstellenden Ergebnisse durch das KNN errechnet wurden, ist anzunehmen, dass Quadratzahlen mit den bestehenden Aktivierungsfunktionen für ein KNN Schwierigkeiten bereiten und daher zu umgehen sind.

5.4 Innere Summe eines Tupels

In diesem Versuch bekommt das Trainingsarray einhundert Tupel aus jeweils zwei Zufallswerten zwischen Null und Hundert zugewiesen. Das Kennungsarray beinhaltet die inneren Summen der entsprechenden Tupel. Das KNN beinhaltet keine Aktivierungsfunktionen und besteht aus vier Schichten. Die Eingabeschicht beinhaltet passend zu den Tupel zwei Neuronen. Die beiden versteckten Schichten haben jeweils acht und die Ausgabeschicht ein Neuron. Zur Optimierung wird wieder Adam zusammen mit der mittleren quadratischen Abweichung genutzt. Die Lernrate ist auf 0,001 gestellt und das KNN durchläuft eintausend Optimierungen.

So wird durch das Training ein Verlust von ungefähr 0,05 erzielt und in Tests liegen die Ergebnisse des KNNs jeweils nur 0,5% von den richtigen Ergebnissen entfernt.

Da die Addition eine wichtige Rolle innerhalb von Neuronen spielt, kommt es zu diesem Ergebnis.

5.5 Inneres Produkt eines Tupels

Für diese Aufgabe wird eine ähnliche Netzarchitektur wie für die Bestimmung von Quadratzahlen verwendet. Der einzige Unterschied liegt in der Eingabeschicht. Dort befinden sich nun anstelle von einem Neuron zwei Neuronen. Diesmal besteht das fünftausendstellige Trainingsarray aus Tupel mit jeweils zwei Zufallswerten zwischen Null und Fünfzig. Das Kennungsarray enthält die jeweiligen inneren Produkte der Tupel. Die Lernrate ist wieder 0,0001 und Adam ist mit der mittleren quadratischen Abweichung als Optimierer eingestellt. Das KNN wird in fünfhundert Durchläufen trainiert.

Obwohl der Verlust bei etwa Drei liegt, kann nur das Produkt aus Zahlen innerhalb des Trainingsbereichs vom KNN berechnet werden. Je größer die Zahlen werden, desto mehr liegt das KNN daneben. Wenn beide Elemente des Tupels größer als Achtzig sind, beträgt die Abweichung vom eigentlichen Produkt bereits über fünfzig Prozent.

KNN sind vermutlich nicht für die Multiplikation zweier unabhängiger Zahlen geeignet.

5.6 Inverses Element

Das Trainingsarray besteht aus viertausend Zufallswerten zwischen Null und Hundert. Hierbei wird das Array so initialisiert, dass es zu gleichen Teilen aus Werten, die kleiner als Eins sind, und größeren Werten besteht. Das Kennungsarray wird mit den zugehörigen Inversen initialisiert.

Nach zwanzig verschiedenen Netzarchitekturen kann keine gefunden werden, mit der im Bereich des Trainingsdatensatzes akzeptable Werte berechnet werden können. Selbst mit sechzehn Schichten aus insgesamt 724 Neuronen kommt das KNN nicht zu einem nützlichen Ergebnis. Durch die Softmax-Funktion in einer mittleren Schicht kann für einen kleinen Teilbereich der Trainingsdatenmenge ein ausreichender Kehrwert bestimmt werden. Dort liegt die Abweichung der Schätzwerte vom inversen Element bei zirka zwanzig Prozent. Außerhalb dieses Teilbereiches liegt sie bei hundert Prozent oder mehr.

Ein vollständig verbundenes KNN kann also keine Inverse berechnen, weshalb die Daten zunächst aufbereitet werden müssen.

5.7 Klassifizierung nach Body-Mass-Index

Zunächst wird ein Trainingsarray mit zweitausend Tupel aus je einem Zufallsgewicht zwischen vierzig und hundertzwanzig Kilogramm und einer zufälligen Körpergröße zwischen 1,4 und 1,9 Metern initialisiert. Zu jedem Tupel wird ein Body-Mass-Index berechnet, aus dem dann anschließend die Klassifizierung im Kennungsarray folgt. Der Body-Mass Index ist ein Quotient mit dem Körpergewicht als Dividend und dem Quadrat aus der Körpergröße als Divisor. Ist der Index kleiner als 18,5, wird dem Kennungsarray das Tupel (1, 0, 0) zugewiesen, was auf Untergewicht hinweist. Ist er größer als Fünfundzwanzig, wird dem Array das Tupel (0, 0, 1) für Übergewicht zugewiesen. Sonst bekommt das Array (0, 1, 0) als Tupel zugewiesen. Das KNN besteht aus fünf Schichten, wobei die erste versteckte Schicht vier Neuronen und den hyperbolischen Tangens als Aktivierungsfunktion nutzt. Die zwei Schichten danach haben jeweils zwölf künstliche Neuronen und werden durch die ReLU-Funktion ausgelöst. In der Ausgabeschicht

befinden sich drei Neuronen und zur Werteverteilung findet die Softmax-Funktion Anwendung. Das Netz wird bei einer Lernrate von 0,0005 zweihundertmal mit Adam trainiert, wobei der Verlust durch die Kreuzentropie berechnet wird.

Bei einem anschließenden Test mit eintausend Werten kann eine Genauigkeit von etwa 97,7 % berechnet werden. Der Body-Mass-Index kann auch als Produkt eines Körpergewichtes mit dem Inversen einer quadrierten Körpergröße betrachtet werden. Demzufolge stehen die Rechenoperationen, die per Regression nicht möglich sind, einer Klassifizierung hingegen nicht im Weg.

5.8 Allgemeine Erkenntnisse

Da alle Probeaufgaben mit verschiedenen Architekturen getestet wurden, konnte festgestellt werden, dass eine Architektur, bei der sich die Anzahl der Neuronen schichtweise gleichmäßig verringert genauer ist, als Architekturen, bei denen das nicht der Fall ist.

Bei allen Durchläufen hat der Verlust nie den Wert Null erreicht, da der Verlust ab einem von der Lernrate abhängigem Stand nicht mehr abgenommen hat.

Außerdem kann aus den Probeaufgaben entnommen werden, dass bei einer kleinen Lernrate der Lernfortschritt anfangs langsam ist, mit der Zeit zunimmt und gegen Ende des Trainingsvorgangs wieder stark abnimmt.

Bei einer großen Lernrate ist der Lernfortschritt insgesamt schneller, der Punkt, an dem der Verlust nicht mehr abnimmt ist aber schon bei einem größeren Verlust erreicht.

Des Weiteren erhöht sich bis zu einem bestimmten Punkt die Genauigkeit, je mehr Schichten und je mehr Neuronen eingesetzt werden.

Bei zu vielen Neuronen erhöht sich die Laufzeit stark und die Ausgaben für Werte außerhalb des Trainingsdatensatzes werden ungenauer und erscheinen in Extremfällen willkürlich.

Zu viele Schichten verlangsamen den Lernfortschritt und tragen dazu bei, dass schon bei einem hohen Verlust der Verlust nicht mehr kleiner wird.

Aus den Aufgaben kann ebenfalls entnommen werden, dass der Punkt, an dem das Training keinen Fortschritt mehr bringt, auch von den Zufallszahlen, mit denen die Gewichtungen am Anfang des Trainings initialisiert werden, abhängt. So können ein Neustart des Trainings und somit auch eine neue zufällige Initialisierung der Attribute jedes Neurons dafür sorgen, dass der Verlust am Ende des Trainings kleiner beziehungsweise größer ist.

6 Anwendung eines Perzeptrons zur Positionsbestimmung

Um ein Künstliches Neuronales Netz (KNN) anhand von Daten aus der echten Welt zu trainieren, müssen zunächst Vorbereitungen gemacht werden. Unter anderem muss ein Datensatz erhoben werden, die Daten verarbeitet werden und die Architektur des KNNs auf den Datensatz zugeschnitten werden. Diese Vorbereitungen werden nun vorgestellt und anschließend werden die Testergebnisse besprochen.

6.1 Erhebung des Trainingsdatensatzes

Zunächst wird eine Tabelle für Messpunkte in der Testhalle angefertigt. Diese beinhaltete jeweils die zweidimensionalen Koordinaten eines Punktes. Die Punkte werden nach den Buchstaben des lateinischen Alphabets benannt. Insgesamt beinhaltet die Tabelle 208 Punkte, also von A0 bis Z8. Die Halle wird anschließend in acht gleich große Bereiche aufgeteilt. In jedem Bereich werden dann Zettel mit den lateinischen Buchstaben auf dem Boden verteilt. Anschließend werden deren Positionen auf den beiden Koordinaten über einen Laser-Entfernungsmesser ermittelt und in die Tabelle eingetragen. Danach werden jeweils die Koordinaten eines Punktes in einer für die Messung entwickelten Applikation eingetragen und der Tag wird auf dem Zettel platziert. Daraufhin wird gewartet, bis die Applikation anzeigt, dass der Server jeweils drei Signalstärken pro Beacon erhalten hat. Danach wird dieser Vorgang mit dem jeweils nächsten Punkt in der Tabelle wiederholt, bis alle 208 Punkte dreimal vermessen wurden. Die Signalstärkewerte der Beacons werden mit den zugehörigen Koordinaten aus der Tabelle gesichert. In der folgenden Abbildung wird kartographisch die Verteilung der Messpunkte innerhalb der Halle dargestellt.

Abb. 3

Wie der Abbildung zu entnehmen ist, sind die Messpunkt nicht optimal verteilt, was zur Folge hat, dass das KNN manche Bereiche schlechter abschätzen kann, als andere. Im Anhang ist ein Ausschnitt der erhobenen Daten zu sehen.

6.2 Vorüberlegungen zur Netzarchitektur

Um die Regression vorbereiten zu können, ist es wichtig, zu wissen, was für eine Art von Daten vorliegen und in was für einem Zusammenhang diese stehen können. In diesem Fall liegen Signalstärken in Form des Indikators RSSI vor. Aus diesen sollen Koordinaten berechnet werden. Nun kann betrachtet werden, wie die Unterschiedlichen Werte zusammenhängen. Zunächst schauen wir uns an, was der RSSI-Wert $RSSI(d)$ mit einer Distanz d zutun hat:

$$RSSI(d) = RSSI(d_0) - 10 \cdot \varphi \cdot \log_{10}\left(\frac{d}{d_0}\right) + X$$

Die Variable d_0 beschreibt hierbei die Entfernung, die zur Kalibrierung des RSSI-Wertes verwendet wurde. Hinzu kommen noch die Ungenauigkeit X und der Wert φ, der die Intensität der Strahlungsdämpfung beschreibt [13, S. 3]. Diese beiden sind für die Aufbereitung der Daten jedoch nicht relevant, da sie unbekannt sind. Wird die Formel nach der Distanz d aufgelöst, sieht sie folgendermaßen aus:

$$d = d_0 \cdot 10^{\left(\frac{X + RSSI(d_0) - RSSI(d)}{10\varphi}\right)}$$

Da die Exponentialfunktion als Aktivierungsfunktion zur Verfügung steht, sollte diese hier auch genutzt werden, denn:

$$10^{\left(\frac{i}{10\cdot\varphi}\right)} = e^{i\cdot\frac{\ln(10)}{10\cdot\varphi}} = e^{i\cdot w}, \qquad w = \frac{\ln(10)}{10\cdot\varphi}$$

Die Variable i stellt hierbei die Eingabe dar, die an das Neuron, welches die Exponentialfunktion nutzt, übergeben wird. Das Gewicht w wird vom Optimierer berechnet. Da die Strahlung in der Halle ortsabhängig unterschiedlich gedämpft wird und es mehrere Beacons gibt, macht es Sinn, eine etwas höhere Anzahl solcher Neuronen in einer Schicht zu benutzen. In der Schicht vor der soeben beleuchteten Schicht wird keine Aktivierungsfunktion benötigt:

$$X + RSSI(d_0) - RSSI(d) = b + w \cdot RSSI(d), \qquad b = X + RSSI(d_0), \qquad w = -1$$

Auch hier sind mehrere Neuronen wichtig, da die Ungenauigkeit X schwanken kann, weshalb es mehrere Schwellwerte b geben sollte.

Da nun eine Architektur zur Distanzberechnung entworfen wurde, fehlt nur noch eine zur Lateration. Nutzt man die Distanz d als Radius eines Kreises innerhalb eines zweidimensionalen Koordinatensystems, so ergibt sich folgende Menge P für alle Punkte innerhalb des Kreises:

$$P = \left\{ \binom{x}{y} \,\middle|\, d^2 \geq (x - x_0)^2 + (y - y_0)^2 \right\}$$

Hierbei stehen x_0 und y_0 für die Koordinaten des Mittelpunktes. Der Punkt $\vec{t}$, an dem sich der Tag befindet, muss sich unter Berücksichtigung der Ungenauigkeit in der Schnittmenge aller Kreise befinden:

$$\vec{t} \in P_1 \cap P_2 \cap \cdots \cap P_n$$

Da die Testhalle achtundzwanzig Meter lang ist, unterscheidet sich die Summe aus den Koordinatenwerten x und y um maximal acht Meter von der aus ihnen berechenbaren Hypotenusenlänge. Weil das KNN keine quadratischen Gleichungen lösen kann, ist also mit dieser maximalen Ungenauigkeit zu rechnen.

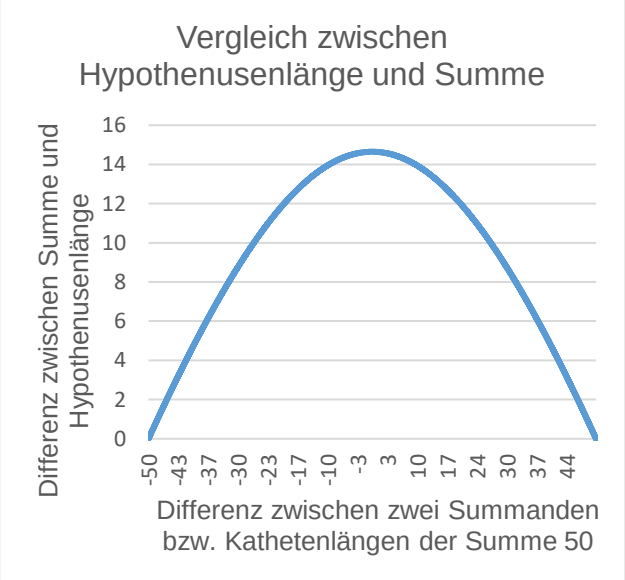

Abb. 4

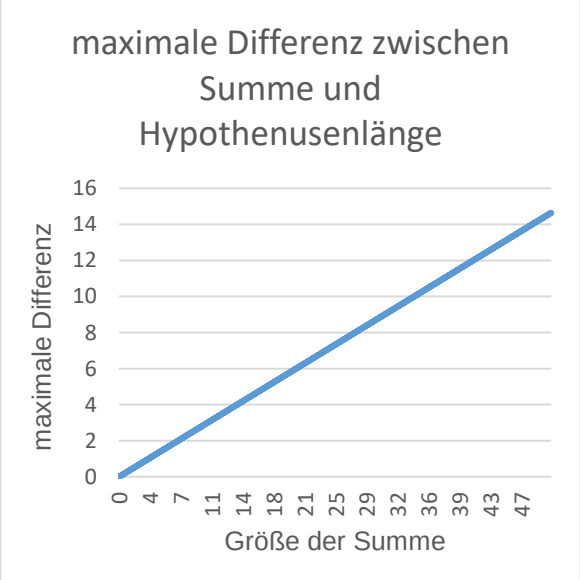

Abb. 5

Aus dem ersten Diagramm (Abb. 4) ergibt sich, dass sich die Hypotenusenlänge bei zwei gleich großen Kathetenlängen am größten von der Summe dieser unterscheidet. Im zweiten Diagramm (Abb. 5) sieht man die auf dieser Tatsache aufbauenden maximalen Abweichungen für verschiedene Summengrößen.

Da in der Halle jedoch sechs Beacons angebracht sind, ist es wahrscheinlich, dass die tatsächliche Abweichung zwischen dem Standort und der prognostizierten Position geringer ist. Zur Filterung von Störfaktoren und zur Eruierung eines Punktes in der Schnittmenge können ReLU-Schichten verwendet werden.

Da die Werte durch die Verwendung der Exponentialfunktion außerhalb des Wertebereichs herkömmlicher Datentypen liegen können, muss eine Normierungsschicht benutzt werden. Diese subtrahiert von jedem Wert x_i den Mittelwert μ und dividiert die Differenz anschließend durch die Standardabweichung σ:

$$\acute{x}_i = \frac{x_i - \mu}{\sigma}\,, \qquad \mu = \frac{1}{n} \cdot \sum_{k=1}^{n} (x_k)\,, \qquad \sigma = \sqrt{\frac{1}{n} \cdot \sum_{k=1}^{n} ((x_k - \mu)^2)}$$

Mittelwert μ und Standardabweichung σ werden für den Trainingsdatensatz bestimmt und anschließend als Konstanten gespeichert. Für unbekannte Daten nach dem Training werden dieselben beiden Konstanten zur Normierung verwendet.

6.3 Einlesen des Datensatzes

Es liegt eine Excel-Tabelle mit den benötigten Daten vor. Aus ihr werden die RSSI-Werte und die Koordinaten jeweils in eine neue Excel-Tabelle kopiert. Aus den beiden Excel-Tabellen werden die Überschriften gelöscht. Nun müssen beide Tabellen noch auf Vollständigkeit überprüft werden, um einen Fehler beim Training zu vermeiden. Anschließend werden beide jeweils als CSV-Datei (Comma Seperated Values) gespeichert.

Da Keras neben Listen auch sogenannte Numpy-Arrays als Datentyp nutzt, werden die beiden Dateien über die Methode `numpy.genfromtxt()` als Arrays eingelesen. Nun

können sie je nach Anwendungsfall direkt zum Training verwendet werden oder über die Methode `numpy.ndarray.tolist()` in eine Liste konvertiert werden.

6.4 Aufbau des Regressionsnetzes

Das KNN, das zur Regression verwendet wird, besteht aus zehn Schichten inklusive der Normierungsschicht.

Die ersten beiden Schichten nach der Eingabeschicht nutzen die ReLU-Funktion, wobei die erste Schicht 192 Neuronen und die Nachfolgeschicht 96 Neuronen besitzt.

In der dritten Schicht wird die Exponentialfunktion zur Berechnung von Entfernungen anhand der RSSI-Werte genutzt. Diese Schicht beinhaltet ebenso viele Neuronen. In ihren sechs Nachfolgeschichten wird wieder die ReLU-Funktion verwendet, um aus den zunächst zusammenhangslos wirkenden Entfernungen eine Position herauszufiltern. Die erste dieser ReLU-Schichten besitzt 64 Neuronen. Diese Anzahl halbiert sich in jeder weiteren Schicht, sodass in der Ausgabeschicht nur noch zwei Werte berechnet werden.

Die hohe Anzahl an Neuronen wird gewählt, da sich viele Störfaktoren in der Testhalle befinden, die jeweils individuell behoben werden müssen. Die Anzahl an Schichten ergibt sich aus mehreren Versuchen, in denen die Anzahl der Schichten sowie die Anzahl der Neuronen erhöht oder verkleinert wurden.

Die Architektur, bei der der niedrigste Verlust am Ende des Trainings auftritt, ist die, die hier geschildert wird.

Dieses Netz wird in zweitausend Durchläufen bei einer Lernrate von 0,0001 mit dem Optimierer Adam trainiert. Die Anzahl der Durchläufe wurde angepasst an den Moment, an dem sich der Verlust nicht mehr verändert. Außerdem konnte durch das Testen verschiedener Einstellungen der Lernrate dieser Moment so beeinflusst werden, dass er erst ab einem kleinen Verlust auftritt.

In Abbildung sechs ist der Code des KNNs in Python zu sehen.

```
normalize = keras.layers.Normalization()
normalize.adapt(training_input)

model = keras.Sequential([
        normalize,
        keras.layers.Dense(units = 192, input_shape = [6], activation = 'relu'),
        keras.layers.Dense(units = 96, activation = 'relu'),
        keras.layers.Dense(units = 96, activation = 'exponential'),
        keras.layers.Dense(units = 64, activation = 'relu'),
        keras.layers.Dense(units = 32, activation = 'relu'),
        keras.layers.Dense(units = 16, activation = 'relu'),
        keras.layers.Dense(units = 8, activation = 'relu'),
        keras.layers.Dense(units = 4, activation = 'relu'),
        keras.layers.Dense(units = 2, activation = 'relu')
])

model.compile(
        optimizer = keras.optimizers.Adam(learning_rate = 0.0001),
        loss = 'mean_absolute_error'
)

model.fit(training_input, training_labels, epochs = 2000)
```

Abb. 6

6.5 Ergebnisse des Regressionsnetzes

Bei dem Trainingsdatensatz können zuverlässige Ergebnisse erzielt werden. So liegen in etwa fünfundfünfzig Prozent der berechneten Koordinaten weniger als einen halben Meter vom tatsächlichen Punkt entfernt. Siebenundachtzig Prozent der Ergebnisse liegen weniger als drei Meter vom Standort des Tags entfernt. Problematisch ist allerdings, dass die Verfehlungen um mehr als fünf Meter immer noch einen Anteil von sechs Prozentpunkten ausmachen. Die Entfernungen werden durch den Satz des Pythagoras berechnet und ihre Anteile beim Testen durch eine „if“-Verzweigung ausfindig gemacht:

```
prediction = model.predict(test_input)

def distance(pred, label):
        sqrd_cath_1 = pow((pred[0] - label[0]), 2)
        sqrd_cath_2 = pow((pred[1] - label[1]), 2)
        return pow((sqrd_cath_1 + sqrd_cath_2), 0.5)

length = len(prediction)
amount = [0, 0, 0, 0, 0, 0, 0]

for i in range(0, length):
        d = distance(prediction[i], test_labels[i])
        if d < 0.5:
                amount[0] += 1
        elif d < 1:
                amount[1] += 1
        elif d < 2:
                amount[2] += 1
        ...
        else:
                amount[6] += 1

for i in range(0, 7):
        amount[i] = 100 * amount[i] / length
print("Verteilung: " + str(amount))
```

Abb. 7

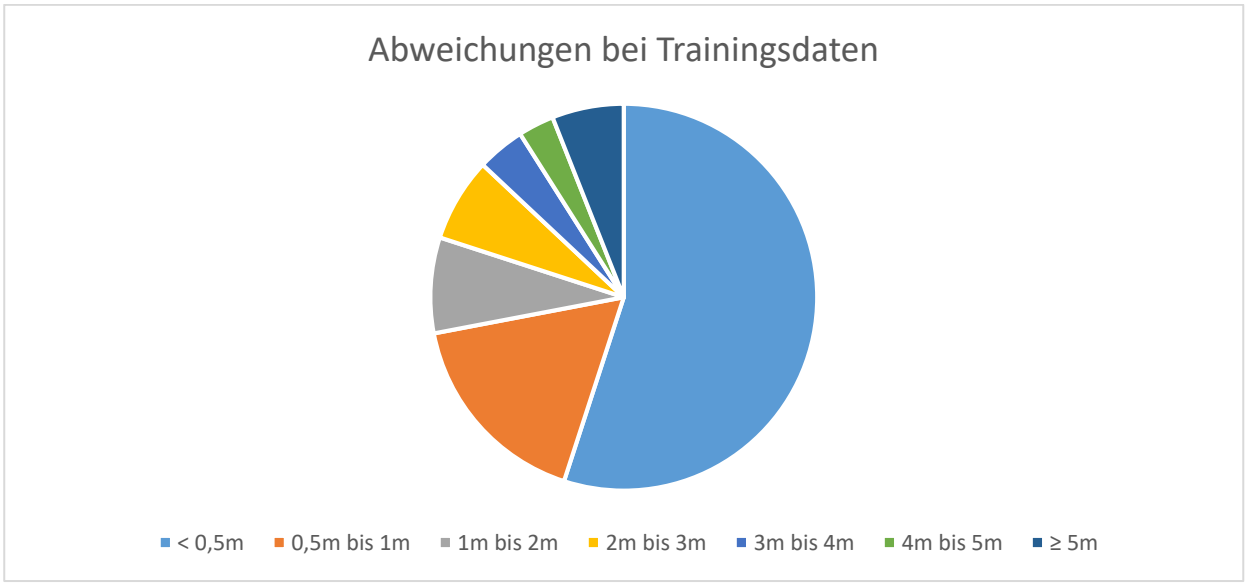

Abb. 8

6.6 Vorbereitung der Daten auf das Klassifizierungsnetz

Für das Klassifizierungsnetz müssen zunächst die gemessenen Daten klassifiziert werden. Die Trainingshalle kann hierfür in zweiunddreißig gleich große Teile aufgeteilt werden:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Abb. 9

Da die Felder jeweils eine feste Länge auf der Y-Achse und eine feste Breite auf der X-Achse haben, lassen sich alle Messpunkte einem Feld zuordnen. Zur Klassifizierung wird jeder Liste von RSSI-Werten eine zweiunddreißigstellige Liste mit einer Eins an der dem Feld entsprechenden Indexposition und Nullen an allen anderen Indexpositionen zugeordnet. So kann das Netz später eine Normalverteilung ausgeben. Realisierbar ist diese Klassifizierung mit einer Verzweigung, also mehreren „if“-Anweisungen.

6.7 Aufbau des Klassifizierungsnetzes

Das Klassifizierungsnetz besteht inklusive der Eingabeschicht aus nur fünf Schichten. In der ersten Schicht nach der Eingabeschicht befinden sich 256 Neuronen. In der Nachfolgeschicht wird diese Zahl jeweils halbiert, sodass sich in der Ausgabeschicht zweiunddreißig Neuronen befinden.

In allen Schichten außer der Ausgabeschicht wird ReLU als Aktivierungsfunktion verwendet. In der Ausgabeschicht ist es Softmax, da so eine Normalverteilung garantiert wird.

Optimiert wird das Netz durch Adam mit der Kreuzentropie als Verlustfunktion. Die Lernrate ist auf 0,0001 gesetzt und das Netz wird in zweitausend Durchläufen trainiert.

Zuletzt gibt das Netz während dem Training noch die aktuelle Genauigkeit heraus. So kann kontrolliert werden, ob sich das Netz tatsächlich dem Trainingsziel nähert und wie schnell es das tut.

```
normalize = keras.layers.Normalization()
normalize.adapt(training_input)

model = keras.Sequential([
        normalize,
        keras.layers.Dense(units = 256, input_shape = [6], activation = 'relu'),
        keras.layers.Dense(units = 128, activation = 'relu'),
        keras.layers.Dense(units = 64, activation = 'relu'),
        keras.layers.Dense(units = 32, activation = 'softmax')
])

model.compile(
        optimizer = keras.optimizers.Adam(learning_rate = 0.0001),
        loss = 'categorical_crossentropy',
        metrics = 'accuracy'
)

model.fit(training_input, training_labels, epochs = 2000)
```

Abb. 10

6.8 Ergebnisse des Klassifizierungsnetzes

Für jede Liste an RSSI-Werten gibt dieses KNN eine Normalverteilung in Form eines Numpy-Arrays aus. Über eine Funktion der Bibliothek Numpy kann die Indexposition des Maximalwertes aus der Normalverteilung ausgegeben werden.

Nach einem Abgleich mit den Indexpositionen für die im Kennungsarray der Wert Eins eingesetzt wurde, kann festgestellt werden, dass alle vom KNN bestimmten Felder, auf denen sich der Tag befinden soll, mit den Feldern, auf dem er sich tatsächlich befand, übereinstimmen:

```
prediction = model.predict(training_input)

right_predictions = 0
length = len(prediction)

for i in range(0, length):
        if np.argmax(prediction[i]) == np.argmax(training_labels[i]):
                right_predictions += 1

print("Genauigkeit: " + str(100 * right_predictions / length) + " %")
```

Abb. 11

```
max@Computer:~/Python$  python3 ./klassifizierung_test.py
Genauigkeit: 100 %
```

Abb. 12

7 Ausblick

Der nächste Schritt ist es, die beiden Perzeptren in der Testhalle mit willkürlichen Positionen zu testen.

Fallen die Tests gut aus, so ist der nächste Schritt, die für das Ortungssystem nötige Hardware in der Halle des Auftraggebers zu installieren.

Danach wird die Halle des Auftraggebers vermessen. Hierbei wird ein großes Koordinatensystem mit einer Tabelle für die Messungen der Signalstärkewerte angelegt. In der Tabelle werden die horizontalen und die vertikalen Koordinaten für gleichmäßig verteilte Punkte in der Halle stehen. Es wird eine höhere Messpunktdichte nötig sein, um die Präzision der Standortvorhersagen zu verbessern. Außerdem werden mehr Messungen der Signalstärkewerte pro Punkt die Genauigkeit ebenfalls erhöhen.

Das Regressions-KNN für diese Halle wird mehr Schichten und mehr Neuronen, als das der Testhalle brauchen. Ein Klassifizierungsnetz kommt bei der Größenordnung der Halle nicht in Frage, da die Bereiche, die es identifiziert maximal drei Meter lang und drei Meter breit sein dürften. Da die Halle zudem auch noch größer ist und die Bereiche zur Klassifizierung in der One-Hot-Codierung ausgegeben werden müssen, würde die Komplexität der Ausgaben einen zu hohen Arbeitsaufwand verursachen.

Das Training eines KNNs mit diesem Umfang an Messungen als Trainingsdaten wird sehr Zeitintensiv. Deshalb ist es in diesem Falle sinnvoll, das Training in einer Cloud zu vollziehen.

Nach dem Training wird das KNN in eine Website implementiert, auf der dann eine Karte von der Fabrikhalle zeigen wird, wo sich welcher Tag befindet. Außerdem wird man Tags ein- und ausblenden können, man kann den Akkustand der Tags sehen und man wird die Tags auf der Website umbenennen können, in den Gegenstand, auf dem sich der Tag befindet.

8 Fazit

Da bisher nicht die Möglichkeit bestand, die beiden Künstlichen Neuronalen Netze mit echten unbekannten Daten zu testen, kann keine genaue Aussage über die Funktionalität gemacht werden.

Sicher ist jedoch, dass beide Netze in den Tests, die gemacht werden konnten, genauer waren, als die manuell erstellten Algorithmen, die zuvor getestet wurden.

Um das Regressionsnetz zu verbessern, sind vermutlich mehr Messpunkte, die zum Training eingesetzt werden können, nötig. Die Messungen beanspruchen allerdings viel Zeit und Arbeit.

Auch trägt eine gleichmäßigere Messpunktverteilung zu einem besseren Ergebnis bei, da das Perzeptron Daten, die in einem Bereich liegen, auf das es trainiert wurde, besser zuordnen kann.

Ob KNN für Ortungssysteme verwendet werden sollen, hängt davon ab, wie genau die Ortung sein soll und wie viel Arbeitszeit investiert werden soll. Außerdem ist es von der gegebenen Technologie und der Räumlichkeit abhängig, wie bei dem Versuch zu sehen ist.

Je spezifischer der Anwendungsort eines Ortungssystems ist, desto empfehlenswerter ist die Nutzung von Machine Learning. Für sehr flächendeckende Systeme, wie GPS oder Galileo, sind größere Ungenauigkeiten in spezifischen Situationen nicht relevant. Es ist zu viel Aufwand, für jede Großstadt mit Hochhäusern ein KNN zu trainieren. Außerdem sind solche Systeme meist weniger störungsanfällig, da sie anstelle von Signalstärken die Zeitstempel einer Atomuhr zur Berechnung der Position nutzen [15].

Dies ist bei den kurzen Entfernungen eines Indoor-Ortungssystems bisher nicht möglich. Die Anschaffung einer Atomuhr zu diesem Zweck ist nicht wirtschaftlich und die Strecken sind zu kurz, um einen ausreichenden Zeitunterschied festzustellen. Deshalb müssen grobe Störungen, wenn es sie gibt, anders behoben werden, wobei ein KNN helfen kann.

Literaturverzeichnis

[1] **Chollet, F. / Lorenzen, K. (2018):** Deep Learning mit Python und Keras. Das Praxis Handbuch vom Entwickler der Keras-Bibliothek. Frechen 2018

[2] **Haas, A. (1922):** Vektoranalysis. In ihren Grundzügen und wichtigsten physikalischen Anwendungen. Berlin / Leipzig 1922 (Nachdr. Berlin / Boston 2021), https://doi.org/10.1515/9783112392287, Einsicht am 25.03.2022

[3] **Deshpande, A. / Kumar, M. (2018):** Artificial Intelligence for Big Data. Complete Guide to Automating Big Data Solutions Using Artificial Intelligence Techniques. Birmingham 2018

[4] **Görz, G. / Schmid, U. / Braun, T. (2020):** Handbuch der Künstlichen Intelligenz. 6. Auflage, Berlin / Boston 2020, https://doi.org/10.1515/9783110659948, Einsicht am 28.03.2022

[5] **Dawani, J. (2020):** Hands-On Mathematics for Deep Learning. Build a solid mathematical foundation for training efficient deep neural networks. Birmingham 2020

[6] **Bonaccorso, G. (2018):** Mastering Machine Learning Algorithms. Expert Techniques to Implement Popular Machine Learning Algorithms and Fine-Tune Your Models. Birmingham 2018

[7] **Kingma, D. / Ba, J. (2015):** Adam. A Method for Stochastic Optimization. Ithaca (New York) 2015, https://hdl.handle.net/11245/1.505367, Einsicht am 30.03.2022

[8] **Paper, D. (2021):** State-of-the-Art Deep Learning Models in TensorFlow. Modern Machine Learning in the Google Colab Ecosystem. New York 2021, https://doi.org/10.1007/978-1-4842-7341-8, Einsicht am 30.03.2022

[9] **o. V. (o. J.):** OpenNN offizielle Webseite. https://www.opennn.net, Einsicht am 31.03.2022

[10] **o. V. (o. J.):** PyTorch offizielle Webseite. https://pytorch.org, Einsicht am 31.03.2022

[11] **o. V. (o. J.):** Caffe offizielle Webseite. https://caffe.berkeleyvision.org, Einsicht am 31.03.2022

[12] **o. V. (o. J.):** Keras offizielle Webseite. https://keras.io/about, Einsicht am 31.03.2022

[13] **Naghdi, S. / O'Keefe, K. (2019):** Trilateration With BLE RSSI Accounting for Path Loss Due to Human Obstacles. Pisa, 2019, https://doi.org/10.1109/ipin.2019.8911816, Einsicht am 12.04.2022

[14] **Tost, F. (2009):** Signalstärkebasierte Ortung. Ein Beitrag zur probabilistischen, symbolischen, zellgenauen Ortung mobiler Netzwerkknoten innerhalb von Gebäuden. Cottbus 2009, https://nbn-resolving.org/urn:nbn:de:kobv:co1-opus-12421, Einsicht am 26.08.2022

[15] **Engler, E. (2008):** Funktionsweise und Status Globaler Navigationssatellitensysteme. Dresden 2008, https://elib.dlr.de/54196/1/2008-05-23-GNSS-Statusr1a.pdf, Einsicht am 01.09.2022

Anhang

A) Ausschnitt aus erhobenen Daten, die zum Training der KNN verwendet wurden.

Koordinaten		RSSI-Werte					
X in m	Y in m	Beacon 11575	Beacon 11578	Beacon 11580	Beacon 11583	Beacon 11585	Beacon 11587
1,23	2,45	-67	-69	-72	-68	-71	-61
1,23	2,45	-82	-69	-72	-69	-68	-63
1,23	2,45	-83	-67	-73	-69	-68	-63
2,7	2,45	-70	-67	-71	-76	-69	-64
2,7	2,45	-70	-68	-68	-77	-70	-64
2,7	2,45	-70	-62	-68	-83	-71	-64
4,2	2,45	-67	-67	-70	-73	-71	-56
4,2	2,45	-66	-68	-68	-72	-74	-56
4,2	2,45	-65	-70	-70	-72	-76	-56
5,5	2,45	-70	-66	-62	-67	-68	-61
5,5	2,45	-70	-66	-67	-67	-67	-67
5,5	2,45	-64	-67	-62	-67	-68	-67
6,84	2,45	-67	-67	-67	-67	-67	-56
6,84	2,45	-67	-67	-68	-65	-67	-55
6,84	2,45	-66	-66	-70	-74	-69	-55
8,45	2,45	-67	-62	-63	-65	-70	-62
8,45	2,45	-65	-56	-68	-70	-67	-65
8,45	2,45	-66	-57	-66	-69	-66	-65
9,7	2,45	-68	-62	-69	-71	-68	-67
9,7	2,45	-70	-56	-71	-69	-71	-67
9,7	2,45	-68	-56	-67	-73	-69	-67
9,7	3,45	-64	-64	-64	-65	-67	-66
9,7	3,45	-67	-68	-66	-65	-68	-65
9,7	3,45	-68	-72	-64	-61	-67	-66
8,45	3,45	-67	-62	-68	-70	-69	-54
8,45	3,45	-71	-63	-69	-69	-69	-59
8,45	3,45	-75	-68	-78	-70	-69	-57
1,9	8,3	-63	-69	-63	-66	-68	-61
1,9	8,3	-64	-66	-71	-72	-61	-62
1,9	8,3	-64	-61	-74	-71	-63	-62