HOCHSCHULE COBURG

Hochschule für angewandte Wissenschaften Coburg

Fakultät Elektrotechnik und Informatik

Studiengang: Informatik (M. Sc.)

Masterarbeit

# Verwendung von Positionsdaten zur automatisierten Klassifizierung von Verkehrsteilnehmern mittels maschinellen Lernverfahren

Dietmar Fischer

Abgabedatum: 31. März 2023

Betreut durch: Prof. Dr. Thomas Wieland

# Inhaltsverzeichnis

**Inhaltsverzeichnis** ..... **2**

**Abbildungsverzeichnis** ..... **6**

**Tabellenverzeichnis** ..... **8**

**Codeverzeichnis** ..... **9**

**1 Einleitung** ..... **10**

1.1 Einführung und wissenschaftliche Fragestellung ..... 10

1.2 Projekthintergrund ..... 11

1.3 Forschungsfragen ..... 11

**2 Verwandte und vorherige Arbeiten** ..... **13**

2.1 Verwandte Arbeiten ..... 13

2.1.1 [Simoncini+2018] ..... 13

2.1.2 [Althoff+2009] ..... 13

2.2 Vorherige Arbeiten ..... 14

2.2.1 [Torlak 2022] ..... 14

2.2.2 [Sohl 2022] ..... 15

2.2.3 [Opitz 2022] ..... 16

**3 Grundlagen** ..... **17**

3.1 CARLA ..... 17

3.2 Neuronale Netze ..... 18

3.2.1 Einführung ..... 18

3.2.2 Aufbau eines neuronalen Netzes ..... 19

3.2.3 Aktivierungsfunktionen ..... 20

3.2.4 Backpropagation ..... 21

3.2.5 Mathematische Betrachtung Backpropagation ..... 22

3.2.6 Findung von optimalen Hyperparametern ..... 23

3.2.7 Weitere Arten von neuronalen Netzen ..... 24

3.3 Reinforcement Learning ..... 25

3.3.1 Einführung ..... 25

3.3.2 Markov Decision Process ..... 25

3.3.3 Mathematische Betrachtung ..... 26

3.3.4 Belohnungen ..... 27

3.3.5 Policies und Value Functions ..... 29

3.3.6 Findung von optimalen Policies ..... 30

3.4 Reinforcement Learning – Q Learning ... 31
3.4.1 Epsilon Greedy Strategy ... 31
3.4.2 Bellman-Gleichung ... 32
3.4.3 Aktualisieren der Q-Values ... 33
3.4.4 Deep Q-Learning ... 33
3.5 K-Means Clustering ... 34
3.6 Weitere Verwendete Funktionen ... 37
3.6.1 Time to Collision (TTC) ... 37
3.6.2 Bremsweg ... 37
3.7 Verwendete Technologien und Libraries ... 38
3.7.1 CSV ... 38
3.7.2 Python ... 38
3.7.3 Tensorflow ... 39
3.7.4 Matplotlib ... 40
3.7.5 Numpy ... 40
**4 Anforderungen und Vision ... 42**
4.1 Verkehrsteilnehmerklassifikation ... 42
4.2 Relevanzklassifikation ... 43
4.2.1 Definition ... 43
4.2.2 Vorstellung der Vision ... 44
4.2.3 Vorstellung der Geschwindigkeitsprognose ... 45
**5 Datengewinnung und Weiterverarbeitung ... 48**
5.1 Datengewinn über CARLA ... 48
5.2 Gewinn von echten Verkehrsdaten ... 50
5.3 Betrachtung der Ausgangsdaten ... 51
5.4 Weiterverarbeitung für Verkehrsteilnehmerklassifikation ... 53
5.5 Weiterverarbeitung für Klassifikation von Risikoklassen ... 54
5.6 Weiterverarbeitung für die Geschwindigkeitsprognose ... 55
**6 Entwicklung Reinforcement Learning ... 57**
6.1 Anforderungsfestlegung Agenten ... 57
6.2 Anforderungsfestlegung Umgebung ... 58
6.3 Konzeption des Agenten ... 59
6.3.1 Zusammenfassung der resultierenden Klasse ... 61
6.4 Konzeption der Umgebung ... 62
6.4.1 Zusammenfassung der resultieren Klasse ... 64

6.5 Implementierung der Klasse Agent ... 65
6.5.1 init() ... 65
6.5.2 act() ... 66
6.5.3 learn() ... 67
6.6 Implementierung der Klasse Umgebung ... 68
6.6.1 init() ... 69
6.6.2 step() ... 69
6.6.3 reset() ... 70
**7 Entwicklung normales Feed Forward Netz ... 71**
7.1 Implementierung ... 72
**8 Verkehrsteilnehmerklassifikation ... 74**
8.1 Festlegung der optimalen Hyperparameter ... 74
8.2 Parameterübergabe Reinforcement Learning ... 75
8.3 Parameterübergabe Gewöhnliches Feed Forward Netz ... 77
**9 Relevanzklassifikation ... 79**
9.1 Bildung von Risikoklassen ... 79
9.2 Entwicklung des Klassifikationsmodells ... 81
9.2.1 Festlegung der optimalen Hyperparameter ... 81
9.2.2 Implementierung Reinforcement Learning ... 83
9.2.3 Implementierung Gewöhnliches Feed Forward Netz ... 84
9.3 Modell zur Prognose von Verkehrsteilnehmeraktion ... 85
9.3.1 Entwicklung der Modi ... 86
9.3.2 Festlegung der optimalen Hyperparameter ... 86
9.3.3 Implementierung ... 87
**10 Evaluation und Diskussion ... 90**
10.1 Verkehrsteilnehmerklassifikation ... 90
10.1.1 Evaluation mit simulierten Validierungsdaten auf simulativ trainierten Modellen ... 90
10.1.2 Evaluation mit realen Validierungsdaten auf simulativ trainierten Modellen ... 92
10.1.3 Evaluation mit realen Validierungsdaten auf real trainierten Modellen ... 95
10.2 Risikoklassen-Klassifikation ... 97
10.2.1 Reinforcement Learning ... 97
10.2.2 Gewöhnliche neuronale Netze ... 98
10.3 Geschwindigkeitsprognose ... 100
10.3.1 Evaluation pro Verkehrsteilnehmerklasse ... 100
10.3.2 Evaluation pro Modus ... 101

**11 Zusammenfassung und Ausblick ..... 103**

**Literaturverzeichnis ..... 106**

**Ehrenwörtliche Erklärung ..... 108**

# Abbildungsverzeichnis

Abbildung 2.1: Interesse im zeitlichen Verlauf ChatGPT [Google-Trends 2023] ..... 18

Abbildung 2.2: Aufbau eines neuronalen Netzes [Dörn 2023] ..... 19

Abbildung 2.3: Neuronales Netz mit Werten und Gewichten [Dörn 2023] ..... 20

Abbildung 2.4: Agent Umgebung Interaktion in einem MDP [Sutton+2018] ..... 26

Abbildung 2.5: Mögliches Ergebnis eines Clustering Verfahrens [Han+2001] ..... 35

Abbildung 2.6: Iterationsschritt K-Means Clustering [Von der Hude 2020] ..... 36

Abbildung 3.1 Vision der Verkehrsteilnehmerklassifikation ..... 42

Abbildung 3.2 Vision der Relevanzklassifikation ..... 45

Abbildung 3.3 Vision der Geschwindigkeitsprognose ..... 46

Abbildung 4.1: Ablaufdiagram der CALRA Datenerfassung [Torlak 2022] ..... 49

Abbildung 4.2: Bildschirmaufnahme aus der App für die Sammlung von realen Daten ..... 50

Abbildung 5.1: Die Agentenklasse visualisiert ..... 62

Abbildung 5.2: Die Umgebungsklasse visualisiert ..... 65

Abbildung 7.1: Grid search Vekehrsteilnehmerklassifikation ..... 74

Abbildung 7.2: Architektur neuronales Netz der Verkehrsteilnehmerklassifikation ..... 75

Abbildung 8.1: Ellenbogenmethode für die Risikoklassen ..... 79

Abbildung 8.2: Ergebnis Clustering Risikoklassen ..... 80

Abbildung 8.3: Grid Search Risikoklassen-Klassifikation ..... 82

Abbildung 8.4: Architektur des neuronalen Netzes für die Risikoklassen-Klassifikation ..... 82

Abbildung 8.5: Grid search Geschwindigkeitsprognose ..... 87

Abbildung 9.1: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit simulierten Validierungsdaten auf simulativ trainiertes Reinforcement Learning Modell ..... 90

Abbildung 9.2: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit simulierten Validierungsdaten auf simulativ trainiertes gewöhnliches neuronales Netzwerk ..... 91

Abbildung 9.3: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf simulativ trainiertes Reinforcement Learning Modell ..... 93

Abbildung 9.4: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf simulativ trainiertes gewöhnliches neuronales Netzwerk ..... 94

Abbildung 9.5: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf real trainiertes Reinforcement Learning Modell ..... 95

Abbildung 9.6: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf real trainiertes gewöhnliches neuronales Netz ..... 96

Abbildung 9.7: Risikoklassen-Klassifikation Präzisionsmatrix Reinforcement Learning ..... 97

Abbildung 9.8: Risikoklassen-Klassifikation Präzisionsmatrix anhand von gewöhnlichen neuronalen Netzen ..... 99

Abbildung 9.9: Geschwindigkeitsprognose durchschnittliche Abweichung pro Klasse ..... 100

Abbildung 9.10: Geschwindigkeitsprognose durchschnittliche Abweichung pro Modus ..... 101

# Tabellenverzeichnis

Tabelle 1: Vorstellung der Modi ..... 46

Tabelle 2: Datenform nach Sammlung von echten GPS-Koordinaten ..... 51

Tabelle 3: Datenform der Ausgangsdaten ..... 52

Tabelle 4: Datenform für die Trainingsdaten der Verkehrsteilnehmerklassifikation ..... 54

Tabelle 5: Datenform der wahren Klassen für die Verkehrsteilnehmerklassifikation ..... 54

Tabelle 6: Datenform der Trainingsdaten für die Klassifikation von Risikoklassen ..... 55

Tabelle 7: Datenform der wahren Cluster für die Klassifikation von Risikoklassen ..... 55

Tabelle 8: Datenform der Trainingsdaten für die Geschwindigkeitsprognose ..... 56

Tabelle 9: Datenform der wahren Geschwindigkeiten für die Geschwindigkeitsprognose ..... 56

Tabelle 10: Parameterübergabe Verkehrsteilnehmerklassifikation des Agenten mit Reinforcement Learning ..... 76

Tabelle 11: Parameterübergabe Verkehrsteilnehmerklassifikation der Umgebung mit Reinforcement Learning ..... 76

Tabelle 12: Parameterübergabe der Verkehrsteilnehmerklassifikation für gewöhnliche neuronale Netze ..... 78

Tabelle 13: Parameterübergabe des Agenten der Risikoklassen-Klassifikation mit Reinforcement Learning ..... 83

Tabelle 14: Parameterübergabe Umgebung der Risikoklassen-Klassifikation mit Reinforcement Learning ..... 84

Tabelle 15: Parameterübergabe der Risikoklassen-Klassifikation mit gewöhnlichen neuronalen Netzen ..... 85

Tabelle 16: Parameterübergabe der Geschwindigkeitsprognose ..... 88

# Codeverzeichnis

Code 1: init()-Funktion der Agentenklasse ..... 66

Code 2: act()-Funktion der Agentenklasse ..... 67

Code 3: learn()-Funktion der Agentenklasse ..... 67

Code 4: init()-Funktion der Umgebungsklasse ..... 69

Code 5: step()-Funktion der Umgebungsklasse ..... 70

Code 6: reset()-Funktion der Umgebungsklasse ..... 70

Code 7: Implementierung des Codes für gewöhnliche neuronale Netze ..... 72

Code 8: Implementierung der Modi ..... 86

# 1 Einleitung

## 1.1 Einführung und wissenschaftliche Fragestellung

In der heutigen Zeit sind Mobilität und Verkehrssicherheit entscheidende Faktoren, die die Lebensqualität und das Wohlbefinden der Menschen beeinflussen. Die Einführung von autonomen Fahrzeugen verspricht eine Verbesserung der Verkehrseffizienz, die Reduzierung von Unfällen und die Entlastung von Fahrerinnen und Fahrern. In diesem Kontext gewinnt das autonome Fahren zunehmend an Bedeutung und entwickelt sich zu einem rasant wachsenden Forschungsbereich, der das Potenzial hat, die Art und Weise, wie wir uns fortbewegen, grundlegend zu verändern.

Um ein sicheres und effizientes autonomes Fahren zu gewährleisten, ist die präzise Erkennung der Umgebung und der Verkehrsteilnehmer von entscheidender Bedeutung. Hierzu wurden bereits diverse Methoden wie kamerabasierte Systeme oder verschiedene Sensoren entwickelt, die jeweils ihre eigenen Stärken und Schwächen aufweisen. Im Zentrum dieser Masterarbeit steht die Entwicklung eines Systems, dass in der Lage ist, Verkehrsteilnehmer und deren Relevanz für das autonome Fahrzeug zu identifizieren und zu bewerten, basierend auf Positionsdaten, die aus GPS-Koordinaten gewonnen wurden.

Durch den Einsatz von Positionsdaten bietet diese Arbeit einen alternativen Ansatz zur Umgebungserkennung, der die bestehenden Technologien ergänzen könnte. Dieses System soll nicht nur in der Lage sein, Verkehrsteilnehmer präzise zu klassifizieren, sondern auch deren Relevanz im Verkehrsgeschehen einzuschätzen, um fundierte Entscheidungen für das autonome Fahrzeug zu ermöglichen. Eine solche Herangehensweise könnte zur Verbesserung der Sicherheit und Effizienz von autonomen Fahrzeugen beitragen und einen Schritt in Richtung einer nachhaltigeren und verantwortungsvolleren Mobilität darstellen.

Die vorliegende Masterarbeit verfolgt das Ziel, ein solches System systematisch zu entwickeln, das System auf gewonnene Datensätze anzuwenden und die daraus resultierenden Ergebnisse zu validieren und zu interpretieren.

## 1.2 Projekthintergrund

In Kooperation mit dem Fraunhofer-Institut für integrierte Schaltungen, dem Innovations-Zentrum Region Kronach e.V., dem Landkreis Kronach, der Valeo Schalter und Sensoren GmbH und der Hochschule für angewandte Wissenschaften Coburg wurde das Projekt 5G Modelregion Kronach ins Leben gerufen. Das Projekt verfolgt dabei das Ziel bis 2035 autonome Fahrzeuge für den öffentlichen Nahverkehr in Kronach einzuführen. Ein erfolgreicher Abschluss dieses Projektes kann dabei nicht nur helfen Personalkosten zu sparen, sondern kann vor allem auch den ländlicheren Regionen Kronachs zugutekommen. Somit hätte man die Möglichkeit für ländlichere Regionen eine zuverlässigere und öfter Anbindung anzubieten. Eine bessere Nutzung der öffentlichen Verkehrsmittel könnte dadurch auch der Umwelt zugutekommen, was zur heutigen Zeit ebenfalls ein relevantes Thema darstellt. [Kronach 2016]

Der Hochschule für angewandte Wissenschaften Coburg wurden im Rahmen dieses Projekts zwei Campus zur Verfügung gestellt. Der Lucas-Cranach-Campus soll dabei für die Forschung genutzt werden und der Campus Hohe Weide für Validierungen. Die Hauptaufgabe, die die Hochschule Coburg dabei verfolgt ist, es das Projekt mit zentralem Schwerpunkt auf der Car-2-X-Kommunikation und Fernsteuerung zu unterstützen. Dabei wird die Vision verfolgt, dass Studenten hieran mitarbeiten sollen, um sich in diesem Bereich ein großes Fachwissen aneignen zu können und somit auch potenzielle neue Fachkräfte der Zukunft auszubilden. [Kronach 2016]

## 1.3 Forschungsfragen

Im Rahmen dieser Masterarbeit wurden zur konkreten Zielsetzungs- und Bearbeitungszwecken Forschungsfragen formuliert. Die Forschungsfragen setzen sich dabei aus einer Hauptforschungsfrage und vier Teilforschungsfragen zusammen, welche im Laufe dieser Masterarbeit beantwortet werden sollen. Die Hauptforschungsfrage dieser Masterarbeit lautet:

**Wie lassen sich Verkehrsteilnehmer und deren Relevanz anhand Ihrer periodischen Positionsdaten mithilfe von Methoden des maschinellen Lernens klassifizieren?**

Die Hauptforschungsfrage lässt sich einzeln betrachtet vermutlich nicht direkt beantworten, weswegen Teilforschungsfragen formuliert wurden, welche die Hauptforschungsfrage im Gesamten beantworten sollen. Die Teilforschungsfragen dieser Masterarbeit lauten:

1. Woher stammen die sequenziellen Positionsdaten, welche Bedeutung haben Sie und welche Vorverarbeitungsschritte sind für eine Klassifikation relevant?

2. Welche Methoden des maschinellen Lernens eignen sich in Bezug auf die vorliegenden Positionsdaten zur Klassifikation und kommen für eine potenzielle Umsetzung in Frage?

3. Wie sähe eine konkrete Entwicklung der ausgewählten Methoden aus und welche Anforderungen sollte ein fertiges Ergebnis erfüllen?

4. Welche Ergebnisse können die entwickelten Klassifikationsmodelle vorweisen und welches Verbesserungspotenzial ergibt sich hieraus?

# 2 Verwandte und vorherige Arbeiten

## 2.1 Verwandte Arbeiten

### 2.1.1 [Simoncini+2018]

Die Arbeit [Simoncini+2018] untersucht die Möglichkeit, durch den Einsatz von Recurrent Neural Networks (RNN) die Klassifizierung von Fahrzeugen auf Basis von GPS-Daten zu verbessern. Hierfür wurden insgesamt 1 Million GPS-Tracks mit mehr als 55 Millionen GPS-Positionen, 56 Millionen zurückgelegten Kilometern und einer Gesamtdauer von 1,3 Millionen Stunden verwendet. Es wurden zwei RNNs trainiert, um leichte und schwere Fahrzeuge sowie mittelschwere Fahrzeuge zu klassifizieren. Dabei wurden verschiedene Parameter, wie beispielsweise die Geschwindigkeit und der Straßentyp, berücksichtigt.

Die Auswertung der Ergebnisse zeigt dabei vielversprechende Ergebnisse. In der binären Klassifikation konnte eine Präzision von bis zu 86% erreicht werden, während die vorherige Arbeit lediglich eine Präzision von 60% mit SVM erreicht hatte. Bei der Klassifikation von drei Klassen sank die Präzision, insbesondere bei der Klasse "Mittelschwer", jedoch immer noch auf ein akzeptables Niveau von 46%. Das RNN wurde nach 50 Epochen trainiert und zeigte insbesondere bei längeren GPS-Sequenzen von mindestens 30 Minuten eine gute Performance.

Das Paper zeigt, dass RNNs aufgrund ihrer Fähigkeit, Sequenzdaten zu verarbeiten, eine vielversprechende Methode zur Verbesserung der Fahrzeugklassifikation auf Basis von GPS-Daten darstellen. Es wurden verschiedene Faktoren berücksichtigt, um das Modell zu trainieren, darunter die Krähenflugdistanz und die Zeitverzögerung. Die Ergebnisse zeigen, dass RNNs eine vielversprechende Alternative zu SVMs darstellen, um die Klassifikation von Fahrzeugen auf Basis von GPS-Daten zu verbessern.

### 2.1.2 [Althoff+2009]

Das Paper [Althoff+2009] beschäftigt sich mit der Verwendung von Markov-Ketten zur Berechnung der Wahrscheinlichkeit von Zusammenstößen im Straßenverkehr. Eine Markov-Kette ist ein mathematisches Modell, das eine Folge von Ereignissen beschreibt, bei dem der

Übergang zu einem zukünftigen Zustand nur von dem aktuellen Zustand abhängt und nicht von den Zuständen, die vorher stattgefunden haben.

Für die Anwendung von Markov-Ketten im Straßenverkehr wurden die Zustände "Links abbiegen", "Geradeaus fahren" und "Rechts abbiegen" als Status definiert. Wenn man die Wahrscheinlichkeitsmatrix mit einem Ausgangszustand multipliziert, kann man die Wahrscheinlichkeiten für den Eintritt jedes möglichen Zustands berechnen.

[Althoff+2009] nutzt diese Methode, um in Kombination mit dem eigenen Fahrverhalten die Wahrscheinlichkeit für einen Zusammenstoß mit anderen Autofahrern zu kalkulieren. Durch die Berechnung der Wahrscheinlichkeiten, die ein anderer Fahrer hat, um in einen bestimmten Zustand zu wechseln, kann man die Wahrscheinlichkeit eines Zusammenstoßes berechnen, wenn man das eigene Fahrverhalten entsprechend anpasst.

Somit bietet das Paper eine Methode zur Berechnung der Wahrscheinlichkeit von Zusammenstößen im Straßenverkehr, was auch als Relevanz interpretiert werden kann. Es zeigt, wie die Verwendung von mathematischen Modellen zur Verbesserung der Sicherheit im Straßenverkehr beitragen kann.

## 2.2 Vorherige Arbeiten

Im Rahmen des Projekts „5G-Kronach“, wurden vor Bearbeitung dieser Masterarbeit, einige Bachelorarbeiten im Voraus geschrieben. Einige dieser Arbeiten bieten die Basis für diese Masterarbeit. Deswegen werden in diesem Kapitel die bisherigen, relevanten Bachelorarbeiten vorgestellt.

### 2.2.1 [Torlak 2022]

Die Bachelorarbeit [Torlak 2022] hat sich mit der Gewinnung von Positionsdaten und deren Weiterverarbeitung beschäftigt. Die Arbeit hatte dabei mehrere Ziele verfolgt, welche im Zeitraum der Bearbeitung erreicht, werden konnten.

Das erste Ziel war es ein Tool zu entwickeln, welches sich mit einer CARLA Instanz verbinden kann und GPS-Koordinaten aus CARLA abfragen kann. Alle Details zu CARLA werden in Kapitel X.X behandelt.

Die GPS-Koordinaten sollen im zweiten Ziel daraufhin weiterarbeitet werden zu Positionsdaten. Um welche Positionsdaten es sich hierbei genau handelt, wird in Kapitel X.X die Grundlage für den ersten der Entwicklungen dieser Masterarbeit sein, weswegen zur Einsicht auf dieses Kapitel verwiesen wird.

Die entwickelte Lösung hat sich hierbei als beständig erwiesen. Das Endergebnis waren mehrere funktionierende Skripte, deren Nutzung sich als intuitiv und nutzerfreundlich gestalten. Um die entwickelte Lösung verwenden zu können, wurde in der Abgabe eine Datei namens „example.py“ hinterlegt, in der man die notwendigen Hyperparameter einstellen konnte und dadurch dann auch die restlichen Skripte aufgerufen wurden.

Die Bachelorarbeit von Torlak hat dabei nicht nur zur Datengewinnung dieser Arbeit beigetragen, sondern auch zu einer weiteren Bachelorarbeit von Sohl, welche im nachfolgenden Kapitel beschrieben wird.

### 2.2.2 [Sohl 2022]

Die Bachelorarbeit von Sohl knüpft an der Arbeit von Torlak an und versucht ähnlich wie diese Masterarbeit mithilfe der Datenerhebungsmaßnamen von Torlak ein Klassifikationsverfahren für Verkehrsteilnehmer zu entdecken. Die Arbeit beschränkt sich dabei allerdings nur auf die Klassen der Verkehrsteilnehmer. Relevanz wird somit nicht abgedeckt.

Im ersten Schritt der Arbeit wurden die Verfahren vorgestellt die später in der Arbeit zur Klassifikation verwendet werden. Dabei wurden die Verfahren Support Vector Machine, Decision Tee und K-Nearest-Neighbours vorgestellt. Parallel dazu fand auch eine Erklärung aller anderen Faktoren statt.

Dannach werden alle Verfahren auf die gewonnenen Daten angewandt und es wurde für jedes Verfahren mehrere Abbildungen über die Präzision der Verfahren erstellt, welche daraufhin betrachtet und validiert werden konnten.

Die Validierung hat dabei ergeben, dass Support Vector Machine die besten Ergebnisse erzielen konnte. Platz Nummer zwei konnte Decision Tree belegen und den dritten Platz konnte k-means für sich beanspruchen. Vor allem bei sehr genauen Datensätzen, also dort wo der Ungenaugikeitsparameter auf 0 gesetzt wurde, konnten alle Verfahren mit einer hohen herausstechen. Support Vector Machine konnte beispielsweise eine Präzision von 93,90% bei drei Klassen und Null Meter Abweichung vorweisen.

Ein Nachteil, den allerdings alle genutzten Verfahren vorweisen mussten, waren, dass bei steigender Ungenauigkeit der Daten die Präzision rapide abnahm. Support Vector Machine konnte demnach bei einer Ungenauigkeit von bis zu drei Metern nur noch eine Präzision 66,56% vorweisen. Eine gewisse Toleranz gegenüber Verfälschungen ist an dieser Stelle allerdings wichtig, da in der Realität leichte Abweichungen von GPS-Signalen die Normalität sind.

Die Arbeit konnte somit einen ersten Meilenstein für die Klassifikation anhand von Positionsdaten setzen und die Fragestellung offenlassen ob andere Methoden des maschinellen Lernens sich als widerstandsfähiger gegenüber Verfälschungen erweisen.

### 2.2.3 [Opitz 2022]

Die Bachelorarbeit von Opitz ist die aktuellste Bachelorarbeit zu diesem Thema und knüpft an den bisherigen Bachelorarbeiten an. Ziel in dieser Bachelorarbeit war es mithilfe von Carla, einer Verkehrssimulation, ein System zu entwickeln, welches in der Lage ist die Relevanz Echtzeit anzuzeigen.

Der Ansatz, den diese Bachelorarbeit verfolgte, war es mithilfe eines Parameters TTC, feste Szenarien zu entwickeln. Diese Szenarien wurden daraufhin erstmals vorgestellt. Dabei wurden beispielsweise verschiedenste Szenarien vorgestellt in denen Autos miteinander kollidieren könnten. Einige Beispiele hierfür wäre eine seitliche Kollision, frontale Kollision oder auch eine Kollision von zwei Ecken.

Die vorgestellten Szenarien wurden daraufhin fest im Code implementiert. Während der Laufzeit wurde daraufhin ein Fahrzeug als Protagonist ausgewählt und in der Ego Perspektive wurden die TTC-Parameter zu anderen Verkehrsteilnehmern aufgezeigt. Dies konnte realisiert werden und indem während der Laufzeit durch eine wiederholte Abfrage zwischen zwei Verkehrsteilnehmern der TTC-Parameter für verschiedene Stellen in Autos berechnet wurde.

# 3 Grundlagen

## 3.1 CARLA

Das Grundlagenkapitel dieser Arbeit beginnt mit der Vorstellung eines Verkehrssimulators. Dieser Verkehrssimulator nennt sich CARLA und wurde speziell dafür entwickelt um die Entwicklung, das Training und die Validierung von autonomen Fahrsystemen zu unterstützen. [CARLA 2023]

Parallel dazu ist CARLA Open-Source, das bedeutet der Quellcode ist öffentlich einsehbar und frei zugänglich. Die Simulationsumgebung bringt in ihrer Standardversion einige Features mit sich, wie zum Beispiel Aktoren, Sensoren, Simulationsumgebungen oder auch einen Verkehrsmanager, der das Verhalten von virtuellem Verkehrsteilnehmer steuern kann. Die eben genannten Features sind allerdings nur einige Beispiele die CARLA zu bieten hat, um eine ausführliche Liste an Features und Möglichkeiten einsehen zu können wird auf Website von CARLA verwiesen. [CARLA 2023]

Um in CARLA eigene Projekte umsetzen zu können, gibt es verschiedene Möglichkeiten dies zu realisieren. Diese Arbeit beschränkt sich allerdings auf die reine Entwicklung über die API, die CARLA einem Entwickler zur Verfügung stellt. Über die API ist es möglich mit einer aktuell laufenden Simulation interagieren zu können. Dies umfasst beispielsweise die Abfrage von aktuellen Sensordaten oder auch die Manipulation von Akteuren. Die Möglichkeiten zur Manipulation von Akteuren sind auf den jeweiligen Akteur beschränkt, eine Ampel beispielsweise bietet im Vergleich zu einem Fußgänger unterschiedliche Möglichkeiten für eine Steuerung an. Für eine ausführliche Dokumentation der Möglichkeiten wird hierfür an die offizielle CARLA Dokumentation verwiesen. [CARLA 2023]

Im Rahmen dieser Arbeit wird CARLA und dessen API für zwei Hauptzwecke verwendet. Der erste Zweck ist die Datengewinnung von Positionsdaten verschiedenen Verkehrsteilnehmern, diese wird in Kapitel X.X vorgestellt. Der zweite Zweck ist die Anbindung eines entwickelten Klassifikationsmodells, um Verkehrsteilnehmer während einer laufenden Simulation klassifizieren zu können. Die Entwicklung und letztendliche Anbindung dieses Klassifikationsmodells werden in Kapitel X.X thematisiert. [CARLA 2023]

## 3.2 Neuronale Netze

### 3.2.1 Einführung

Ein neuronales Netz ist ein Modell, dass sich an der Funktionsweise des menschlichen Gehirns orientiert, um dessen Informationsverarbeitung nachzuahmen. Zur heutigen Zeit sind neuronale Netze im Bereich der künstlichen Intelligenz zu einem fundamentalen Werkzeug geworden, dessen Anwendungsbereich stetig weiterwächst. [Nielsen 2015]

Heute werden neuronale Netze vor allem in der Mustererkennung und Klassifikation eingesetzt, aber auch in der Kommunikation finden neuronale Netze immer mehr Gebrauch.


Abbildung 3.1: Interesse im zeitlichen Verlauf ChatGPT [Google-Trends 2023]

Ein Beispiel hierfür wäre der kürzlich veröffentlichte Chatbot namens ChatGPT von OpenAI, welcher binnen kürzester Zeit das Internet erobert hat. Abbildung X.X zeigt, das Interesse an Chat GPT im zeitlichen Verlauf, wobei ein Score von 100 als das Maximum interpretiert werden kann. [Google-Trends 2023]

### 3.2.2 Aufbau eines neuronalen Netzes

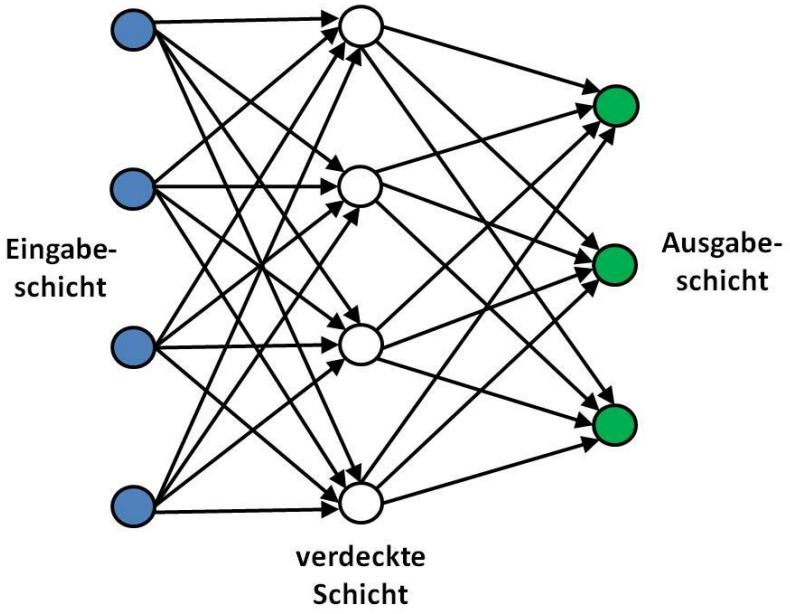


Abbildung 3.2: Aufbau eines neuronalen Netzes [Dörn 2023]

Um sich ein neuronales Netz besser vorstellen zu können, wird Abbildung X.X als Referenz verwendet. Ein neuronales Netz besteht auf den ersten Blick erstmals aus Schichten und Neuronen. Die erste Schicht eines neuronalen Netzes ist für gewöhnlich die Eingabeschicht. In der Eingabeschicht erfolgt, wie es schon im Namen enthalten ist, die Eingabe, die man dem neuronalen Netz mitgeben möchte. [Nielsen 2015]

Alle weiteren Schichten, die bis zur Ausgabeschicht folgen nennt man verborgene Schichten. In den verborgenen Schichten befinden sich die Neuronen die letztlich für die Weiterverarbeitung der Eingaben zuständig sind. [Nielsen 2015]

Die letzte Schicht eines neuronalen Netzes ist die Ausgabeschicht. In dieser Schicht erfolgt die Ausgabe, die ein neuronales Netz tätigen soll. Bei einer Klassifikation von beispielsweise drei Klassen, enthält die Ausgabeschicht drei Neuronen als Repräsentant jeder Klasse. Eine Ausgabe könnte dann beispielsweise die Wahrscheinlichkeit für die Aktivierung jedes Ausgabeneurons darstellen. [Nielsen 2015]

Betrachtet man ein Neuron im Einzelnen, ohne jeglichen Einflüssen, so ist ein Neuron lediglich eine Zahl. In vielen Fällen wird diese Zahl normalisiert und bewegt sich in einem Wertebereich zwischen 0 und 1. Eine Tendenz näher zu 0 würde dabei für eine geringe Aktivie-

rung sprechen und eine von 1 dementsprechend für eine hohe Aktivierung. Wie sich diese Zahl allerdings konkret zusammensetzt und verändert, ist das, was neuronale Netze ausmachen und wird nun in den nachfolgenden Unterkapiteln behandelt. [Nielsen 2015]

Neuronen haben die Eigenschaft, dass sie mit jedem Neuron in der nächsten Schicht verbunden sind. Damit hat jedes Neuron in der vorherigen Schicht einen Einfluss auf die Aktivierung der nachfolgenden Neuronen in der nächsten Schicht. Wie groß dieser Einfluss letztendlich ist, spiegelt sich in Form von Gewichten wider. [Nielsen 2015]

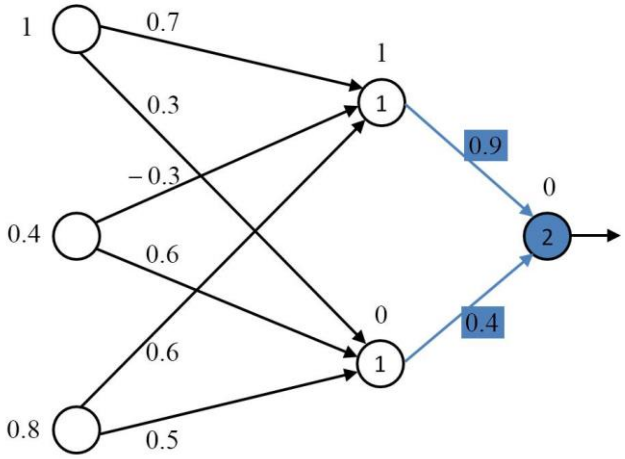


Abbildung 3.3: Neuronales Netz mit Werten und Gewichten [Dörn 2023]

Betrachtet man die Zusammensetzung eines einzelnen Neurons, so zeigt sich, dass sich der Wert dieses Neurons über die Summe aller vorigen Neuronen multipliziert mit den jeweiligen Gewichten, die zum Neuron führen, ist. Gewichte werden am Anfang eines Netzes zufällig initialisiert und im Laufe des Trainings angepasst, um optimale Ergebnisse zu erzielen. [Nielsen 2015]

### 3.2.3 Aktivierungsfunktionen

Neuronale Netze verwenden Aktivierungsfunktionen, um die Ausgabe eines Neurons zu beeinflussen. Ohne Aktivierungsfunktionen wären neuronale Netze lediglich lineare Modelle, die für Eingabe x eine lineare Ausgabe y erzeugen würden. Somit wären diese nicht in der Lage, komplexe Probleme zu lösen. [Nielsen 2015]

Ein weiterer wichtiger Aspekt in Aktivierungsfunktionen besteht darin, dass man mit diesen festlegen kann, ob ein Neuron aktiviert oder nicht aktiviert wird. Das bedeutet man kann mit

Aktivierungsfunktionen regulieren, ob ein Neuron eine Ausgabe erzeugen soll oder nicht. [Nielsen 2015]

Es gibt viele verschiedene Möglichkeiten für Aktivierungsfunktionen. Oft verwendete Funktionen sind beispielsweise die Sigmoid-Funktion, die ReLu Funktion oder auch die Hyperbolic-Tangent-Funktion. Im Rahmen dieser Masterarbeit wurde zur Entwicklung die ReLu Funktion verwendet, diese wird nun im nachfolgenden vorgestellt. [He+2015]

Die ReLu (Rectified Linear Unit) Funktion ist eine derzeit populäre Aktivierungsfunktion und wird häufig im Bereich der neuronalen Netze verwendet. Die ReLu Funktion hat folgende Form.

$$R(x) = \max(0, x)$$

Wenn x, also der aktuelle Wert des Neurons, größer als 0 ist, dann ist das Neuron aktiviert und der Ausgang x. Dies ermöglicht es dem Modell, schneller zu lernen und die Genauigkeit zu verbessern. Wenn x kleiner als 0 ist, dann ist der Ausgang 0 und es findet demnach keine Aktivierung statt. Das bedeutet, dass die Gewichte, die mit x assoziiert sind, ebenfalls nicht aktualisiert werden. [He+2015]

Das macht die ReLu Funktion im Vergleich zu anderen Aktivierungsfunktionen performanter, da keine sonderlich rechenintensiven Operationen stattfinden. [He+2015]

Einen Nachteil bringt ReLu Funktion jedoch mit sich, nämlich das Neuronen unter der ReLu Funktion „sterben“ könnten. Das bedeutet, dass ein Neuron und dessen Gewichtung niemals den Schwellenwert 0 erreichen und somit niemals deren Gewichte angepasst werden. Diese Neuronen werden auch „tote“ Neuronen genannt, da diese niemals aktiviert werden und somit niemals einen Beitrag zur Gesamtheit des Neuronalen Netzes beitragen. [He+2015]

### 3.2.4 Backpropagation

Nachdem im Detail betrachtet wurde, wie sich neuronale Netze zusammensetzen, wird in diesem Kapitel ein tieferer Blick in das Lernen von neuronalen Netzen gezogen. Neuronale Netze lernen durch ein Verfahren, welches sich „Backpropagation“ nennt. [Nielsen 2015]

Backpropagation ist ein Algorithmus, der verwendet wird, um die Gewichte in einem neuronalen Netz anpassen zu können. Der Algorithmus basiert dabei auf dem Prinzip des Gradien-

tenabstiegs, bei dem die Gewichte des Modells so angepasst werden, dass die Fehlerrate in Richtung eines Minimums verringert wird. [Nielsen 2015]

Der Algorithmus setzt sich aus zwei Schritten zusammen. Der erste Schritt ist die Vorwärtspropagation, bei der Eingabedaten durch das neuronale Netz geschickt werden und daraufhin in der Ausgabeschicht eine Ausgabe erzeugt wird. Im zweiten Schritt findet dann die eigentliche Backpropagation statt. Hier wird der Fehler zwischen Ausgabe des Netzes und gewünschter Ausgabe berechnet, um daraufhin durch das Netzwerk zurückzupropagieren und die Gewichte anzupassen. [Nielsen 2015]

### 3.2.5 Mathematische Betrachtung Backpropagation

Mathematisch betrachtet lässt sich der erste Schritt, also die Vorwärtspropagation folgendermaßen darstellen.

$$y_{pred} = f(h) = f(w_{1_{x_1}} + w_{2x_2} + \ldots + w_{n_{x_n}})$$

$y\_pred$ stellt die vorhergesagte Ausgabe eines Neurons dar, welche durch die Aktivierungsfunktion $f$ und der gewichteten Summe der Eingänge berechnet wird.

Aus der Ausgabe kann man daraufhin einen mithilfe einer Fehlerfunktion E(w), den Fehler zwischen vorhersage und wirklicher Ausgabe berechnen. Eine häufig verwendete Fehlerfunktion ist die Summe des Quadrats der Fehler. [Nielsen 2015]

$$E(w) = \frac{1}{2} * (\sum \left(y_{pred} - y\right)^2$$

Ein neuronales Netz verfolgt das Ziel diese Fehlerfunktion auf ein Minimum zu reduzieren, dies wird im zweiten Schritt der Backpropagation versucht zu realisieren. [Nielsen 2015]

In der Backpropagation wird für jedes Gewicht der Gradient des Fehlers $\frac{\partial \mathrm{E}}{\partial \mathrm{w_i}}$ mithilfe der Ausgabe $y_{pred}$, der tatsächlichen Ausgabe $y$, der Ableitung der Aktivierungsfunktion $f(h)$ und des Eingangswerts $x_i$ berechnet. Die Formel hierfür sieht folgendermaßen aus. [Nielsen 2015]

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w_i}} = \left(y_{pred} - y\right) * f(h) * x_i$$

Ein Gradient des Fehlers (oder auch Fehlergradient genannt) ist ein Vektor, der die Änderung des Fehlers in Bezug auf jedes Gewicht im Neuronalen Netz beschreibt. Es gibt einen Gradienten des Fehlers für jedes Gewicht im Netzwerk. [Nielsen 2015]

Nachdem die Gradienten berechnet wurden, können daraufhin die Gewichte angepasst werden. [Nielsen 2015]

$$w_i = w_i - \eta * \frac{\partial \mathrm{E}}{\partial \mathrm{w_i}}$$

η stellt dabei eine Lernrate dar, mit der man bestimmen kann, wie stark die Gewichte verändert werden sollen. Wählt man beispielsweise einen hohen Wert für η, dann führt dies zu einer hohen Änderung der Gewichte, wodurch das Minimum der Fehlerfunktion leicht verfehlt werden kann. Wird η niedrig gewählt, dann muss das Netz dementsprechend sehr lange trainieren, bis ein Minimum der Fehlerfunktion erreicht wird. Grundsätzlich orientiert man sich bei der Lernrate in der Regel eher an kleineren Werten, welche sich meist im Rahmen von 0.001 bis 0.01 bewegen. [Nielsen 2015]

### 3.2.6 Findung von optimalen Hyperparametern

Zur Beantwortung der Frage nach der optimalen Anzahl von verstecken Schichten und der Anzahl von Neuronen in diesen Schichten, gibt es keine allgemeingültige Methode, die diese Frage beantworten kann. [Bergstra+2012]

Die Wahl der versteckten Schichten und Neuronen ist sehr vom Anwendungsfall abhängig und unterscheidet sich dadurch stetig. Neben normalen Try and Error gibt es ein paar Methoden, die sich verwenden lassen, um die optimale Anzahl an versteckten Schichten und Neuronen zu finden. Gängige Methoden, die sich verwenden lassen, sind beispielsweise K-faltige Kreuzvalidierung, Grid-Search, Randomized-Search oder auch Early-Stopping. [Bergstra+2012]

In dieser Masterarbeit wurde zur Findung der optimalen Hyperparameter die Grid-Search Methode verwendet. Die Idee hinter Grid-Search ist, dass man einen Bereich von möglichen Werten für die Hyperparameter festlegt und dann das Modell für jede Kombination von Werten aus diesem Bereich trainiert und die Leistung des Modells auf einem Validierungsdatensatz beurteilt. Die Kombination von Werten, die die beste Leistung auf dem Validierungsdatensatz liefert, wird als die besten Hyperparameter des Modells angesehen. [Bergstra+2012]

Grid Search kann dabei automatisch oder manuell durchgeführt werden und bringt den großen Vorteil mit sich, eine Beurteilung für möglichst viele Hyperparameter-Kombinationen vorlegen zu können und somit eine optimalere Entscheidung treffen zu können. [Bergstra+2012]

### 3.2.7 Weitere Arten von neuronalen Netzen

Das Kapitel Neuronale Netze dieser Masterarbeit hat die Grundsteine von Neuronalen Netzen vorgestellt. Die vorgestellten Grundsteine lassen sich auf ein normales Feed Forward Netz, wie es in dieser Masterarbeit implementiert wird anwenden. Im Laufe der Zeit sind die Aufgabenbereiche für neuronale Netze allerdings stetig erweitert worden, dementsprechend gibt es mittlerweile verschiedene Arten von neuronalen Netzen, welche für sich für spezialisiertere Aufgaben eignen. Jedes dieser Netze im Detail vorzustellen, würde den Rahmen dieser Masterarbeit sprengen, dennoch erhalten einige häufig verwendete Netze in diesem Kapitel der Vollständigkeit wegen einer kurzen Erwähnung.

**Convolutional Neural Networks (CNN)**

Convolutional Neural Networks sind Neuronale Netze die speziell für die Verarbeitung von Bilddaten konizipiert worden sind. Sie verwenden dabei sog. Convolutional Layers, um relevante Merkmale in Bilddaten zu extrahieren und diese Merkmale dann durch mehrere Fully-Connected Layers zu klassifizieren. [Nielsen 2015]

**Recurrent Neural Networks (RNN)**

Recurrent Neural Networks sind Neuronale Netze, die sich speziell für die Verarbeitung von Sequenzen in Daten eignen. Sie verwenden dabei Rückkopplungsschleifen, um Informationen über die vorherigen Schritte der Sequenz zu speichern und zu verwenden, um die aktuellen Schritte vorherzusagen. [Sherstinsky 2020]

**Long Short-Term Memory (LSTM)**

LSTMs sind eine Erweiterung von RNNs, die besonders nützlich für die Verarbeitung von langen Sequenzen sind. Sie verwenden dabei spezielle Neuronen namens "Memory Cells", um Informationen über längere Zeitperioden zu speichern und zu verwenden. [Sherstinsky 2020]

**Generative Adversarial Networks (GAN)**

Generative Adversarial Networks bestehen aus zwei Netzen, einem Generator und einem Discriminator. Der Generator versucht, Daten zu generieren, die so aussehen, als ob sie aus dem Ausbildungsdatensatz stammen, während der Discriminator versucht, die generierten Daten von echten Daten zu unterscheiden. Das Ziel von GANs ist es, den Generator dazu zu bringen, Daten zu generieren, die so realistisch wie möglich sind. GANs werden in vielen Anwendungen wie zum Beispiel Bildgenerierung, Textgenerierung und Musikgenerierung eingesetzt. [Goodfellow+2020]

## 3.3 Reinforcement Learning

### 3.3.1 Einführung

Reinforcement Learning ist ein Teilgebiet des maschinellen Lernens, bei dem ein sog. „Agent“ mit einer Umgebung durch Aktionen interagiert und versucht aus diesen Interaktionen zu lernen. Dies geschieht, indem der Agent Belohnungen erhält, wenn er richtige Aktionen ausführt und Strafen erhält, wenn er falsche Aktionen ausführt. Ein Beispiel für den Einsatz von Reinforcement Learning findet sich zur heutigen Zeit in Computerspielen wieder. Dort erzielen Reinforcement Learning Agents in manchen Spielen teilweise bessere Ergebnisse als heutige Profispieler. Ein anderer Bereich wäre beispielsweise die Steuerung von autonomen Fahrzeugen. [Sutton+2018]

### 3.3.2 Markov Decision Process

Ein Markov-Entscheidungsprozess (MDP) ist ein mathematisches Modell, das verwendet wird, um optimale Entscheidungen in Problemen mit mehreren Schritten zu treffen. Es wird häufig im Bereich der künstlichen Intelligenz und des maschinellen Lernens verwendet, um Probleme zu lösen, bei denen der optimale Pfad durch eine Reihe von Entscheidungen und deren Auswirkungen auf den zukünftigen Zustand bestimmt wird. [Sutton+2018]

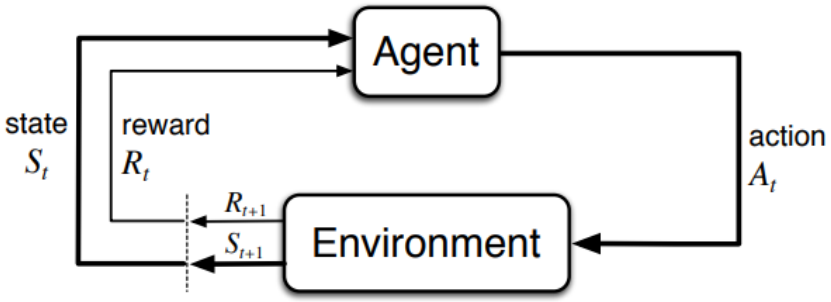


Abbildung 3.4: Agent Umgebung Interaktion in einem MDP [Sutton+2018]

Ein MDP besteht aus einer Reihe von Zuständen, Aktionen und Belohnungen zwischen einem Agenten und einer Umgebung. Der Prozess beginnt in einem Startzustand und ein Agent wählt eine Aktion, die die Umgebung in einen neuen Zustand versetzt. Der Agent erhält dann eine Belohnung, die von dem neuen Zustand und der Aktion abhängt. Dieser Prozess wiederholt sich, bis der Agent einen Endzustand erreicht hat. [Sutton+2018]

Der Zweck eines MDP ist es, den Agenten dabei zu helfen, optimale Entscheidungen zu treffen, indem er den Wert jedes möglichen Zustands berechnet und die maximalen Belohnungen für jede mögliche Aktion in diesem Zustand berücksichtigt. Die Berechnungen werden durch die Verwendung von Algorithmen wie der Bellman-Funktion in Kapitel X.X durchgeführt. [Sutton+2018]

Hierbei ist zu beachten, dass Anwendungsfälle existieren, die keinen Endzustand besitzen, sondern stetig weiterlaufen. Man unterscheidet hierbei zwischen fortlaufenden und episodischen Anwendungsfällen. Aus diesem Grund werden in den nachfolgenden Kapiteln angepasste Formeln verwendet, welche eine Notation für beide Anwendungsfälle darstellen können. [Sutton+2018]

### 3.3.3 Mathematische Betrachtung

Wie im vorherigen Kapitel festgelegt, sind im MDP fünf Komponenten festgelegt, in diesem Kapitel werden diese Komponenten Mathematisch betrachtet.

Mathematisch betrachtet ergibt sich daraus eine Menge an Status **S**, eine Menge an Aktionen **A** und eine Menge an Belohnungen **R**. Im gewöhnlichen MDP sind diese Mengen endlich, die nachfolgenden Formeln eignen sich allerding auch für unendliche Mengen. [Sutton+2018]

Für jeden zeitlichen Schritt $t = 0,1,2 \ldots$ erhält der Agent einen Status von der Umgebung $S_t \in \boldsymbol{S}$. Abhängig von diesem Status reagiert der Agent mit einer Aktion A auf diesen Status $A_t \in \boldsymbol{A}$. Dadurch entsteht ein sog. „State Action Pair“ $(S_t, A_t)$. [Sutton+2018]

Der zeitliche Schritt wird danach inkrementiert $t + 1$ und die Umgebung in einen neuen Zustand $S_{t+1} \in \boldsymbol{S}$ überführt. Der Agent erhält zu diesem Zeitpunkt eine Belohnung $R_{t+1} \in \boldsymbol{R}$ für dessen durchgeführte Aktion $A_t$ beim Zustand $S_t$. [Sutton+2018]

Eine Funktion zum Erhalt von Belohnungen $f(S_t, A_t)$, könnte dadurch folgendermaßen dargestellt werden. [Sutton+2018]

$$f(S_t, A_t) = R_{t+1}$$

Der Verlauf, der den sequenziellen Prozess der Auswahl einer Aktion aus einem Zustand, des Übergangs zu einem neuen Zustand und des Erhalts einer Belohnung darstellt, kann wie folgt dargestellt werden. [Sutton+2018]

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

### 3.3.4 Belohnungen

Ein großer Punkt in Reinforcement Learning ist die Vergabe von Belohnungen. In Kapitel X.X wurde festgelegt, dass ein Agent versucht sein Verhalten mit dem stetigen Ziel der Belohnungsmaximierung anzupassen. In diesem Kapitel wird eine Möglichkeit vorgestellt kumulative Belohnungen zusammenzufassen und zu formalisieren. Aus diesem Zweck wird in diesem Kapitel das Konzept „expected Return“ (zu Deutsch.: erwartete Belohnung) der Belohnungen zu einem bestimmten zeitlichen Schritt vorgestellt. [Sutton+2018]

Wenn die Folge der nach dem Zeitschritt $t$ erhaltenen Belohnungen laut Kapitel X.X wie folgt bezeichnet wird $R_{t+1}, R_{t+2}, R_{t+3} \ldots R_{t+n}$ stellt sich die Frage welcher Aspekt dieser Folge denn nun genau maximiert werden soll. Vielleicht lohnt es sich beispielsweise in $R_{t+2}$ eine negative Belohnung entgegenzunehmen, da sich durch diesen Schritt die Gesamte Belohnung in $R_{t+3}$ deutlich erhöhen kann. [Sutton+2018]

Um diese Frage zu beantworten, wird hierfür der bereits erwähnte „expected return“ berechnet. [Sutton+2018]

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

$G_t$ stellt in dieser Formel den expected return dar und $R_T$ den finalen zeitlichen Schritt. [Sutton+2018]

Durch die Berechnung des expected return ist es dem Agenten nun möglich eine möglichst hohe Gesamtbelohnung zu erzielen, auch wenn das bedeutet in manchmal eine negative Belohnung zu erhalten. [Sutton+2018]

Die obige Formel wäre die Formel für den einfachsten Fall. Denn die oben genannte Formel ist für Anwendungen sinnvoll in denen ein finaler zeitlicher Schritt existiert, das bedeutet, wenn die Interaktion zwischen Agenten und Environment auf natürliche Weise in Teilsequenzen zerfällt. Diese Teilsequenzen werden auch Episoden genannt. Ein Beispiel für diesen natürlichen Zerfall wäre ein Computerspiel, bei dem der Agent einem Labyrinth entkommen soll und sich dabei nur maximal X Schritte bewegen darf. Unabhängig davon, ob der Agent es geschafft hat dem Labyrinth zu entkommen, beginnt eine neue Episode, die nicht vom vorherigen Ergebnis abhängig ist. [Sutton+2018]

Nun existieren allerdings auch Szenarien, für die kein finaler zeitlicher Schritt existiert. Das bedeutet Szenarien, die theoretisch unendlich lang anhalten können. Ein Beispiel hierfür wäre eine industrielle Maschine, welche niemals stoppen soll. Für solche Szenarien lässt sich die obige nicht anwenden, weswegen hierfür eine Anpassung notwendig ist. Um diesem Problem entgegenzuwirken wird sog. „Discounting“ verwendet und das Ziel des Agenten ist es nun den sog. „Discounted expected return“ zu maximieren. [Sutton+2018]

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$\gamma$ hat dabei den Wertebereich $0 \leq \gamma \leq 1$ und wird als „discount rate“ bezeichnet. [Sutton+2018]

Die discount rate bestimmt den Gegenwartswert zukünftiger Belohnungen. Eine Belohnung die k Zeitschritte entfernt ist, ist zum aktuellen Zeitschritt nur das k1-fache von dessen wert, was sie wert wäre, wenn sie aktuell ist. Das bedeutet zukünftige Belohnungen die näher am aktuellen zeitlichen Schritt sind werden höher gewichtet, während Belohnungen die zeitlich weiter entfernt sind an Gewichtung und somit an Relevanz verlieren. [Sutton+2018]

Bei der Wahl des Wertes für die discount rate ist somit zu beachten, dass die discounted expected Return bei Annäherung an die 1 weitsichtiger wird, da mehr potenzielle Belohnungen

mit einbezogen werden können. Dementsprechend gilt für eine Annäherung an die Zahl 0, das Gegenteil, da weniger potenzielle Belohnungen mit einbezogen werden können. [Sutton+2018]

### 3.3.5 Policies und Value Functions

Nachdem im letzten Kapitel festgelegt wurde, dass das Ziel eines Reinforcement Learning Agenten die Maximierung eines expected Returns ist, beschäftigt sich dieses Kapitel mit Policies und Value Functions. [Sutton+2018]

Policies (zu Deutsch Regeln) stellen eine Funktion dar, die für einen gegeben Status eine Wahrscheinlichkeit für eine Aktion ermittelt. Wenn ein Agent zum Beispiel einer Policy $\pi$ zum Zeitpunkt $t$ folgt, dann ist

$$\pi(a|s)$$

die Wahrscheinlichkeit für die Aktion, $A_t$ wenn der Status $S_t$ eintritt. Das bedeutet, dass zum Zeitpunkt $t$ unter der Policy $\pi$ die Wahrscheinlichkeit Aktion $A$ bei Status $S$ zu nehmen ist. [Sutton+2018]

Value Functions sind Funktionen für Status oder State Actions Pairs, die zurückgeben, wie gut ein Status für einen Agenten funktioniert oder wie gut es für einen Agenten ist eine bestimmte Handlung in einem bestimmen Zustand auszuführen. [Sutton+2018]

Dabei unterscheidet man zwischen State-Value-Functions und Action-Value-Functions. [Sutton+2018]

Die State-Value-Function für Policy $\pi$ wird als $v_\pi$ zurückgegeben und beschreibt wie ein Status für einen Agenten und der Policy $\pi$ funktioniert. Formal betrachtet ist der Wert des Zustands $s$ unter der Politik $\pi$ die expected return, die sich ergibt, wenn man von dem Zustand zu einem bestimmten Zeitpunkt ausgeht und danach der Policy folgt. Mathematisch lässt sich die State-Value-Function folgendermaßen ausdrücken

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

Die Action-Value-Function für Policy $\pi$ wird als $q_\pi$ zurückgegeben und beschreibt, wie gut es für einen Agenten wäre eine Aktion unter dem Status $s$ zu wählen, während ein Agent der Policy $\pi$ folgt. Formal betrachtet ist der Wert einer Handlung $a$ im Zustand $s$ unter der Politik

π der expected return, der sich ergibt, wenn man zum Zeitpunkt $t$ vom Zustand $s$ ausgeht, die Handlung $a$ ausführt und dabei die Politik π verfolgt. [Sutton+2018]

Mathematisch wird die Action-Value_Function folgendermaßen ausgedrückt. [Sutton+2018]

$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

Üblicherweise wird die Action-Value-Function als Q-Funktion bezeichnet, und die Ausgabe der Funktion für ein beliebiges Zustands-Aktionspaar wird als Q-Value bezeichnet. [Sutton+2018]

### 3.3.6 Findung von optimalen Policies

Im vorherigen Kapitel wurde das Konzept von Policies und Value Functions vorgestellt. Das Ziel eines Agenten ist es eine optimale Policy zu finden die möglichst viele Belohnungen bringt, wenn der Agent dieser Policy folgt. Doch was genau optimal in diesem Zusammenhang bedeutet und wie ein Agent dieses Ziel erreicht, wird in diesem Kapitel vorgestellt. [Sutton+2018]

**Optimale Policy**

Wenn von einer optimalen Policy die Rede ist, dann ist damit eine Policy π gemeint bei der die größte Belohnung erzielt werden kann. [Sutton+2018]

$$\pi \geq \pi' \; if \; and \; only \; if \; v_\pi(s) \geq v_{\pi'}(s) for \; all \; s \in \boldsymbol{S}$$

Dementsprechend wird eine neue Policy nur übernommen, wenn unter dessen Befolgung ein größerer expected return erzielt werden kann. [Sutton+2018]

**Optimale State-Value Function**

Eine optimale State-Value Function lässt sich mathematisch folgendermaßen definieren. [Sutton+2018]

$$v_*(s) = \max v_\pi(s)$$

Das bedeutet je höher $v_*$, also je höher der exptected return, desto besser. [Sutton+2018]

**Optimale Action-Value Function**

Ähnlich wie bei der Optimalen State-Value Function ist eine optimale Action-Value Function erreicht, wenn die expected return ihr Maximum erreicht hat. [Sutton+2018]

$$q_*(s,a) = \max q_\pi(s,a)$$

## 3.4 Reinforcement Learning – Q Learning

Nachdem in den vorherigen Kapiteln optimale Policies und Value Functions vorgestellt wurden, stellt sich nun die konkrete Frage wie ein Agent sich weiterentwickeln kann, um eine optimale Policy finden zu können. Eine dieser Möglichkeiten nennt sich Q-Learning und diese Möglichkeit wird nun in diesem Kapitel vorgestellt.

Q-Learning verfolgt das Ziel mithilfe von sog. Q-Values eine optimale Policy zu finden. Hierfür wird im Q-Learning-Verfahren eine Tabelle mit allen möglichen Zuständen und Aktionen erstellt, die als Q-Tabelle bezeichnet wird. Der Agent füllt die Q-Tabelle aus, indem er mit der Umgebung interagiert und dabei Belohnungen und Strafen erhält. Die Q-Tabelle dient dann als Referenz für den Agenten, wenn er Entscheidungen trifft, indem diese ihm sagt, welche Aktion in einem bestimmten Zustand die höchste Belohnung verspricht. [Sutton+2018]

Damit diese Q-Values möglichst optimal werden, wird iterativ mit der Bellman Equation, weche in Kapitel X.X vorgestellt wird, versucht die Q-Values zu aktualisieren. Das Ganze wird dabei so lange versucht bis die Q-Funktion gegen die optimale Q-Funktion konvergiert. Dieses Verfahren wird auch “Value Iteration” genannt. [Sutton+2018]

### 3.4.1 Epsilon Greedy Strategy

Im vorigen Kapitel wurde das Prinzip von Q-Learning vorgestellt, wo versucht wird eine Tabelle aufzuspannen, welche für eine Aktion in einem bestimmten Zustand die höchste Belohnung verspricht. Doch wo genau soll der Agent anfangen die Tabelle zu aktualisieren? Gerade am Anfangszustand wo noch kein einziger Q-Value gesetzt wurde. Um diesen Problem entgegen zu kommen wird in diesem Kapitel die Epsilon Greedy Strategy vorgestellt.

Die Epsilon-Greedy-Strategie wird häufig in Q-Learning-Verfahren verwendet, um dem Agenten beizubringen, in einer Vielzahl von Statusen die besten Entscheidungen zu treffen. Sie hilft dem Agenten, eine gute Balance zwischen Exploitation und Exploration zu finden, indem sie ihm erlaubt, neue Aktionen auszuprobieren, während er gleichzeitig die bisherigen Aktionen nutzt, die er für die besten hält. [Sutton+2018]

Das funktioniert indem der Agent dazu zu gebracht wird, zwischen Explorations- und Exploitation-Aktionen zu wechseln. Die Idee hinter der Epsilon-Greedy-Strategie ist, dass der Agent in den meisten Fällen die Aktion wählt, die er für am besten hält, basierend auf seiner bisherigen Erfahrung. Dies wird als Exploitation bezeichnet. Allerdings gibt es auch Fälle, in denen der Agent neue Aktionen ausprobieren sollte, um zu sehen, ob sie besser sind als die bisherigen. Dies wird als Exploration bezeichnet. [Sutton+2018]

Die Epsilon-Greedy-Strategie verwendet einen Parameter namens Epsilon, welcher eine Repräsentation für die Explorationsrate ist. Wenn Epsilon klein ist, wählt der Agent in den meisten Fällen die Aktion, die er für am besten hält, und macht nur selten neue Aktionen. Wenn Epsilon jedoch groß ist, wählt der Agent häufiger neue Aktionen aus, um zu sehen, ob sie besser sind als die bisherigen. Epsilon = 1 steht dabei für 100% Wahrscheinlichtkeit für Exploration und Epsilon = 0 dementsprechend für 0%. [Sutton+2018]

### 3.4.2 Bellman-Gleichung

Die Bellman-Gleichung ist ein wichtiger Bestandteil des Q-Learning-Verfahrens und wird in den nachfolgenden dazu genutzt, um die Q-Werte für den Agenten zu aktualisieren. Sie beschreibt, wie sich der Q-Wert für einen Zustand und eine Aktion ändert, wenn der Agent von einem Zustand zu einem anderen übergeht und dabei eine Aktion ausführt. [Sutton+2018]

Das Ziel der Bellman-Gleichung ist es, den Q-Wert für jeden Zustand und jede Aktion so zu aktualisieren, dass er dem maximal erwarteten Belohnungswert entspricht. Der maximale erwartete Belohnungswert ist der Belohnungswert, den der Agent erwarten würde, wenn er immer die Aktion wählt, die den höchsten Belohnungswert verspricht. [Sutton+2018]

Die Bellman-Gleichung lautet wie folgt:

$$Q(S_t, A_t) = E[R_{t+1} + \gamma \max Q(S_{t+1},\ A_{t+1})]$$

In dieser Gleichung ist $Q(S_t, A_t)$ der Q-Wert für den Zustand $S$ und die Aktion $A$, $R_{t+1}$ ist der sofortige Belohnungswert, $\gamma$ ist der diskontierte zukünftige Belohnungswert und $maxQ(S_{t+1}, A_{t+1})$ ist der maximal erwartete Belohnungswert für den nächsten Zustand $S_{t+1}$. [Sutton+2018]

Die Bellman-Gleichung wird verwendet, um den Q-Wert für einen Zustand und eine Aktion auf der Grundlage der Belohnungen und der Q-Werte der nächsten Zustände zu aktualisieren. Wenn der Agent von einem Zustand zu einem anderen übergeht und dabei eine Aktion aus-

führt, wird der Q-Wert für den Zustand und die Aktion aktualisiert, indem der sofortige Belohnungswert $R_{t+1}$ und der diskontierte zukünftige Belohnungswert $\gamma * maxQ(S_{t+1}, A_{t+1})$ hinzugefügt werden. Der diskontierte zukünftige Belohnungswert $\gamma$ gibt an, wie stark zukünftige Belohnungen den Q-Wert beeinflussen. Wenn $\gamma$ klein ist, werden die Q-Werte weniger durch potientielle zukunftige Belohnungen beeinflusst und der Agent konzentriert sich mehr auf die Verbesserung der Gegenwart. Andersrum gilt, wenn $\gamma$ groß gewählt wird, versucht der Agent zukünftige Belohnungen zu maximieren. [Sutton+2018]

### 3.4.3 Aktualisieren der Q-Values

Um Q Values aktualisieren zu können, findet die Bellman Gleichen, wie Sie im vorigen Kapitel vorgestellt wurde Gebrauch. Die Gleichung zur Aktualisierung diesem Kapitel lautet die Gleichung wie folgt:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma * maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Ein Unterschied, der im Vergleich zur Standard Bellman Funktion auffällt, ist ein etwas anderer Aufbau und neuer Parameter α. [Sutton+2018]

α ist die learning rate (auch Lernrate genannt). Die restlichen Parameter sind Q(s, a) der Q-Wert für den Zustand s und die Aktion a, r ist der sofortige Belohnungswert, γ ist der diskontierte zukünftige Belohnungswert und maxa' Q(s', a') ist der maximal erwartete Belohnungswert für den nächsten Zustand s'. [Sutton+2018]

Die learning rate α gibt an, wie stark der aktuelle Durchlauf den Q-Wert beeinflusst. Wenn α ein kleiner Wert zugewiesen ist, werden die Q-Werte langsamer aktualisiert und der Agent lernt langsamer. Wenn α jedoch ein hoher Wert zugewiesen wird, werden die Q-Werte schneller aktualisiert und der Agent lernt schneller. Es ist wichtig, dass die learning rate angemessen gewählt wird, damit das gesamte Q-Learning-Verfahren effektiv laufen kann. Eine zu niedrige learning rate kann dazu führen, dass die Agent nicht genügend lernt, während eine zu hohe learning rate dazu führen kann, dass die Q-Werte instabil werden und das Lernen negativ beeinflussen. [Sutton+2018]

### 3.4.4 Deep Q-Learning

In den vorigen Kapiteln wurde Q-Learning und dessen Bestandteile erklärt. Jedoch stellt sich noch die Frage was man machen könnte wenn die Anzahl der möglichen Statusse unbegrenzt

ist? Dann lässt sich hierfür nämlich keine Q-Tabelle anlegen. Eine Methode um dieses Problem anzugehen nennt sich Deep Q-Learning. [Sutton+2018]

Deep Q-Learning ist ein Reinforcement Learning Algorithmus, bei dem ein neuronales Netz verwendet wird, um die optimale Q-Funktion zu finden. Eine Q-Funktion gibt an, welche Belohnung ein Agent durch die Ausführung einer Aktion in einem bestimmten Zustand erwarten kann. Die Eingaben des neuronalen Netzes sind der aktuelle Zustand und die Ausgaben sind die voraussichtliche Belohnung für jede mögliche Aktion. Ein Neuronales Netz welches zu diesem Zweck aufgespannt wurde nennt man auch “Deep Q-Network” oder abgekürzt DQN. [Sutton+2018]

Damit das neuronale Netz lernen kann, werden die Gewichte der Neuronen mittels Backpropagation und SGD aktualisiert. Die Aktualisierung erfolgt allerdings im Vergleich zum normalen neuronalen Netz auf eine andere Weise. In einem gewöhnlichem neuronalem Netz wird das richtige Ergebnis dem Netz am Ende mitgegeben und die Gewichte können anhand eines Vergleiches zwischem richtigem Ergebnis und Ergebnis des neuronalen Netztes angepasst werden. [Sutton+2018]

Bei DQN wird dem Netz allerdings nicht das richtige Ergebnis mitgegeben, sondern es wird ein Vergleich zwischen aktuellem und optimalen Q-Wert gezogen. Der optimale Q-Wert wird aus der Bellman Funktion wie in Kapitel X.X beschrieben gewonnen. Diese beiden Werte werden voneinander subtrahiert, woraus der sog. Loss kalkuliert wird. Das neuronale Netz versucht diesen Loss so gering wie nur möglich zu halten und versucht daraufhin mittels Backpropagation und SGD die Gewichte der Neuronen so anzupassen, dass sich dieser Loss auf ein Minimum reduziert. [Sutton+2018]

## 3.5 K-Means Clustering

Clustering ist ein Data Mining Verfahren, dessen Ziel es ist Datensätze in Klassen oder auch Mengen zu gruppieren. Der Unterschied zur Klassifikation ist, dass diese Klassen im Voraus noch nicht bekannt sind. Aus diesem Grund spricht man bei Clustering auch vom „unüberwachtem Lernen“. ([von der Hude 2020], [Han+2001])

Folgende Abbildung zeigt ein mögliches Ergebnis eines Clustering Verfahrens.

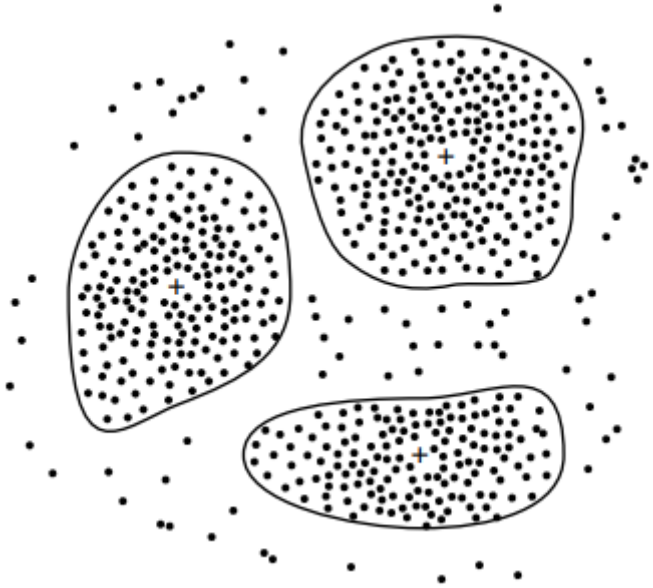

Abbildung 3.5: Mögliches Ergebnis eines Clustering Verfahrens [Han+2001]

Wie in Abbildung 2.5 zu sehen, lassen sich drei gebildete Cluster, gekennzeichnet durch die Grenzen, sowie die Datensätze gekennzeichnet als Punkte erkennen. Das Zentrum eines jeweiligen Clusters lässt sich am „+“ Zeichen erkennen. Die Bedeutung der Cluster ist zum Zeitpunkt der Bildung noch nicht bekannt und muss im Anschluss in der Evaluation interpretiert werden. In der Evaluation werden die Gemeinsamkeiten der Datensätze eines Clusters betrachtet, um daraus schließen zu können weswegen gewisse Datensätze in einem zusammengekommen sind. Diese Gemeinsamkeiten führen dann einerseits zu einem allgemeinem Informationsgewinn und andererseits zu einer Klassifikation der Datensätze innerhalb eines Clusters. Parallel dazu lassen sich Datensätze validieren welche keine Zugehörigkeit besitzen oder eher als Außenseiter auffallen. ([von der Hude 2020], [Han+2001])

Um zu verstehen wie Cluster gebildet werden können, wird nun der K-Means Algorithmus betrachtet, welcher häufig für Clustering Aufgaben verwendet wird.

Der K-Means Clustering Algorithmus ist im Data Mining Bereich ein beliebtes Verfahren, um Datensätze zu Clustern. Ein Vorteil des K-Means Algorithmus ist die Effizienz, die dieser mit sich bringt. Dadurch ist der Algorithmus vor allem für sehr große Datenmengen gut geeignet. Nachteilig hingegen ist, dass man die Anzahl der Cluster selber bestimmen muss. Dadurch wird ein etwas höhere Aufwand hervorgerufen, um eine passende Anzahl an Clustern bestimmen zu können. Ein weiterer Nachteil ist, dass im K-Means Algorithmus jeder Datensatz

einem Cluster zugeordnet wird. Das hat zur Folge, dass die Interpretation etwas aufwendiger werden kann, da Ausreißer ebenfalls eine Clusterzugehörigkeit finden. [von der Hude 2020]

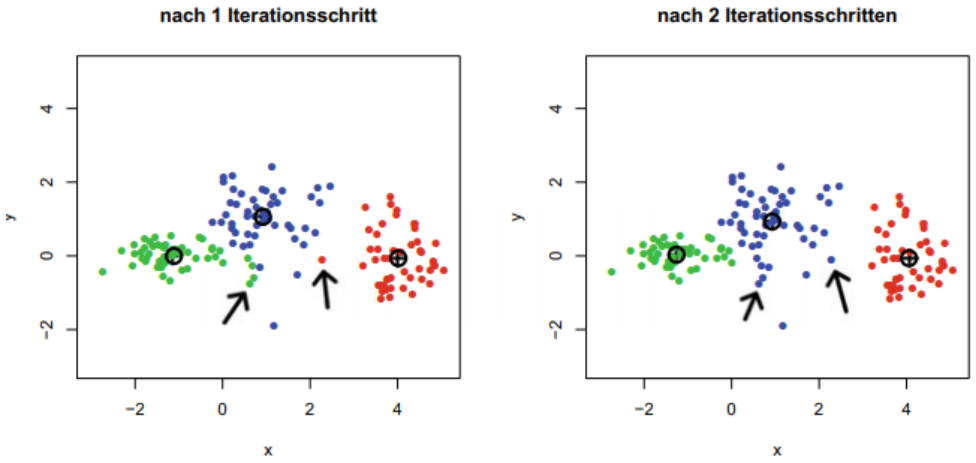


Abbildung 3.6: Iterationsschritt K-Means Clustering [Von der Hude 2020]

K-Means ist ein iterativer Algorithmus, das bedeutet ein Algorithmus der sich so lange wiederholt bist ein gewisses Ereignis eintritt. Abbildung 2.6 kann parallel mit der Erklärung zum Verständnis betrachtet werden. [von der Hude 2020]

Schritt eins: Festlegung der Clusteranzahl *k*.

Schritt zwei: Auswahl von k Startobjekten als Clusterzentren.

Schritt drei: Zuweisung von Datensätzen zu einem der Clusterzentren auf Basis des geringsten Abstands. Als Abstandsmaß wird hierfür standardmäßig die euklidische Distanz verwendet wie sie in Kapitel 2.3 vorgestellt wurde.

Schritt vier: Neuberechnung der Clusterzentren. Bei Abweichung zur aktuellen Position werden die Zentren neu gesetzt.

Schritt fünf: Haben sich in Schritt vier die Position eines oder mehreren Clusterzentren verändert, so wird wieder ab dem dritten Schritt weitergemacht. Hat sich keine Position eines Zentrums verändert, ist der Algorithmus abgeschlossen.

## 3.6 Weitere Verwendete Funktionen

### 3.6.1 Time to Collision (TTC)

TTC ist ein Indikator für die verbleibende Zeit bis zum Zusammenstoß zweier Fahrzeuge, wenn sich beide mit ihrer aktuellen Geschwindigkeit und Richtung fortbewegen. Der Parameter wurde in dieser Bachelorarbeit dabei auf folgende Weise berechnet. [Jiménez+2013]

*Relativgeschwindigkeit = Geschwindigkeit Fahrzeug 1 - Geschwindigkeit Fahrzeug 2*

Die Relativgeschwindigkeit beschreibt, wie schnell zwei Fahrzeuge gerade zueinander aufholen. [Jiménez+2013]

*TTC = Abstand zwischen Fahrzeugen / Relativgeschwindigkeit*

Der TTC-Parameter wird daraufhin berechnet, indem der Abstand zwischen den beiden Fahrzeugen durch die Relativgeschwindigkeit geteilt wird. Die Formel gibt die Zeitangabe in Sekunden an, die verbleibt, bis sich die Fahrzeuge treffen, wenn sie mit unveränderter Geschwindigkeit und Richtung weiterfahren. Wenn TTC einen negativen Wert annimmt, bedeutet das, dass sich die Fahrzeuge bereits auf Kollisionskurs befinden. [Jiménez+2013]

TTC wird oft als Warnparameter in Fahrassistenzsystemen eingesetzt, um eine drohende Kollision frühzeitig zu erkennen und den Fahrer zu warnen. Auch im Bereich der Verkehrsforschung wird TTC verwendet, um die Sicherheit von Verkehrssituationen zu bewerten und Unfallrisiken zu minimieren. Allerdings ist zu beachten, dass TTC von vielen Faktoren beeinflusst wird und nicht immer zuverlässige Ergebnisse liefert. [Jiménez+2013]

### 3.6.2 Bremsweg

Der Bremsweg ist der Weg, den ein Fahrzeug benötigt, um zum vollständigen Stillstand zu kommen. Die Formel setzt sich aus dem Reaktionsweg und dem Bremsweg zusammen. Der Reaktionsweg ist dabei der Weg, den das Fahrzeug während der Reaktionszeit des Fahrers zurücklegt. [Green 2000]

Um den Bremsweg zu berechnen, müssen Reaktionsweg und Bremsweg separat berechnet werden. Der Reaktionsweg kann durch Multiplikation der Geschwindigkeit des Fahrzeugs mit der Reaktionszeit des Fahrers berechnet werden:

*Reaktionsweg = Geschwindigkeit des Fahrzeugs * Reaktionszeit des Fahrers*

Der Bremsweg hingegen hängt von verschiedenen Faktoren ab, wie der Geschwindigkeit des Fahrzeugs, der Reibung zwischen den Reifen und der Fahrbahn, dem Gewicht des Fahrzeugs sowie dem Zustand der Bremsen und Reifen. Eine mögliche Formel zur Berechnung des Bremswegs lautet:

*Bremsweg = (Geschwindigkeit des Fahrzeugs in km/h / 10) * (Geschwindigkeit des Fahrzeugs in km/h / 10) * Bremswegfaktor + Reaktionsweg*

Der Bremswegfaktor variiert je nach Fahrbahnbeschaffenheit und beträgt z.B. bei trockener Fahrbahn etwa 0,4 und bei nasser Fahrbahn etwa 0,8. [Green 2000]

Es ist wichtig zu beachten, dass der Bremsweg nur eine Näherung darstellt und von vielen Faktoren beeinflusst wird. Je nach Situation kann der Bremsweg kürzer oder länger sein als berechnet. [Green 2000]

## 3.7 Verwendete Technologien und Libraries

### 3.7.1 CSV

Die csv (Comma Separated Values) Bibliothek in Python ermöglicht es, CSV-Dateien zu lesen und zu schreiben. CSV-Dateien sind ein Format zur Speicherung von tabellarischen Daten, bei dem jede Zeile eine neue Datenzeile darstellt und die Spaltenwerte durch Kommas getrennt sind. [Döhmen 2016]

Die csv-Bibliothek in Python enthält zwei wichtige Klassen: Reader und Writer. Reader liest eine CSV-Datei ein und wandelt diese dementsprechend in das richtige Python-Format um, zum Beispiel ein Array. Writer schreibt dagegen eine CSV-Datei auf die Basis eines Arrays. Erweiterungen zum Standard Reader/Writer wären Klassen wie DictReader oder DictWriter, welche den Standard um Python Dictionaries Erweitern. In dieser Masterarbeit wurde die CSV Bibltiothek verwendet, um die gewonnen Daten einlesen zu können und in Python damit arbeiten zu können. [Döhmen 2016]

### 3.7.2 Python

Python ist eine allgemein verwendete, hochgradig interpretierte Programmiersprache. Sie wurde 1989 von Guido van Rossum entwickelt und hat bis heute eine große Popularität er-

langt. Python wirbt mit einer leicht verständlichen Syntax, die es vor allem Anfängern erleichtert, schnell damit vertraut zu werden. Die verständliche Syntax hat dabei viele Befürworter, sodass Python eine aktive und wachsende Community vorweist, die ständig neue Bibliotheken und Tools entwickelt, die die Entwicklung erleichtern. Python ist eine vielseitige Sprache, die sowohl für Webentwicklung, Datenanalyse, maschinelles Lernen, als auch für die Entwicklung von Desktop- und Mobilanwendungen verwendet werden kann. [Python 2023]

Python wird heutzutage in verschiedenen Bereichen angewendet, darunter Wissenschaft, Finanzen, Webentwicklung, Datenanalyse und auch maschinelles Lernen. Die Python Version, die beim Entwicklungsteil dieser Masterarbeit verwendet wurde, ist Version 3.7.4. [Python 2023]

### 3.7.3 Tensorflow

TensorFlow ist eine Open-Source-Bibliothek für maschinelles Lernen, die von Google entwickelt wurde. Die Bibliothek ermöglicht es Entwicklern, Machine Learning Modelle zu definieren, zu trainieren und zu entwicklen. TensorFlow bietet dabei eine umfangreiche Palette an Algorithmen und Tools an, die es Entwicklern erleichtern, komplexe Modelle für eine Vielzahl von Anwendungen zu erstellen. [Tensorflow 2023]

TensorFlow ist vor allem für die Unterstützung von Neuronale Netzen bekannt, insbesondere von Deep Learning-Modellen. Dabei ermöglicht die Bibliothek Entwicklern, komplexe Netzwerke mit mehreren Schichten und Neuronen zu erstellen und zu trainieren. TensorFlow unterstützt dabei neben CPU-Berechnungen Kalulationen auch GPU-Berechnungen, was es ermöglicht, große Modelle schneller zu trainieren und fertigzustellen. [Tensorflow 2023]

Ein weiteres wichtiges Merkmal von TensorFlow ist die Möglichkeit, Modelle zu exportieren und in Produktionsumgebungen einzusetzen. Es unterstützt eine Vielzahl von Plattformen und Sprachen, darunter iOS, Android, JavaScript und C++. [Tensorflow 2023]

Neben den obigen genannten Features, bietet TensorFlow auch eine große Anzahl von vorab trainierten Modellen und Tools an. Diese können beispielsweise zur Bilderkennung, Sprachverarbeitung oder andere Anwendungen von Entwicklern verwendet werden, um schnell und einfach Modelle für ihre Projekte zu erstellen, ohne dass sie von Grund auf trainiert werden müssen. [Tensorflow 2023]

### 3.7.4 Matplotlib

Matplotlib ist eine Open-Source-Bibliothek für Python, die es ermöglicht, Diagramme und Visualisierungen zu erstellen. Die Bibliothek bietet eine breite Auswahl an Funktionen für 2D- und 3D-Diagramme an. Einige Beispiele für Visualisierungen wären Linienplots,Bar-Charts, Scatterplots, Histogramme und viele mehr. Matplotlib kann parallel dazu auch verwendet werden, um die erstellen Diagramme in einer Vielzahl von Formaten wie PNG, PDF, SVG und JPG zu exportieren. [Matplotlib 2023]

Ein Vorteil von Matplotlib ist, dass die erstellten Diagramme einerseits eine große Flexibilität anbieten können und andererseits bei richtiger Nutzung sich als äußerst detailliert erweisen können. Ein Beispiel für die Flexibilität wären die Anpassungsmöglichkeiten wie Farben, Marker oder auch Linienstile. [Matplotlib 2023]

Matplotlib wird in einer Vielzahl von Anwendungen verwendet. So findet die Biblitiothek in Bereichen wie z.B. in Wissenschaft, Finanzen, Datenanalyse und andere Gebiete Gebrauch, in denen es von großer Bedeutung sein kann, Daten visuell darzustellen und zu analysieren. Es ist zudem auch eine gängige Wahl für die Erstellung von Plots in wissenschaftlichen Arbeiten und Präsentationen. Im Falle dieser Masterarbeit findet Matplotlib genau hierfür Gebrauch, wie beispielsweise Abbildung X.X oder Abbildung X.X. [Matplotlib 2023]

### 3.7.5 Numpy

NumPy (Numerical Python) ist eine Open-Source-Bibliothek für Python, die es ermöglicht, effizient mit großen Arrays und Matrizen von numerischen Daten zu arbeiten. Die Bibliothek bietet eine Vielzahl von Funktionen, die es Entwicklern erleichtern, numerische Berechnungen und Array Operationen durchzuführen. [Numpy 2023]

Ein besonderer Vorteil von NumPy ist dessen Effizienz im Umgang mit großen Datensätzen. Sogenannte “NumPy-Arrays“, welche mit Numpy Funktionen Bearbeitet werden, sind oftmals schneller und effizienter als Listen der Python Standardbibliothek und bieten dazu auch noch eine größere Vielzahl von Funktionen für die Verarbeitung von Array-Daten an. [Numpy 2023]

Einige Beispiele für die Verarbeitung von Array-Daten, wären statistische Funktionen, Sortierfunktionen und Funktionen zur Änderung der Form von Arrays. Numpy arbeitet zudem

auch gut mit anderen Bibliotheken wie Matplotlib oder auch Tensorflow, welche beide in dieser Arbeit verwendet werden, zusammen. [Numpy 2023]

NumPy ist eine der am häufigsten verwendeten Bibliotheken für Datenanalyse in Python und wird oft in Anwendungen wie, maschinelles Lernen, Finanzanalyse und vielen anderen Anwendungsbereichen verwendet. Insgesamt trägt Numpy in dieser Masterarbeit die Weiterverarbeitung von großen Datensätzen erleichtert, um schnell und effizient große Mengen an Daten zu verarbeiten. [Numpy 2023]

# 4 Anforderungen und Vision

Bevor mit der Entwicklung begonnen wird, werden in diesem Kapitel die Anforderungen und Visionen für das Projekt festgelegt. Es wurde erkannt, dass das Gesamtprojekt für eine direkte Entwicklung zu umfangreich ist und es wird daher beschlossen, es in zwei Teilprojekte aufzuteilen. Durch die Fusion der Ergebnisse beider Teilprojekte kann daraufhin schließlich das gesamte Projekt abgebildet werden. Die beiden Teilprojekte die gebildet werden lauten demnach Verkehrsteilnehmerklassifikation und Relevanzklassifikation.

## 4.1 Verkehrsteilnehmerklassifikation

Die Verkehrsteilnehmerklassifikation beschäftigt sich damit Positionsdaten entgegenzunehmen, diese Daten zu verarbeiten und nach der Verarbeitung eine Ausgabe geben zu können, welche als eine Verkehrsteilnehmerklasse interpretiert werden kann. Die Vision, die daraus entstehen würde, wäre ein neuronales Netz, welche diese Aufgaben erfüllt. Darstellen und präzisieren könnte man dies mit der nachfolgenden Abbildung.

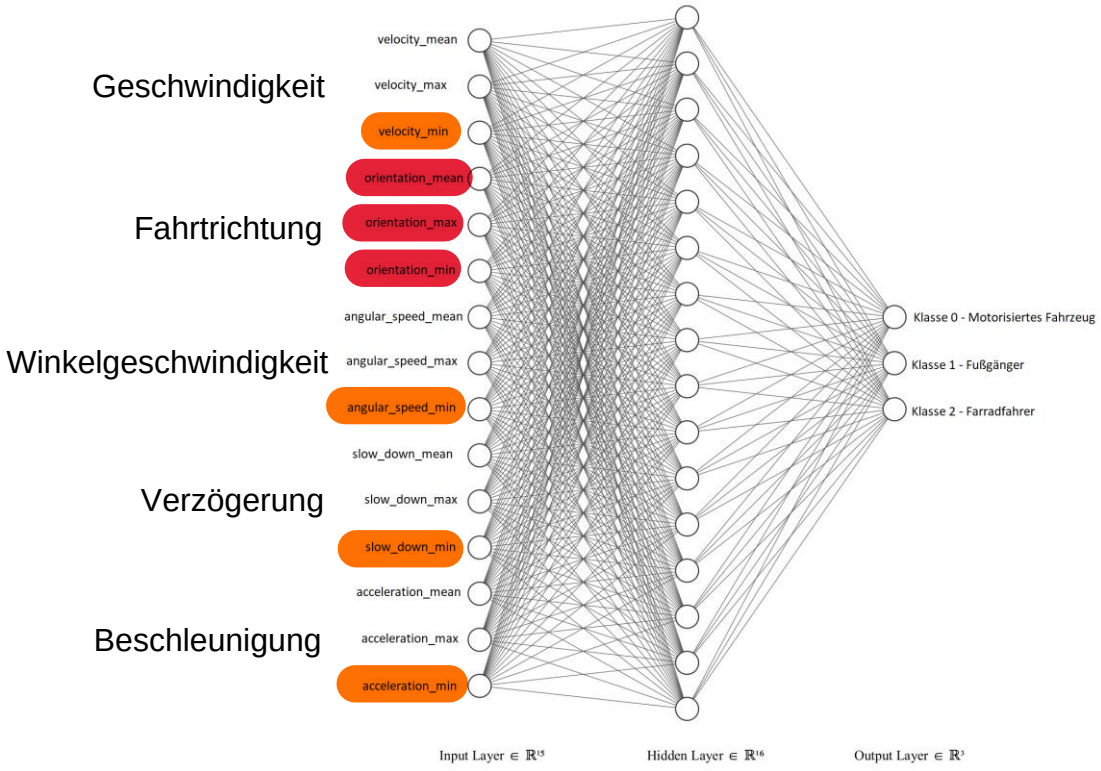


Abbildung 4.1 Vision der Verkehrsteilnehmerklassifikation

Betrachtet man die Abbildung der Vision, so lassen sich daraus Anforderungen herableiten. Die erste Anforderung wäre es die Daten so bereitzustellen, dass diese die Form wie in der Abbildung annehmen. Hierfür wäre die Weiterverarbeitung so weit notwendig, dass Ein- und Ausgabedaten übereinstimmen und Ergebnisse verglichen werden können.

Hinzukommen würde die optimale Architektur eines neuronalen Netzes herauszufinden, da diese je nach vorhanden Daten unterschiedlich ausfallen kann.

Zudem wäre es wünschenswert nicht nur ein Lernverfahren anzuwenden, sondern mehrere. Das dient dazu, um später in der Evaluation vergleiche ziehen zu können und die Stärken und Schwächen der eingesetzten Verfahren validieren zu können. Aus diesem Grund soll neben gewöhnlichen neuronalen Netzen mittels gewöhnlicher Backpropagation auch Reinforcement Learning zum Einsatz kommen.

## 4.2 Relevanzklassifikation

### 4.2.1 Definition

Was bedeutet Relevanz im Straßenverkehr eigentlich? Diese Frage wird oft gestellt, aber nicht immer eindeutig beantwortet. Dabei entstehen Fragen was einen Verkehrsteilnehmer relevant macht, aus welcher Sicht ein Teilnehmer relevant ist oder auch wie man Relevanz messen kann. Um Klarheit zu schaffen und einen tieferen Einblick in diese Materie gewinnen zu können, wurden im Rahmen dieser Masterarbeit verschiedene Paper durchgelesen und aus diesen Papern Inspiration gesammelt, wie man Relevanz im Straßenverkehr berechnen und darstellen könnte.

[Althoff+2009]: Das Paper versucht mithilfe von Markov Ketten die Wahrscheinlichkeit für einen Zusammenstoß mithilfe von Markov Ketten zu kalkulieren. Markov Ketten kann man sich als eine Matrix vorstellen, die die Wahrscheinlichkeiten von einem Statusübergang in den nächsten angeben. Multipliziert man die daraus resultierende Matrix mit einem Ausgangszustand, so erhält man die Wahrscheinlichkeiten für den Eintritt jedes möglichen Status.

[Xiong+2018]: Dieses Paper versucht ebenfalls mithilfe von Markov Ketten eine Wahrscheinlichkeit zu kalkulieren. Diesmal wird allerdings nicht die Wahrscheinlichkeit für einen Zusammenstoß kalkuliert, sondern die Wahrscheinlichkeit für Risiko. Hierbei werden erstmals mithilfe des K-Means Algorithmus Risikoklassen mithilfe von TTC und TTH gebildet. Im

Anschluss wird daraufhin die Markovkette gebildet, die die Wahrscheinlichkeitsübergänge von einer Risikoklasse in die andere Risikoklasse enthält.

[Jiménez+2013]: Dieses Paper wählt einen bisher komplett anderen Ansatz als die vorherigen. Hier wird versucht mithilfe des Parameters TTC, welcher auch in Kapitel X.X vorgestellt wurde Szenarien zu entwickeln. Dabei sollen möglichst viele Szenarien für Zusammenstöße abgedeckt werden, um daraufhin mithilfe des TTC-Parameters den Eintritt solcher Szenarien vermeiden zu können.

Mithilfe dieser Papers wurde Relevanz als die Antwort auf die folgende Frage definiert:

„Wie hoch ist die Wahrscheinlichkeit, dass ein Protagonist mit seinem zukünftigen Verhalten und dem zukünftigen Verhalten von Verkehrsteilnehmer XY zusammenstoßen wird?“

Diese Frage wird dabei immer aus sich eines Protagonisten (manchmal auch Hero genannt) gestellt und kann dabei helfen das geplante zukünftige Verhalten so zu verändern, sodass die Wahrscheinlichkeit eines Zusammenstoßes gemindert wird.

### 4.2.2 Vorstellung der Vision

Mithilfe der vorgestellten Papers aus Kapitel X.X konnte auch eine neue Idee zur Klassifikation von Relevanz entwickelt werden. Dabei wurde darauf geachtet, dass diese Idee keine Kopie der bisherigen Ansätze darstellt, sondern vielmehr die bisherigen Ansätze als Inspiration verwendet und darauf mit dem bisher erarbeiteten Wissen aufbaut. Die entwickelte Idee setzt sich dabei aus den folgenden Schritten zusammen:

1. Mithilfe des TTC-Parameters und Clustering Risikoklassen bilden und den gebildeten Clustern eine Bedeutung zuweisen. Die dabei herauskommenden Klassen sollen daraufhin die Basis für die spätere Klassifikation bilden.
2. Das bisherige aufgebaute Wissen Nutzen und mithilfe der neu klassifizierten Daten und Machine Learning Ansätzen ein Modell entwickeln, welches nach dem Training in der Lage ist, Datensätze richtig zu klassifizieren.
3. Jeder Verkehrsteilnehmerklasse einen Aktionsbereich für zukünftige Aktionen zuweisen. Dies kann ebenfalls mithilfe von Machine Learning Ansätzen gelöst werden. Wichtig ist hierbei, dass wirklich auf die einzelnen Klassen Rücksicht genommen wird, da ein Fußgänger im Regelfall nicht so stark wie ein Auto beschleunigen kann.

4. Daraufhin werden für jede mögliche zukünftige Aktion und die nächste geplante Aktion die TTC-Parameter ermittelt. Diese können dann an das Trainierte Modell von Schritt zwei gegeben werden und es kann ermittelt werden ob in der Zukunft angemessene Risikoklassen beibehalten werden können.

Dieser Ansatz könnte dadurch funktionieren, dass die Relevanzklassifikation aus der Sicht eines Protagonisten erfolgt. Der sequenziell nächste geplante Schritt ist dem Protagonisten somit bekannt, wodurch eine präzise Schätzung des nächsten Schrittes anderer Verkehrsteilnehmer bei der Berechnung des TTC-Parameters ebenfalls präzise werden kann. Insgesamt ergibt sich aus dem bisher geschriebenen das Bild einer Version, welches mit der nachfolgenden Abbildung verglichen werden kann.

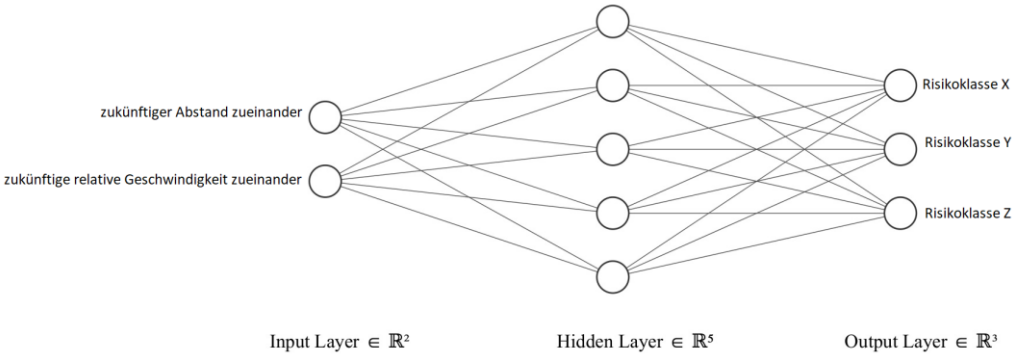


Abbildung 4.2 Vision der Relevanzklassifikation

Um diese Vision zu realisieren, ist nicht nur die Erfüllung der zuvor genannten Anforderungen erforderlich, sondern auch die Entwicklung eines Modells, das in der Lage ist, eine Geschwindigkeitsprognose für einen potenziellen Verkehrsteilnehmer zu erstellen.

### 4.2.3 Vorstellung der Geschwindigkeitsprognose

Im vorigen Kapitel wurde definiert, dass zur Erfüllung der Vision ein Modell benötigt wird, welches in der Lage ist, eine Geschwindigkeitsprognose für einen potenziellen Verkehrsteilnehmer zu erstellen. Demnach wird bei der Entwicklung eines solchen Modells versucht einen Wert so exakt wie möglich zu prognostizieren. Die Idee wie so ein Netz aussehen könnte, soll die nachfolgende Abbildung darstellen.

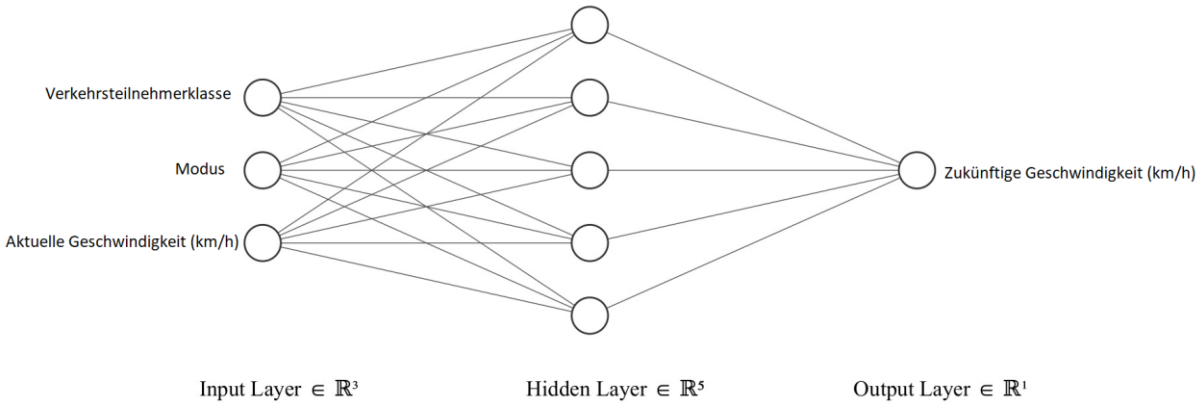


Abbildung 4.3 Vision der Geschwindigkeitsprognose

Die Übergabe der Verkehrsteilnehmerklasse wird dabei verwendet, da in den Daten verschiedene Verkehrsteilnehmerklassen sind. Ein Fahrradfahrer könnte beispielsweise nicht so schlecht beschleunigen wie ein Autofahrer, daher soll dies vom Modell berücksichtigt werden.

Zudem folgt ein sog. „Modus“, dieser gibt an welche Aktion für die aktuelle Geschwindigkeit und der aktuellen Klasse durchgeführt werden soll. Folgende Modi sind zum aktuellen Zeitpunkt vorgesehen.

| Modus | Beschreibung |
|---|---|
| 0 | Schwaches Beschleunigen |
| 1 | Normales Beschleunigen |
| 2 | Starkes Beschleunigen |
| 3 | Schwaches Abbremsen |
| 4 | Normales Abbremsen |
| 5 | Starkes Abbremsen |

Tabelle 1: Vorstellung der Modi

Der letzte Parameter ist die aktuelle Geschwindigkeit des Verkehrsteilnehmers. Diese soll die Basis für die Prognose der nächsten Geschwindigkeit bilden. Würde man dem Modell als Eingabe die Parameter [0, 2, 10] geben, so ließe sich das interpretiere als:

„Prognostiziere die Geschwindigkeit eines Autos bei Normaler Beschleunigung und einer Ausgangsgeschwindigkeit von 10km/h“.

Um dieses Unterfangen angehen zu können, wird ein gewisses Maß an Datenverarbeitung notwendig sein. Vor allem die Berechnung der Modi wird in der späteren Entwicklung voraussichtlich eine Rolle spielen.

# 5 Datengewinnung und Weiterverarbeitung

Voraussetzung für die erfolgreiche Bearbeitung eines Machine Learning Projektes sind eine ausreichende Menge an qualitativ hochwertigen Daten. Ohne ausreichend Daten kann in den Lernprozess schlichtweg nicht genügend Variation einfließen, sodass ein guter und breit aufgestellter Lernprozess nicht gewährleistet werden kann. Aus diesem Grund ist die Gewinnung von Daten ein wichtiger Bestandteil dieser Masterarbeit. In diesem Kapitel wird betrachtet welche Daten gewonnen wurden, wie diese Daten gewonnen wurden und welche Weiterverarbeitung die Daten für die weitere Bearbeitung des Projekts unterzogen wurden.

## 5.1 Datengewinn über CARLA

Die ersten Daten, die in diesem Projekt gewonnen wurden, waren Daten der Verkehrssimulation CARLA. Eine Kurzübersicht zu CARLA lässt sich in Kapitel X.X finden. Für die Gewinnung von Daten mithilfe von CARLA wurde der Ansatz und Code der Bachelorarbeit [Torlak 2022] verwendet, welche bereits in Kapitel X.X eine Kurze Erwähnung gefunden hat.

Das Verfahren wurde in der Programmiersprache Python entwickelt, die hierfür verwendete Version von CARLA war die Version 0.9.12.

Das Endergebnis von [Torlak 2022] beginnt mit einer Python-Datei namens „example.py“. Im Code dieser Datei lassen sich Hyperparameter wie beispielsweise Simulationsstadt, Anzahl an Samples pro Datensatz, Verfälschungsfaktor oder auch Zeitabstand pro Aufnahme setzen. Folgende Parameter wurden für die Erfassung der Daten gesetzt.

**Simulationsstadt** – City02

**Anzahl an Samples pro Datensatz** – 100

**Verfälschungsfaktor** – 0 Meter

**Zeitabstand pro Aufnahme** – 0.5 Sekunden

Durch diese Festlegung der Hyperparameter enthält ein Datensatz das Fahrverhalten eines Verkehrsteilnehmers über einen Zeitraum von 60 Sekunden.

Nach Setzung der Hyperparameter beginnt der Code mit der Festlegung eines Verkehrsteilnehmers zum sog. „Hero“ (zu Deutsch auch Held genannt). Der Held wird unter anderem verwendet, um Daten in Relation zu sich selbst wiedergeben zu können. Ein konkretes Bei-

spiel hierfür wäre ein Datensatz namens „distance_to_hero“, welches den Abstand eines Verkehrsteilnehmers zum Helden beschreibt.

Das bedeutet alle wiedergegebenen Daten werden immer aus der Sicht des Helden zurückgegeben.

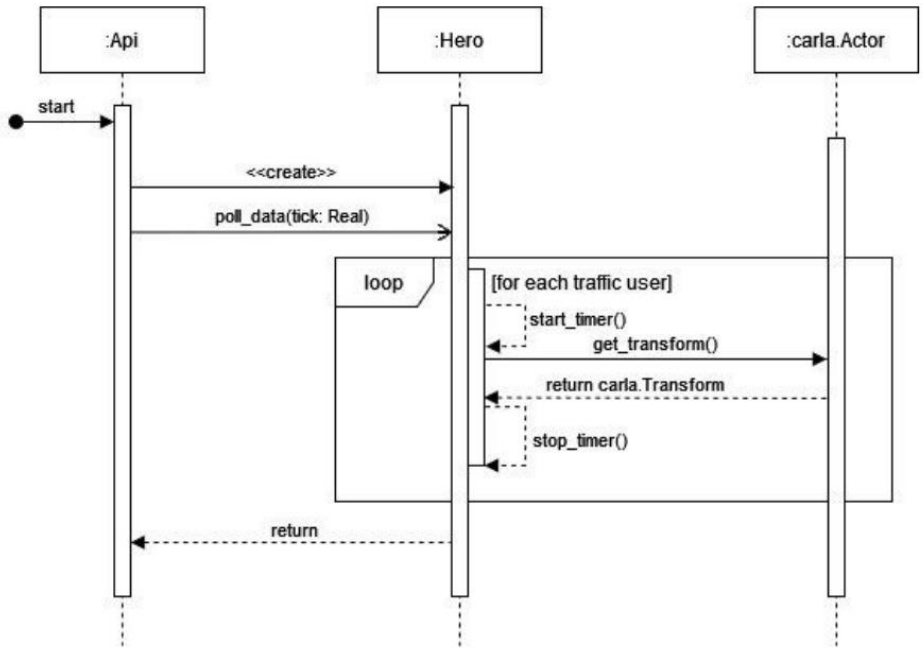


Abbildung 5.1: Ablaufdiagram der CALRA Datenerfassung [Torlak 2022]

Der Gesamtablauf des Konzepts gliedert sich demnach nach der folgenden Vorgehensweise.

1. Festlegung eines Helden,

2. Iteration über alle Verkehrsteilnehmer,

3. in jeder Iteration werden die Positionsdaten des aktuellen Verkehrsteilnehmers erfasst und berechnet.

4. Nachdem über alle Verkehrsteilnehmer iteriert wurde, werden alle gesammelten Daten an die API zurückgegeben und können demnach für weitere Zwecke verwendet werden.

Wenn man das Programm beendet, wird eine CSV-Datei erstellt, welche für alle Verkehrsteilnehmer (Held ausgeschlossen) die festgelegten Positionsdaten speichert.

Es wurden insgesamt 5781 Datensätze erfasst und die Daten wurden in einer Datei namens Map_03_0m_erorr.CSV gespeichert. Diese Datei wird als die Basis für die Weiterverarbeitung der Daten hergenommen.

## 5.2 Gewinn von echten Verkehrsdaten


Abbildung 5.2: Bildschirmaufnahme aus der App für die Sammlung von realen Daten

Neben den Gewinn von simulierten Verkehrsdaten über CARLA, konnten im Rahmen dieser Arbeit auch echte Verkehrsdaten gesammelt werden. Diese wurden mithilfe einer App gesammelt, welche durch die Forschung und Entwicklung der Hochschule Coburg bereitgestellt wurde. Die App misst dabei in einem Zeitabstand von 0.5 Sekunden die GPS-Koordinaten des jeweiligen Handys aus und speichert nach Beendung der App alle gemessenen Koordinaten lokal in einer .txt-Datei. Das Format, in dem die Daten darin gespeichert wurden, ist das JSON-Format. Die Daten müssen demnach erstmals in das Ausgangsformat gebracht, wie es in Tabelle X.X bereits vorliegt. Hierfür werden die Formeln von [Torlak 2022] verwendet, um die GPS-Koordinaten in die notwendigen Positionsdaten umwandeln zu können. Die weitere Weiterverarbeitung gestaltet sich, nachdem die Daten das Ausgangformat erhalten, haben gleich und lässt sich somit für simulierte als auch für die echten Daten anwenden.

Die Ausgangsdaten haben dabei die folgende Form, wie sie in der nachfolgenden Tabelle beschrieben werden.

| Bezeichnung | Datentyp | Bedeutung |
|---|---|---|
| Latitude | Float | Der Längengrad einer GPS-Koordinate |
| Longitude | Float | Der Breitengrad einer GPS-Koordinate |
| Velocity | Float | Die Geschwindigkeit im Vergleich zu letzter Koordinate, dieser Parameter wurde allerdings nicht verwendet, da die Werte nicht korrekt berechnet wurden |

Tabelle 2: Datenform nach Sammlung von echten GPS-Koordinaten

## 5.3 Betrachtung der Ausgangsdaten

Nachdem in den vorigen Kapiteln der Gewinn der Daten ausgeführt wurde, wird in diesem Kapitel die Form der Ausgangsdaten präsentiert. Die nachfolgende Tabelle stellt dabei diese Form da und wird die Basis für jede notwendige Weiterverarbeitung darstellen. Der Dateiname für die simulierten Daten lautet Map_03_0m_error.csv und beinhaltet 5871 Datensätze. Für die echten Daten lautet der Dateiname realdaten_weitververabeitet_richtiges_format.csv und beinhaltet 433 Datensätze.

| Bezeichnung | Datentyp | Bedeutung |
|---|---|---|
| ID | Integer | Die Identifikationsnummer eines Verkehrsteilnehmers |
| Type | Integer | Die Klasse eines Verkehrsteilnehmers<br>0 = Auto<br>1 = Fußgänger<br>2 = Truck<br>3 = Motorrad<br>4 = Fahrrad |

| | | |
|---|---|---|
| Velocity 0 bis 99 | Float | 100 gemessene sequentielle Geschwindigkeiten eines Verkehrsteilnehmers in km/h.<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |
| Orientation 0 bis 99 | Float | 100 gemessenen sequentiellen gemessenen Ausrichtungen eines Verkehrsteilnehmers in Grad.<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |
| Angular Speed 0 bis 99 | Float | 100 gemessene Winkelgeschwindigkeiten eines Verkehrsteilnehmers in km/h.<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |
| Accelaration 0 bis 99 | Float | 100 gemessene Beschleunigungen eines Verkehrsteilnehmers in km/h<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |
| distance_to_hero 0 bis 99 | Float | 100 gemessene Abstände eines Verkehrsteilnehmers in Relation zum Helden in Meter.<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |
| angle_to_hero 0 bis 99 | Float | Die letzten zehn gemessenen Winkel eines Verkehrsteilnehmers in Relation zum Helden in Grad.<br>Bei den Echtdaten sind allerdings nur 10 davon befüllt, um mehr Datensätze zu haben. |

Tabelle 3: Datenform der Ausgangsdaten

## 5.4 Weiterverarbeitung für Verkehrsteilnehmerklassifikation

Würde man alle Variablen so verwenden, wie sie in Tabelle X.X vorgestellt wurden, so müsste man ein neuronales Netz entwickeln, welches 600 Neuronen in der Eingangsschicht benötigt. Allein aus Sicht der Performanz wäre dies keine optimale Verwendung der vorhandenen Daten. Aus diesem Grund werden die Daten nach Ihrer Erfassung nochmals weiterverarbeitet, damit diese optimal für die Entwicklung eines neuronalen Netzes verwendet werden können.

Der Code, der zur Weiterverarbeitung der Daten geschrieben wurde, lässt sich in der Projektabgabe finden. Zur Weiterverarbeitung wurde der Ansatz verwendet aus jeder Variable das Minimum, Maximum und den Durchschnitt zu berechnen. Parallel dazu konnte mithilfe der Variable Acceleration (Beschleunigung) auch eine Variable für Geschwindigkeitsreduktion errechnet werden. Somit konnten die Daten zu der nachfolgenden Form weiterverarbeitet werden, wie die nachfolgende Tabelle zeigt. Diese Daten werden für Kapitel X.X verwendet.

| Bezeichnung | Datentyp | Beschreibung |
|---|---|---|
| velocity_min | Float | Die kleinste gemessene Geschwindigkeit |
| velocity_max | Float | Die größte gemessene Geschwindigkeit |
| velocity_avg | Float | Die durchschnittliche gemessene Geschwindigkeit |
| orientation_min | Float | Die kleinste gemessene Ausrichtung |
| orientation_max | Float | Die größte gemessene Ausrichtung |
| orientation_avg | Float | Die durchschnittliche gemessene Ausrichtung |
| angular_speed_min | Float | Die kleinste gemessene Winkelgeschwindigkeit |
| angular_speed_max | Float | Die größte gemessene Winkelgeschwindigkeit |
| accelaration_min | Float | Die kleinste gemessene Beschleunigung |
| accelaration_max | Float | Die größte gemessene Beschleunigung |
| accelaration_avg | Float | Die durchschnittliche gemessene Beschleunigung |
| slow_down_min | Float | Die kleinste gemessene Bremsung |
| slow_down_max | Float | Die größte gemessene Bremsung |
| slow_down_avg | Float | Die durchschnittliche gemessene Bremsung |

Tabelle 4: Datenform für die Trainingsdaten der Verkehrsteilnehmerklassifikation

Zudem wird noch eine zweite CSV-Datei gebildet, welche die jeweilige Klasse eines Datensatzes enthält. Der Inhalt der Datei hat dabei die folgende Form.

| Bezeichnung | Datentyp | Beschreibung |
|---|---|---|
| Type | Integer | Die Klasse eines Verkehrsteilnehmers<br>0 = Motorisiertes Fahrzeug<br>1 = Fußgänger<br>2 = Fahrradfahrer |

Tabelle 5: Datenform der wahren Klassen für die Verkehrsteilnehmerklassifikation

Diese Form der Weiterverarbeitung hat den Vorteil, dass erstmals eine Reduktion von 600 Einträgen auf 15 Einträgen unter der Beibehaltung von 5781/433 Datensätzen erfolgt. Das bedeutet, dass in der Entwicklung des neuronalen Netzes nur noch 15 Neuronen in der Eingangsschicht gebraucht werden. Parallel dazu ergibt sich durch die Auswahl von Minima, Maxima und Durchschnitt ein Ergebnis, in der jeder Wert von den Einträgen einen Einfluss ausüben kann. Die weiterverarbeiteten Daten wurden dabei aufgeteilt in Trainingssatz und Validierungssatz. Dies wurde für die simulierten als auch für die echten Daten vollzogen. Die Dateinamen haben hierfür ein jeweiliges Präfix oder Suffix zur Identifizierung erhalten.

## 5.5 Weiterverarbeitung für Klassifikation von Risikoklassen

In Kapitel X.X werden mithilfe von Clustering Risikoklassen gebildet und anhand dieser Risikoklassen können Modelle trainiert werden, die sich auf die Erkennung solcher Risikoklassen spezialisieren. Damit dieses Vorhaben ermöglicht werden kann, müssen die Daten dementsprechend weiterverarbeitet werden. Hierbei finden Teile, die zur Berechnung des TTC-Parameters verwendet werden, gebrauch, diese wurden in Kapitel X.X vorgestellt. Die Daten wurden dabei so bearbeitet, sodass zwischen Helden und anderem Fahrzeug ein positiver Wert für TTC entsteht. Daraufhin wurde, wie im vorigen Kapitel, Trainingsdatensätze und Validierungsdatensätze von jeweils zwei Dateien gebildet. Die erste Datei umfasst alle Parameter, außer die Klassifikation. Demnach hat diese Datei die folgende Form:

| Bezeichnung | Datentyp | Bedeutung |
|---|---|---|
| Distance | Float | Der Abstand zwischen zwei Verkehrsteilnehmer in Meter. |
| Relative_velocity | Float | Die Relativgeschwindigkeit zwischen zwei Verkehrsteilnehmern in km/h. |

Tabelle 6: Datenform der Trainingsdaten für die Klassifikation von Risikoklassen

Der andere Datensatz enthält die zugehörigen Cluster für die vorherigen Daten und hat demnach die nachfolgende Form.

| Bezeichnung | Datentyp | Bedeutung |
|---|---|---|
| Cluster | Integer | Das zugehörige Cluster. Dieses Feld wird in Kapitel X.X befüllt. |

Tabelle 7: Datenform der wahren Cluster für die Klassifikation von Risikoklassen

## 5.6 Weiterverarbeitung für die Geschwindigkeitsprognose

In Kapitel X.X wird versucht mithilfe von gewöhnlichen neuronalen Netzen eine Geschwindigkeitsprognose erstellen zu können. Demnach wird für das Training dieses Netzes ebenfalls eine Datenverarbeitung benötigt. Folgende Form haben die Daten hierbei erhalten.

| Bezeichnung | Datentyp | Bedeutung |
|---|---|---|
| Verkehrsteilnehmerklasse | Integer | Die Klasse des Verkehrsteilnehmers für diesen Datenpunkt<br>0 = Motorisiertes Fahrzeug<br>1 = Fußgänger<br>2 = Fahrradfahrer |
| Modus | Integer | Eine Kennzahl, die wieder- |

| | | |
|---|---|---|
| | | gibt um was für Aktion es handelt.<br>0 = Langsames Beschleunigen<br>1 = Normales Beschleunigen<br>2 = Schnelles Beschleunigen<br>3 = Kleine Bremsung<br>4 = Normale Bremsung<br>5 = Starke Bremsung<br>Dieses Feld wird allerdings erst in Kapitel X.X befüllt |
| Velocity 1 | Float | Die erste Geschwindigkeit eines Verkehrsteilnehmers |

Tabelle 8: Datenform der Trainingsdaten für die Geschwindigkeitsprognose

Zudem wurden die wahren nächsten Geschwindigkeiten in anderen Dateien gespeichert. Diese haben die folgende Form

| Bezeichnung | Datentyp | Beschreibung |
|---|---|---|
| Velocity 2 | Float | Die Geschwindigkeit die direkt nach Velocity 1 folgt. |

Tabelle 9: Datenform der wahren Geschwindigkeiten für die Geschwindigkeitsprognose

Der Code zur Weiterverarbeitung in diese Form lässt sich in der Projektabgabe finden. Hierbei wurden die Gesamtdaten wieder in einen Trainingsdatensatz und ein Validierungsdatensatz aufgeteilt. Diese werden in der Entwicklung und in der Auswertung gebrauch finden und wurden mit dem jeweiligen Präfix gespeichert.

# 6 Entwicklung Reinforcement Learning

In dieser Arbeit soll die maschinelle Methode des Reinforcement Learning zum Einsatz kommen, um zu validieren, ob diese Technologie sich für die Zielsetzung eignet oder nicht. Demnach werden in diesem Kapitel die notwendigen Voraussetzungen hierfür entwickelt. Vor allem die Entwicklung eines Klassifikationsagenten und einer Umgebung stehen hierbei im Vordergrund. In der Entwicklung wird parallel dazu noch darauf geachtet, das Ganze möglichst generisch zu entwickeln. Das bedeutet, dass das fertige Resultat als eine Art Vorlage dienen kann und für spätere Zwecke nur noch eine Parameterübergabe erfolgen muss, um ein neues Modell zu trainieren.

## 6.1 Anforderungsfestlegung Agenten

Beim Agenten soll es sich um ein Deep Q-Learning Netz handeln

Wie im Grundlagenkapitel X.X festgelegt, eignen sich Deep Q-Learning Netze dafür, um einen Agenten erstellen zu können. Solch ein Netz sollte für den Agenten zum Einsatz kommen.

Kapazität zur Übernahme von weiterverarbeiteten Positionsdaten als Status

Der Reinforcement Learning Agent muss in der Lage sein, die Positionsdaten als aktuellen Status zu erfassen und zu verarbeiten. Diese Daten dienen als Grundlage für die weitere Entscheidungsfindung des Agenten.

Weiterverarbeitung der Eingabe

Der Agent muss in der Lage sein, die empfangenen Positionsdaten in versteckten Schichten weiterzuverarbeiten, um eine bessere Verarbeitung und Klassifikation zu ermöglichen.

Erzeugung von Ausgabe in Form von erwarteter Belohnung für jede Klasse:

Der Agent soll für jede Klasse eine erwartete Belohnung ausgeben können, die als Ausgabe seiner Entscheidungsfindung dient.

Fähigkeit zum Lernen aus wirklicher Belohnung für stetige Verbesserung der Klassifikation

Der Agent muss in der Lage sein, aus der wirklichen Belohnung zu lernen und dadurch die Qualität der Klassifikation kontinuierlich zu verbessern. So kann er im Laufe der Zeit eine bessere Entscheidungsfindung ermöglichen.

Kontinuierliche Erforschung der Umgebung

Neben der Fähigkeit zu lernen und sich stetig zu verbessern, sollte der Agent auch in der Lage sein etwas Neues auszuprobieren und zu entdecken. Dies soll dem Agenten dabei flexibler zu werden, indem neben der Verbesserung des bisherigen auch Neues entdeckt werden kann.

## 6.2 Anforderungsfestlegung Umgebung

Übernahme einer Aktion des Agenten

Die Reinforcement Learning Umgebung muss in der Lage sein, eine Aktion des Agenten anzunehmen, um Belohnungen kalkulieren zu können.

Auswertung der Aktion und Erstellung einer Belohnung

Die Umgebung muss in der Lage sein, die Aktion des Agenten auszuwerten und eine Belohnung für die ausgeführte Aktion zu erstellen. Die Belohnung dient als Feedback für den Agenten, um dessen Lernfortschritt zu fördern.

Übermittlung der Belohnung an den Agenten

Die Umgebung muss in der Lage sein, die erstellte Belohnung an den Agenten weiterzugeben, damit dieser aus der Belohnung lernen kann.

Aktualisierung aller notwendigen Parameter nach Aktionsauswertung

Die Umgebung muss in der Lage sein, alle notwendigen Parameter nach der Auswertung der Aktion zu aktualisieren, um den Agenten mit neuen Daten versorgen zu können.

Rückgabe aller Faktoren als ein Paket

Die Umgebung sollte all diese Faktoren, einschließlich der Belohnung, des neuen Status und der Positionsdaten, als ein Paket zurückgeben, um eine einfache Übertragung dieser Informationen an den Agenten zu gewährleisten. Dies verbessert die Effizienz des Reinforcement Learning Prozesses und fördert den Lernfortschritt des Agenten.

## 6.3 Konzeption des Agenten

Anhand der in Kapitel X.X festgelegten Anforderungen, können in diesem Kapitel die Aktivitäten von Agenten und Umgebung detaillierter geplant werden, sodass sich am Ende dieses Kapitels ein konkretes Modell ergibt, welches zur Entwicklung verwendet werden kann.

Beim Agenten soll es sich um ein Deep Q-Learning Netz handeln:

Beim Agenten handelt es sich um ein Deep Q-Learning Netz. Damit diese Anforderung allerdings konzipiert werden kann, wird um das neuronale Netz eine Klasse gebaut. Diese Klasse wird im Laufe dieses Kapitels die dementsprechenden Funktionen erhalten, um alle Anforderungen des neuronalen Netzes zu erfüllen.

Im ersten Schritt des Konzeptes erhält diese Klasse erstmals die Funktionen __init__ und learn().

Die Attribute der Klasse die in der init()-Funktion initialisiert werden lauten dabei:

action_space – Anzahl der Aktionen/Ausgabeneuronen, die das Modell des Agenten ausführen kann.

obervation_space – Anzahl Eingabeparameter die das Modell des Agenten entgegen nehmen soll.

discount_factor – Der Faktor der die Gewichtung zukünftiger Belohnungen festlegt.

model – Das neuronale Netz des Agenten.

Kapazität zur Übernahme von weiterverarbeiteten Positionsdaten als Status

Im vorherigen Schritt wurde eine Agentenklasse festgelegt, welche konzipiert wird, um die Anforderungen des Neuronalen Netzes zu erfüllen. Damit die weitverarbeiteten Positionsdaten an die Eingabeschicht des Neuronalen Netzes weitergegeben werden können, wird eine learn()- und act()-Funktion implementiert. Diese Funktionen nehmen dabei die Rückgaben der Umgebung als Parameter entgegen. Die Parameter werden daraufhin in den Funktionen verwendet, um die Eingabeschicht des neuronalen Netzes stetig mit neuen Positionsdaten als Status versorgen zu können.

Weiterverarbeitung der Eingabe:

Die Weiterverarbeitung der Eingabe wird über die versteckten Schichten des neuronalen Netzes realisiert. Dementsprechend hat man in Bezug auf diese Anforderungen in der Konzeption relativ wenig Einfluss, da man im Voraus schlichtweg nicht weiß wie sich die Gewichtungen des Netzes entwickeln werden. Ein wenig Einfluss auf die Weiterverarbeitung kann man allerdings im Bereich der Hyperparameter-Festlegung ausüben, indem man Parameter wie Anzahl von Neuronen oder auch die Learning Rate festlegt.

Erzeugung von Ausgabe in Form von erwarteter Belohnung für jede Klasse:

Die Ausgabe wird in der letzten Schicht des Neuronalen Netzes, auch Ausgabeschicht genannt, erzeugt. Diese Schicht wird dabei so viele Ausgaben wie Klassifikationsmöglichkeiten erzeugen. Die Ausgabe wird für jede Klassifikationsmöglichkeit eine zu erwartende Belohnung für jede Klasse sein. Das bedeutet jedes Ausgabeneuron wird zu einem gewissen Anteil aktiviert, es wird allerdings nur das Neuron mit der höchsten erwarteten Belohnung gewählt und der Umgebung als Aktion zurückgegeben.

Dementsprechend wird sich die Agentenklasse im ersten Schritt eine Ausgabe über das Neuronale Netz erzwingen, hierfür wurde bereits eine act()-Funktion implementiert. Von der Ausgabe wird daraufhin das Neuron mit der höchsten zu erwartenden Belohnung ausgesucht und die Klassifizierung dieses Neurons an die Umgebung weitergeben.

Fähigkeit zum Lernen aus wirklicher Belohnung für stetige Verbesserung der Klassifikation:

Damit neuronale Netze lernen können wird ein Verfahren namens Backpropagation angewendet. Dies wurde bereits in Kapitel Kapitel X.X vorgestellt. Der Agent erhält eine Belohnung und vergleicht die tatsächliche Belohnung mit der optimalen erwarteten Belohnung, errechnet sich hieraus einen Loss und führt mithilfe dieses Losses die Backpropagation aus.

Für die Konzeption bedeutet dies, dass hier die bereits erwähnte Learn()-Funktion hergenommen wird. Diese Funktion nimmt dabei die Ausgabeparameter der Umgebung entgegen.

Kontinuierliche Erforschung der Umgebung:

Hierfür soll die Epsilon Greedy Strategie angewendet werden, wie sie im Grundlagenkapitel X.X vorgestellt wurde. Damit Epsilon Greedy angewendet werden kann, wird die Agentenklasse um diese folgenden Attribute erweitert.

epsilon – Der Vergleichsfaktor, der die Wahrscheinlichkeit einer Zufallsaktion bestimmt

epsilon_min – Der minimale Wert für den Vergleichsfaktor, wenn Epsilon kleiner gleich diesem Wert ist, soll Epsilon nicht mehr verringert werden

epsilon_decay – Der Verminderungsfaktor, der festlegt um wieviel Prozent Epsilon jeweils verringert wird

Parallel dazu wird die act()-Funktion erweitert werden müssen. In dieser Funktion soll zusätzlich überprüft werden, ob eine Zufallszahl kleiner als der momentane Wert für Epsilon ist. Ist dies der Fall, so soll der Agent im nächsten Schritt eine zufällige Aktion auswählen. Sollte dies allerdings nicht der Fall sein, so soll der Agent versuchen mit der nächsten Aktion die maximale Belohnung zu erzielen.

### 6.3.1 Zusammenfassung der resultierenden Klasse

Um das bisher geschriebene final zusammen zu fassen und um ein übersichtliches Bild der resultierenden Klasse zu erstellen, wurde ein abschließendes UML-Diagramm der Klasse erstellt.

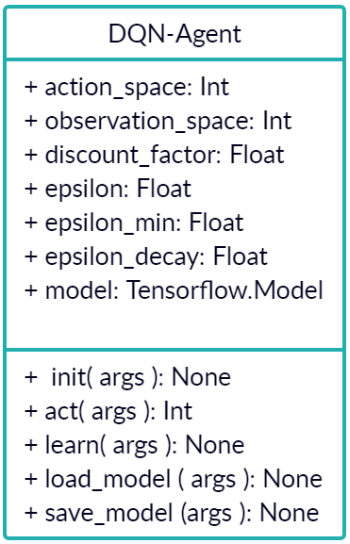


Abbildung 6.1: Die Agentenklasse visualisiert

Die erstellte Abbildung kann somit verwendet werden, um sich ein stetiges Bild über den Aufbau im Überblick zu behalten und der geschrieben Text kann demnach für die detaillierte Konzeption verwendet werden.

## 6.4 Konzeption der Umgebung

Die Umgebung hat im Vergleich zum Agenten andere Aufgaben und wird dementsprechend auch anders konzipiert. Die Umgebung wird, ähnlich wie der Agent, erstmals als eine Klasse konzipiert. Allerdings mit dem Unterschied, dass in dieser Klasse kein neuronales Netz im Mittelpunkt steht. Am Anfang des Konzeptes wird für diese Klasse eine __init()__ Funktion festlegt. In dieser initialen Funktion, welche bei Erstellung der Klasse ausgeführt wird, sollen die ersten Attribute dieser Klasse einen initialen Wert erhalten.

Die ersten Attribute die diese Klasse erhält lauten demnach:

current_count – Eine Zählervariable, die angibt in welcher Position man sich im Datensatz befindet

data – Die Daten in der Form eines Arrays

true_classes – Die wahren Klassifikationen, welche zum Vergleich und zur Berechnung der Belohnung hergenommen werden

Übernahme einer Aktion des Agents

Die Agentenklasse, welche im vorigen Kapitel vorgestellt wurde, gibt über dessen act() eine Klassifikation zurück, welche als Aktion interpretiert werden.

Die Übernahme wird in der Umgebungsklasse über eine Funktion realisiert, die als step() bezeichnet wird. Der Funktion sollen dabei als Parameter die Aktion des Agenten übergeben werden.

Auswertung der Aktion und Erstellung einer Belohnung

Im vorherigen Schritt wurde die step() Funktion vorgestellt, welche einen Parameter die Aktion des Agenten entgegen nimmt. In dieser Funktion wird ebenfalls die Auswertung des erhaltenen Parameters stattfinden und im Anschluss eine Belohnung festgelegt.

Damit dies geschehen kann, muss ein Vergleich zwischen der wirklichen Klasse und der von dem Agenten ausgegebenen Klasse stattfinden. Hierfür wird innerhalb der Umgebung das Attribut das jeweilige Element aus true_classes hergenommen. Daraufhin wird dann der Wert des Parameters Action mit dem Wert des Elements verglichen und es kann daraufhin eine Belohnung erstellt und zurückgegeben werden.

Übermittlung der Belohnung an den Agenten

Die Belohnung wird am Ende der step() Funktion zurückgegeben. Demnach wird für diese Anforderung an der Konzeption keine Änderung vorgenommen.

Änderung des Status nach der Aktionsauswertung

Die Änderung des Status findet auf zwei Ebenen statt. Erstmals wird ein initialer Status gesetzt in der reset()-Funktion der Klasse. In der step()-Funktion findet dann bei Statuswechsel ebenfalls ein Übergang in den neuen Status statt. Dieser spiegelt sich wieder in dem der eingelesene Datensatz zum nächsten Datensatz wechselt.

Aktualisierung aller notwendigen Parameter nach Aktionsauswertung:

Die Aktualisierung aller notwendigen Parameter wird ebenfalls in der step()-Funktion entwickelt. Der Grund hierfür ist, dass Python mit seiner Syntax eine übersichtliche und kurze Implementierung ermöglichen kann. Die nachfolgenden Parameter müssen nach der Auswertung der aktuellen Aktion aktualisiert werden:

- Status – Muss einen neuen Wert annehmen.

- current_count – Muss inkrementiert werden.

An der Modellierung der Umgebung müssen dadurch erstmals keine Änderungen vorgenommen werden.

Rückgabe aller Faktoren als ein Paket:

Ein Punkt der in der step() Funktion dieser Klasse noch nicht behandelt wurde, ist was die Funktion letztendlich zurückgeben soll. Die folgenden Parameter werden in dieser Funktion zurückgegeben:

state – Der neue Status nach Kalkulation der Belohnung

reward – Die errechnete Belohnung für die Aktion des Agenten

Die Reihenfolge, in der die Parameter zurückgegeben werden, muss dabei der Reihenfolge dieser Aufzählung entsprechen, da dies bei der Annahme der Agentenklasse ebenfalls eine Rolle spielen wird.

An der Modellierung der Umgebung müssen hierdurch ebenfalls keine Änderungen vorgenommen werden.

### 6.4.1 Zusammenfassung der resultieren Klasse

Ähnlich wie in Kapitel 4.3.1 wird nun zum Abschluss der Konzeption die Klasse als UML-Klasse dargestellt. Dies dient einerseits, um das Geschriebene nochmals Revue passieren zu lassen und andererseits um nochmals ein Gesamtbild der Klasse zeigen zu können.

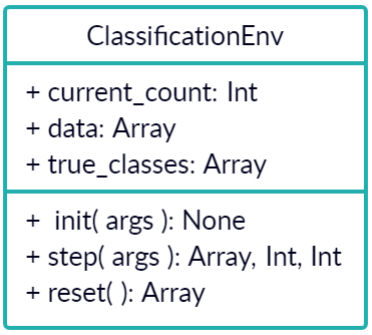


Abbildung 6.2: Die Umgebungsklasse visualisiert

Mit der Darstellung der UML-Klasse hat man somit einen Gesamtüberblick über die Klasse und hat dabei die Möglichkeit mithilfe des Geschriebenen die Details hierzu zu erarbeiten. Dies wird den nächsten Schritt, die Implementierung, unterstützen.

## 6.5 Implementierung der Klasse Agent

Die Funktionen der Agentenklasse sind ebenfalls bereits in Kapitel X.X definiert worden. Demnach wird in diesem Kapitel, ähnlich im vorigen Kapitel, ein tieferer Blick in die Umsetzung des Codes gezogen.

### 6.5.1 init()

```
def __init__(self, action_space, observation_space, num_of_layers,
num_of_neurons_per_layer, model=None):
    self.discount_factor = 0.01
    self.action_space = action_space
    self.epsilon = 1  # exploration rate
    self.epsilon_min = 0.015
    self.epsilon_decay = 0.9999

    # define the neural network
    if model == None:
        self.model = tf.keras.Sequential()


self.model.add(tf.keras.layers.Dense(num_of_neurons_per_layer, in-
put_shape=(1, ), activation='relu'))
        if num_of_layers > 1:
            for i in range(1, num_of_layers):

self.model.add(tf.keras.layers.Dense(num_of_neurons_per_layer, acti-
vation='relu'))
```

```
        self.model.add(tf.keras.layers.Dense(action_space, activa-
tion='linear'))
        self.model.compile(loss='mse', optimiz-
er=tf.keras.optimizers.Adam(learning_rate=0.001))
    else:
        self.model = model
```

Code 1: init()-Funktion der Agentenklasse

Die init()-Funktion dient zur Initialisierung der Klasse sobald diese erstellt wird. Sie erhält folgende Parameter als Übergabe:

action_space = Alle Mögliche Aktionen die der Agent, bzw. dessen Netz ausführen kann.

observation_space = Die Form in der ein Status dem Agenten übergeben wird.

num_of_layers = Die Anzahl der verborgenen Schichten.

num_of_neurons_per_layer = Die Anzahl der Neuronen in den verborgenen Schichten.

model = eine Variable, die gegeben werden kann, falls ein bestehendes Neuronales Netz geladen wurde und weitertrainiert, werden soll.

Am Anfang dieser Funktion werden die Variablen in der Übergabe Klasseninternen Attributen zugewiesen.

Daraufhin folgt die Initialisierung von drei Attributen epsilon, epsilon_min und epsilon_decay. Diese drei Attribute sind für die Epsilon Greedy Strategie wichtig, diese wurde in Kapitel X.X vorgestellt und ist wichtig für die Erforschung neuer Vorgänge.

Im Anschluss wird in der init()-Funktion das neuronale Netz definiert. Die Anzahl der versteckten Schichten und der Neuronen in diesen versteckten Schichten wird beispielsweise in Kapitel X.X mithilfe des GridSearch Algorithmus ermittelt und wird demnach in der Parameterübergabe mitgegeben. Hierbei ist wichtig zu verstehen, dass die erste angegebene Schicht bereits eine versteckte Schicht darstellt. Die Eingabeschicht wird lediglich über den Parameter „input_shape“ definiert. Zur Aktivierungsfunktion „relu“ kann nochmals in Kapitel X.X die Funktionsweise nachgesehen werden.

### 6.5.2 act()

```
def act(self, state):

    # Wähle die beste Aktion anhand des Neuronalen Netzes und der
```

```
aktuellen Q-Werte
    if np.random.rand() <= self.epsilon:
        return random.randrange(self.action_space)

    q_values = self.model.predict(state, verbose=0)
    action = np.argmax(q_values[0])
    return action
```

Code 2: act()-Funktion der Agentenklasse

Die Aufgabe der act-Funktion() ist es festzulegen ob der Agent in seinem nächsten Schritt eine zufällige Aktion durchführen soll oder nicht. Dabei entscheidet der Zufall, ob es zu einer zufälligen Aktion kommt. Mithilfe der Library Random, wird eine Zufallszahl festgelegt. Ist die Zufallszahl kleiner als das Attribut epsilon, dann wird eine zufällige Aktion zurückgegeben. Andernfalls wird die Aktion mit der zu höchsten versprechenden Belohnung ausgewählt. Das ganze Verfahren orientiert sich an der Epsilon Greedy Strategie, welche in Kapitel X.X erklärt wurde. Der Sinn dahinter ist, dass der Agent in der Lage ist komplett neue Aktionen durchzuführen, damit diese potenziellen anderen Wege entdecken kann und nicht in seinem gewohnten Verhalten festgefahren ist.

### 6.5.3 learn()

```
def learn(self, state, action, reward, next_state, done):

    # Berechne die Ziel-Q-Werte
    q_values_next = self.model.predict(next_state, verbose=0)
    q_target = reward + self.discount_factor *
np.amax(q_values_next[0])
    if done:
      q_target = reward

    # Berechne die aktuellen Q-Werte für den aktuellen Zustand
    q_values = self.model.predict(state, verbose=0)

    # Aktualisiere die Q-Werte für die aktuelle Aktion
    q_values[0][action] = q_target

    # Trainiere das Neuronale Netz mit den aktualisierten Q-Werten
    self.model.fit(state, q_values, verbose=0)

    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

Code 3: learn()-Funktion der Agentenklasse

Die learn()-Funktion ist für das eigentliche lernen des Neuronalen Netzes zuständig. Die Funktion erhält folgende Parameter als Übergabe

State = Der aktuelle Status

Action = Die Aktion die der Agent für den aktuellen Status getätigt hat

Reward = Die Belohnung die der Agent für aktuellen Status und Aktion erhält

Next_state = Der nächste Status

Done = Ein Abschlussparameter mit dem man das Ende eines Datennetzes angeben kann

Im ersten Teil des Codes wird mit dem nächsten Status eine Prognose gemacht. Diese Prognose fließt dann in die Berechnung der optimalen Belohnung ein. Betrachtet man die Formel, so orientiert diese sich an der Bellman Funktion, wie sie im Grundlagenkapitel X.X vorgestellt wurde. Der Parameter discount_factor bestimmt dabei, wie sehr die zukünftige Prognose in die Berechnung mit einfließen soll.

Nachdem die optimale Belohnung berechnet wurde, werden für den aktuellen Status nochmals die Q-Values, also die Belohnungswerte ermittelt. Daraufhin wird das Neuron, welches für die Aktion des aktuellen Status den höchsten Belohnungswert versprochen hat, hergenommen und dessen Belohnungswert wird dem optimalen Belohnungswert ersetzt.

Nun kann mithilfe der fit()-Funktion die Backpropagation stattfinden und alle Gewichte der hinter stehenden Neuronen werden so verlagert, sodass diese der Ausgabe gerecht werden können. Was die Backpropagation ist und wie dieses Verfahren funktioniert kann nochmals in Kapitel X.X nachgelesen werden.

Zum Abschluss der Funktion wird, der der Wert für die Epsilon Greedy Strategie mit dem definiertem Decay reduziert, dies wird so lange gemacht bis der Wert für Epsilon ein definiertes Minimum erreicht hat.

## 6.6 Implementierung der Klasse Umgebung

Die Funktionen der Klasse Umgebung sind in Kapitel X.X festgelegt worden und demnach schon bekannt. In diesem Kapitel wird demnach ein etwas tieferer Blick in die wichtigsten Codestellen geworfen, um ein tieferes Verständnis zur Klasse Umgebung aufbauen zu können.

### 6.6.1 init()

```
def __init__(self, data, true_classes):

    # current count in Dataset
    self.current_count = 0
    # init data
    self.data = data
    # init true_classes
    self.true_classes = true_classes
    self.current_true_class = None
```

Code 4: init()-Funktion der Umgebungsklasse

Die init()-Funktion ist für die Initiierung der Umgebung zuständig. Die Funktion erhält die folgenden Parameter:

Data = Die verarbeiteten Daten mit richtigen Datentypen

True_classes = Die zugehörigen Klassen zu den Daten

Wirft man einen Blick in den Code, so sieht man das in diesem Fall hauptsächlich Zuweisungen von Eingabeparameter auf Klassenattribute erfolgen. Die Zuweisungen spielen allerdings im späteren Verlauf eine große Rolle, da diese in der Klasse selbst immer wieder abgerufen werden können, um einen Agenten stetig mit neuen Status aus den Daten versorgen zu können.

### 6.6.2 step()

```
def step(self, action):

    # calculate the reward based on the classification accuracy
    if action == self.current_true_class:
        reward = 1
    else:
        reward = -1

    self.current_count += 1
    if self.current_count == len(self.data):
        self.current_count = 0

    # define the next state as the current state
    current_data_point = self.data[self.current_count]

    self.state = current_data_point

    self.current_true_class = self.true_classes[self.current_count]
```

```
    # define a flag to indicate if the episode has finished
    done = False

    return self.state, reward, done, {}
```

Code 5: step()-Funktion der Umgebungsklasse

Die Funktion step() ist die Kernfunktion der Umgebungsklasse. Sie erhält als Eingabeparameter die Aktion des Agenten für den aktuellen Status. Demnach wird in der step()-Funktion zuerst die Belohnung für den Agenten ermittelt, indem Aktion (oder auch Klassifikation) des Agenten mit der wahren Klasse verglichen werden. Wichtig ist hierbei zu erwähnen, dass diese Belohnungsfunktion nur für diese Masterarbeit verwendet, werden sollten und nicht für alle möglichen Anwendungsszenarien funktionieren. Verschiedene Szenarien brauchen dementsprechend verschiedene Belohnungsfunktionen.

Nach Festlegung der Belohnung wird im Anschluss der neue Status und die neue wahre Klasse festgelegt. Dabei wird noch beachtet, dass das Datenarray Von vorne wiedergegeben wird, falls der Letzte Datensatz im Array erreicht wurde.

### 6.6.3 reset()

```
def reset(self):

    self.current_count = 0

    # define the next state as the current state
    current_data_point = self.data[self.current_count]

    self.state = current_data_point

    self.current_true_class = self.true_classes[self.current_count]

    return self.state
```

Code 6: reset()-Funktion der Umgebungsklasse

Die reset-Funktion() wird verwendet, um die Umgebung in ihren initialen Status zurückversetzen zu können. In Bezug auf diese Umgebung wird der aktuelle Datenpunkt auf den ersten Punkt im Datensatz zurückversetzt und die aktuelle wahre Klasse wird demnach ebenfalls auf die Kasse des ersten Punktes im Datensatz zurückgesetzt.

# 7 Entwicklung normales Feed Forward Netz

Neben der Klassifikation mithilfe von Reinforcement Learning werden in dieser Masterarbeit auch gewöhnliche Feed Forward Netze zum Einsatz kommen. Dies dient primär dazu, um einen stetigen Vergleich zwischen Reinforcement Learning und gewöhnlichen neuronalen Netzen ziehen zu können. Allerdings gibt es auch einen weiteren Grund für den Einsatz von normalen neuronalen Netzen, nämlich dass sich mit der Entwickelten Vorlage aus Kapitel X.X nicht alle Aufgaben in dieser Masterarbeit lösen lassen können. Demnach wird in diesem Kapitel ebenfalls eine Vorlage entwickelt, welche lediglich eine Parameterübergabe benötigt, um daraufhin ein gewöhnliches neuronales Netz entwickeln zu können. Hierbei ist zu erwähnen, dass die Komplexität der Entwicklung nicht an die von Reinforcement Learning rankommt. Das liegt daran, dass in diesem Falle keine Klassenstrukturen entwickelt werden müssen, welche im stetigen Austausch miteinander sind. So kann in diesem Falle direkt mit der Implementierung begonnen werden.

## 7.1 Implementierung

```
def train_model_and_save(number_of_output_neurons, number_of_layers,
number_of_neurons, number_of_input_neurons,
                         train_data, train_labels, val_data,
                         val_labels, epochs, model_file_name):

    # Erstellen des Modells
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(number_of_neurons, in-
put_shape=(number_of_input_neurons,), activation='relu'))

    if number_of_layers > 1:
        for i in range(1, number_of_layers):
            model.add(tf.keras.layers.Dense(number_of_neurons, acti-
vation='relu'))

    model.add(tf.keras.layers.Dense(number_of_output_neurons, acti-
vation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimiz-
er='adam', metrics=['accuracy'])

    print(len(train_data), len(train_labels))
    # Training des Modells mit Validierungsdaten
    history = model.fit(train_data, train_labels, epochs=epochs,
validation_data=(val_data, val_labels))

    # Speichern des Modells
    model.save(model_file_name)
```

Code 7: Implementierung des Codes für gewöhnliche neuronale Netze

Dies ist die entwickelte Funktion, die verwendet werden kann, um lediglich über die Übergabe von Parametern ein gewöhnliches neuronales Netz entwickeln zu können. Demnach stehen hierbei auch die Parameter im Vordergrund. Folgende Parameter werden der Funktion bei Aufruf übergeben:

Number_of_output_neurons = Anzahl der Ausgabeneuronen, welche als die Anzahl der Klassen interpretiert werden kann.

Number_of_layers = Anzahl der versteckten Schichten des neuronalen Netzes.

Number_of_neurons = Anzahl der Neuronen in den versteckten Schichten.

Number_of_input_neurons = Anzahl der Neuronen in der Eingabeschicht.

Train_data = Die Daten mit denen das neuronale Netz trainiert werden soll als Array.

Train_labels = Die korrekten Outputs zu den Trainingsdaten.

Val_data = Die Daten mit denen nach Ende einer Episode eine Validierung durchgeführt werden soll.

Val_labels = Die korrekten Outputs zu dem Validierungsdatensatz.

Epochs = Anzahl der Epochen für die trainiert werden soll.

Model_file_name = Der Dateiname mit der das Modell gespeichert werden soll.

Der Code geht dabei so vor, dass erstmals das Modell selbst erstellt wird. Mithilfe der ersten drei Parameter und for-Schleifen, lässt sich dies generisch entwickeln. Nach dem Setzen der Hyperparameter kann daraufhin auch schon das eigentliche Training stattfinden. Mit der Library Tensorflow wird das über die fit()-Funktion realisiert. Zum Abschluss des Trainings wird das Modell daraufhin gespeichert und kann für spätere Zwecke wieder geladen werden. In der Architektur ist dabei noch zu beachten, dass nur ein Ausgabeneuron gegeben wird. Bei einer Klassifikation wird das Ausgabeneuron die Klasse als Zahl wiedergeben und bei einer Schätzung wird das Ausgabeneuron ebenfalls eine Zahl wiedergeben.

# 8 Verkehrsteilnehmerklassifikation

## 8.1 Festlegung der optimalen Hyperparameter

Im ersten Schritt der Entwicklung werden erstmals die optimalen Hyperparameter für das neuronale Netz herausgefunden. Dabei wird der GridSearch-Algorithmus verwendet, welcher in Kapitel X.X vorgestellt wurde. Der Algorithmus beruht darauf einen Wertebereich auszuwählen und für jeden dieser Wertebereiche zu testen. Das Ergebnis mit den besten Testergebnissen kann daraufhin als Maßstab für optimale Hyperparameter hergenommen werden.

Dabei wurde der Wertebereich [4, 8, 16, 32, 64, 128, 256] festgelegt und der GridSearch Algorithmus wurde einmal mit einer versteckten und einmal mit zwei versteckten Schichten getestet. Im Anschluss wurden die Ergebnisse daraufhin mithilfe der Library Matplotlib visualisiert, sodass am Ende das nachfolgende Bild präsentiert werden kann.

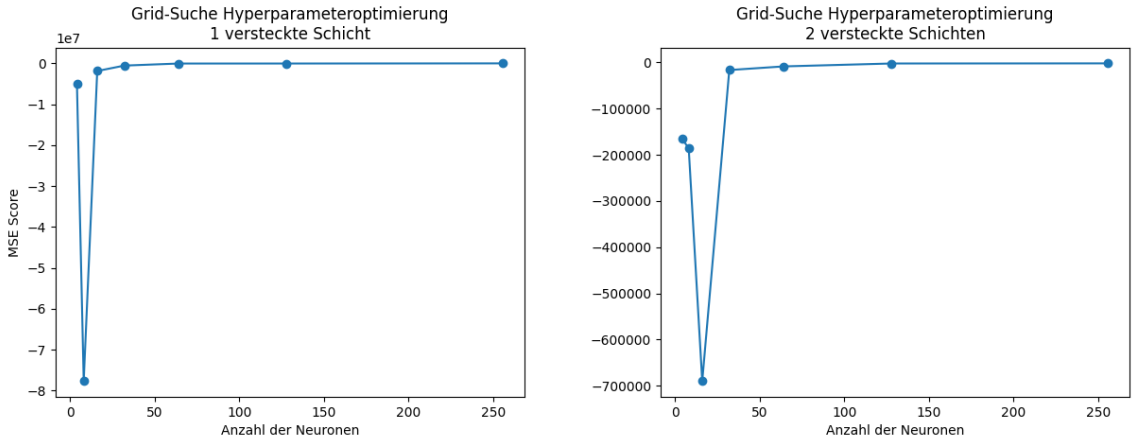


Abbildung 8.1: Grid search Vekehrsteilnehmerklassifikation

Die Tests haben dabei für eine versteckte Schicht ergeben, dass 16 Neuronen, das beste Ergebnis aus den Daten herausholen konnten. Ab 16 Neuronen konnte ein Mean Squared Error von nahezu 0 erreicht werden.

Bei zwei versteckten Schichten liegt die optimale Anzahl ebenfalls bei 32 Neuronen. Ab 32 Neuronen hat man ebenfalls einen Mean Squared Error, welcher nahezu gegen Null tendiert.

Somit sind die besten Hyperparameter bei einer versteckten Schicht und 16 Neuronen in dieser versteckten Schicht. Somit wird in den nächsten Schritten eine Architektur entwickelt, welche mit den nachfolgenden Abbildungen verglichen werden kann.

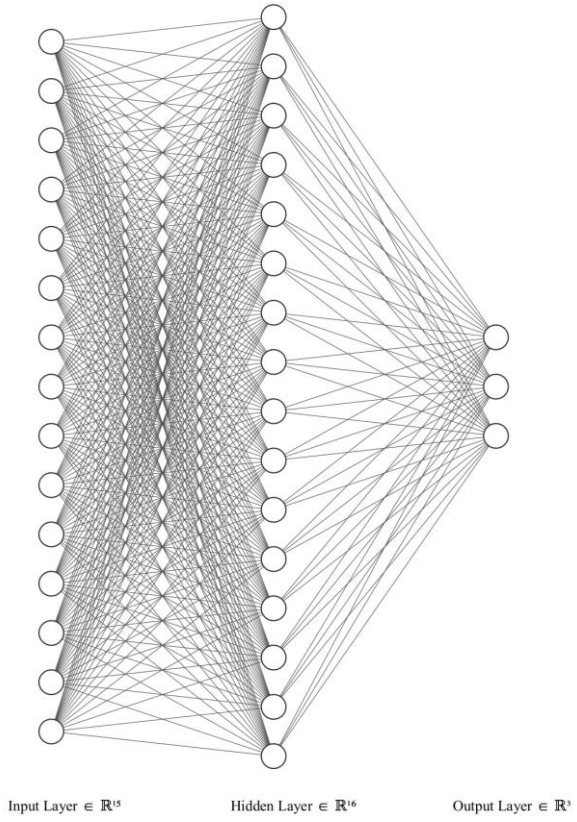


Abbildung 8.2: Architektur neuronales Netz der Verkehrsteilnehmerklassifikation

## 8.2 Parameterübergabe Reinforcement Learning

Wie in Kapitel X.X bereits erwähnt, wurde der Code zur Entwicklung eines Reinforcement Learning Models so generisch geschrieben, sodass in diesem Kapitel lediglich eine Parameterübergabe erfolgen muss, um ein Modell entwickeln zu können. Demnach werden folgende Parameter an den Agenten übergeben.

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Action_space | 3 | Action Space beschreibt wieviele Ausgabeneuronen/Aktionen der Agent ausführen kann. In den Daten existieren 3 Klassen, somit 3 Klassifikationsmöglichkeiten und 3 Ausgabeneuronen. |
| Observation_space | 15 | Observation Space beschreibt die Anzahl |

| | | |
|---|---|---|
| | | der Input Neuronen. Betrachtet man die Datensätze, so werden 15 verschiedene Daten übergeben, also 15 Neuronen |
| Num_of_layers | 1 | Anzahl der verborgenen Schichten, festgelegt mithilfe von GridSearch |
| Num_of_neurons_per_layer | 16 | Ebenfalls mithilfe von GridSearch festgelegt |

Tabelle 10: Parameterübergabe Verkehrsteilnehmerklassifikation des Agenten mit Reinforcement Learning

Für die Umgebung wird sich die Parameterübergabe nach der folgenden Tabelle richten.

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Data | Weiterverarbeiteten Daten aus Kapitel X.X als Array | Die Bedeutung bzw. die Form der Daten kann im jeweiligen Kapitel nachgesehen werden |
| True_classes | Die weiterverarbeiteten wahren Klassen aus Kapitel X.X als Array | Hier kann ebenfalls die Bedeutung bzw. die Form der Daten im jeweiligen Kapitel nachgesehen werden. |

Tabelle 11: Parameterübergabe Verkehrsteilnehmerklassifikation der Umgebung mit Reinforcement Learning

Das Modell wird so lange trainiert, bis in der Präzision keine deutlichen Verbesserungen mehr vorkommen. Der letztliche Code, der hierbei entsteht, lässt sich in der Projektabgabe nachsehen. Am Ende wird das Modell gespeichert, um es für spätere Zwecke wiederverwenden zu können.

## 8.3 Parameterübergabe Gewöhnliches Feed Forward Netz

Bei der Entwicklung eines gewöhnlichen Feed Forward Netzes wurde ebenfalls in Kapitel X.X darauf geachtet den Code so generisch wie möglich zu gestalten. Dies ermöglicht es in diesem Kapitel lediglich eine Parameterübergabe festzulegen, um daraufhin ein Modell entwickeln zu können. Folgende Parameter werden dementsprechend weitergegeben:

Number_of_layers = Anzahl der versteckten Schichten des neuronalen Netzes.

Number_of_neurons = Anzahl der Neuronen in den versteckten Schichten.

Number_of_input_neurons = Anzahl der Neuronen in der Eingabeschicht.

Train_data = Die Daten mit denen das neuronale Netz trainiert werden soll als Array.

Train_labels = Die korrekten Outputs zu den Trainingsdaten.

Val_data = Die Daten mit denen nach Ende einer Episode eine Validierung durchgeführt werden soll.

Val_labels = Die korrekten Outputs zu dem Validierungsdatensatz.

Epochs = Anzahl der Epochen für die trainiert werden soll.

Model_file_name = Der Dateiname mit der das Modell gespeichert werden soll.

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Number_of_output_neurons | 3 | Anzahl der Ausgabeneuronen, welche mit der Anzahl der Klassifizierungen gleichgestellt werden, kann. Da es drei Klassen gibt, wird es demnach 3 Ausgabeneuronen geben. |
| Number_of_layers | 1 | Eine versteckte Schicht, welche mithilfe von GridSearch erarbeitet wurde. |
| Number_of_neurons | 16 | 16 Neuronen in den versteckten Schichten. Ebenfalls mithilfe von GridSearch erarbeitet. |
| Number_of_input_neurons | 15 | 15 Inputneuronen, welche die weiterverarbeiteten Daten aus Kapitel X.X wider- |

| | | |
|---|---|---|
| | | spiegeln. |
| Train_data | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Val_data | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Val_labels | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Epochs | 10 | Es wird für 10 Epochen trainiert. |
| Model_file_name | „participant_cassification_model.h5” | Der Dateiname unter dem das Modell gespeichert werden soll. |

Tabelle 12: Parameterübergabe der Verkehrsteilnehmerklassifikation für gewöhnliche neuronale Netze

Das Modell wird dabei so lange trainiert bis im Mean Squared Error keine Verbesserung mehr in den Validierungsdaten stattfindet. Dies dient zu Prävention von Overfitting. Zum Abschluss wird das Modell gespeichert, um es für spätere Zwecke weiter verwenden zu können.

# 9 Relevanzklassifikation

Neben der Klassifikation der eigentlichen Verkehrsteilnehmerklasse, soll in dieser Masterarbeit auch eine Klassifikation von Relevanz stattfinden. Dieses Kapitel beschäftigt sich aus diesem Grund rund um das Thema Relevanz und es wird Schritt für erklärt, wie ein Klassifikationssystem für Relevanz entstehen kann.

## 9.1 Bildung von Risikoklassen

Der erste festgelegte Schritt zur Entwicklung von Risikoklassifikation war die Bildung von Risikoklassen mithilfe des TTC-Parameters und mithilfe von Clustering. Damit erfolgreich geclustert werden kann, benötigten die Daten eine gewisse Vorverarbeitung, welche die Kalkulation und Speicherung der TTC-Parameter umfasst. Diese Vorverarbeitung wurde bereits im Datenkapitel X.X beschrieben und die daraus resultierte CSV-Datei wird als Basis für dieses Kapitel genommen.

Als Clusteringverfahren wird der K-Means Algorithmus verwendet, welcher bereits in Kapitel X.X vorgestellt wurde. Eine Einschränkung des K-Means Algorithmus ist, dass man im Voraus angeben muss, wie viele Cluster man bilden möchte. Als Lösung hierfür wird oftmals die sog. „Ellenbogenmethode“ verwendet. Hierbei wählt man den Punkt, ab dem kein größerer Einbruch der quadratischen Abstände mehr stattfindet.

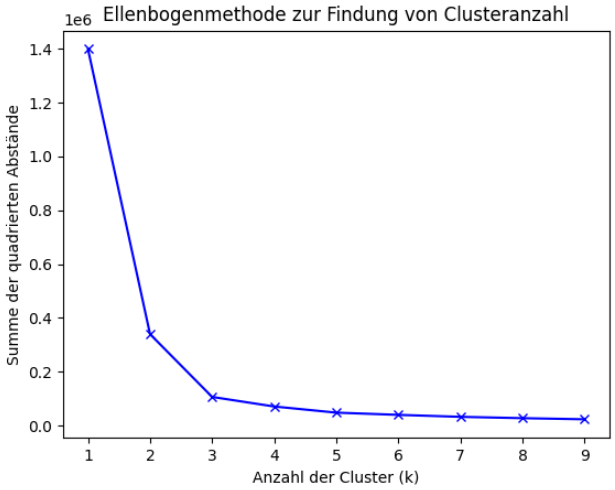


Abbildung 9.1: Ellenbogenmethode für die Risikoklassen

Das Ergebnis der Ellenbogenmethode zeigt in diesem Fall, dass ab vier Clustern kein größerer Einbruch mehr stattfindet und man die Clusteranzahl demnach bei vier wählen sollte.

Führt man den Clustering Algorithmus nun mit der Clusterzahl vier aus, so ergibt sich daraus folgendes Bild.

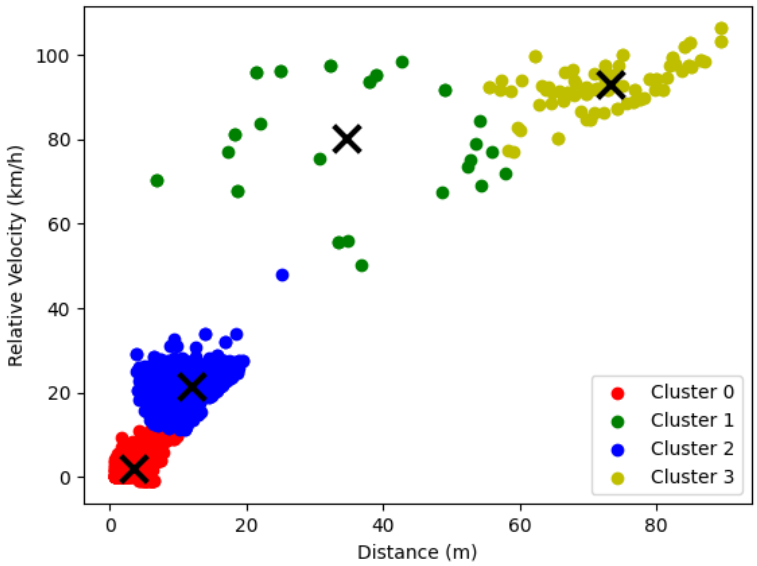


Abbildung 9.2: Ergebnis Clustering Risikoklassen

Eine Analyse der Cluster hat dabei folgende Interpretation ergeben können.

Cluster 0: Niedriger Abstand und niedrige relative Geschwindigkeiten. Häufig zu finden in Städten und an Ampeln. Vergleichsweise ungefährlich, allerdings relevant aufgrund der geringen Distanz zwischen einander.

Cluster 1: Das wahrscheinlich gefährlichste Cluster von allen. Zu hohe Geschwindigkeiten bei zu wenig Abstand sorgen bei diesem Cluster für ein besonders hohes Risiko. Sollte eine Partei bremsen, könnte es knapp werden. Demnach wird hier eine Geschwindigkeitsverringerung oder Abstandsgewinnung empfohlen.

Cluster 2: Leicht erhöhter Abstand und Geschwindigkeit. Häufig zu finden in Städten wo sich der Verkehr nicht gerade im Stop & Go befindet, sondern fährt. Kann durchaus schnell gefährlich werden, da in Städten oftmals unerwartet gebremst wird. Somit erhöhtes Relevanzlevel.

Cluster 3: Dieses Cluster stellt eine etwas harmlosere Variante als Cluster 1 dar. Die Geschwindigkeiten in diesem Cluster sind immer noch hoch, allerdings sind die Abstände hierfür auch höher. Das macht dieses Cluster etwas weniger gefährlich als Cluster 1. Dennoch ist aufgrund der hohen Geschwindigkeit Vorsicht geboten.

Nachdem die Cluster gebildet wurden, wurde der die Datei aus Kapitel X.X dementsprechend erweitert. Sodass zu jedem Datensatz nun auch das passende Cluster zugewiesen ist.

## 9.2 Entwicklung des Klassifikationsmodells

Nachdem die Daten im vorigen Kapitel klassifiziert werden konnten, kann nun im zweiten Schritt ein Modell entwickelt werden, welches in der Lage ist, die Klassifikationen zu lernen und später selbstständig neue Datensätze zu klassifizieren.

In diesem Kapitel werden zwei Modelle entwickelt. Da es sich hierbei um eine Klassifikation handelt, kann der Reinforcement Learning Ansatz verwendet werden, welcher in Kapitel X.X konzipiert und entwickelt wurde. Als zweites Modell wird ein herkömmliches Feed Forward Network entwickelt. Ein Vergleich der beiden Netze wird daraufhin in Kapitel X.X stattfinden.

### 9.2.1 Festlegung der optimalen Hyperparameter

Bevor es an die Entwicklungen der beiden Modelle geht, wird in diesem Kapitel erstmal festgelegt welche Hyperparameter mit den vorhandenen Daten am besten funktionieren. Hierfür wird der GridSearch Algorithmus verwendet, welcher bereits in Kapitel X.X vorgestellt wurde verwendet.

Für diesen Fall wurde dabei der Wertebereich [4, 8, 16, 32, 64 128, 256] festgelegt und der GridSearch Algorithmus wurde einmal mit einer versteckten und einmal mit zwei versteckten Schichten getestet. Am Ende wurden dann mithilfe der Bibliothek Matplotlib Abbildungen erstellt, die die Ergebnisse visualisieren und bei der Entscheidungsfindung helfen sollen.

Folgende Ergebnisse konnten mithilfe des GridSearch Algorithmus ermittelt werden.

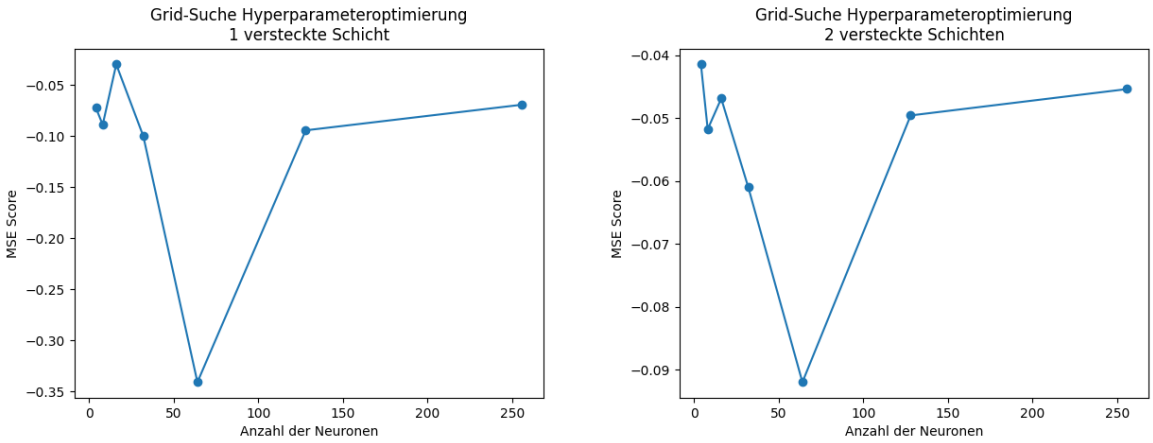


Abbildung 9.3: Grid Search Risikoklassen-Klassifikation

Das beste Ergebnis bei einer versteckten Schicht erreicht man mit 16 Neuronen. Dort konnte während den Tests ein Mean Squared Error von ca. -0.04 erreicht werden.

Bei zwei versteckten Schichten konnte das beste Ergebnis bei einem Mean Squared Error von ca. -0.04 mit 4 Neuronen pro Schicht erreicht werden.

Betrachtet man nun Abbildung X.X so lässt sich sehen, dass das beste Ergebnis bei ca. 0.04 liegt. Dieses Ergebnis kann bereits mit 4 Neuronen bei zwei versteckten Schichten erreicht werden. Demnach macht das nun Sinn, dass das die zu entwickelten Netze genau diese Anzahl an Schichten und Neuronen erhalten. Hieraus ergibt sich eine Architektur, welche mit der folgenden Abbildung verglichen werden kann.

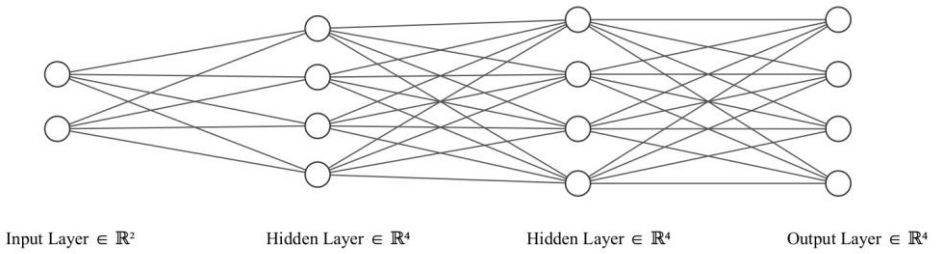


Abbildung 9.4: Architektur des neuronalen Netzes für die Risikoklassen-Klassifikation

### 9.2.2 Implementierung Reinforcement Learning

Der Code für einen Reinforcement Learning Modell wurde bereits in Kapitel X.X geschrieben und dabei so generisch gehalten, sodass in diesem Kapitel lediglich eine Festlegung der Parameterübergabe stattfinden wird.

Folgende Parameter werden an die Agentenklasse übergeben:

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Action_space | 4 | Action Space beschreibt wie viele Ausgabeneuronen/Aktionen der Agent ausführen kann. In den Daten existieren 4 Klassen, somit 4 Klassifikationsmöglichkeiten und 4 Ausgabeneuronen. |
| Observation_space | 2 | Obervation Space beschreibt die Anzahl der Input Neuronen. Betrachtet man die Datensätze, so werden 2 verschiedene Daten übergeben, also 2 Neuronen |
| Num_of_layers | 2 | Anzahl der verborgenen Schichten, festgelegt mithilfe von GridSearch |
| Num_of_neurons_per_layer | 4 | Anzahl der Neuronen in den verborgenen Schichten. Ebenfalls mithilfe von GridSearch festgelegt |

Tabelle 13: Parameterübergabe des Agenten der Risikoklassen-Klassifikation mit Reinforcement Learning

Folgende Parameter werden an die Umgebung übergeben:

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Data | Weiterverarbeiteten Daten aus Kapitel X.X als Array | Die Bedeutung bzw. die Form der Daten kann im jeweiligen Kapitel nachgesehen werden |

| | | |
|---|---|---|
| True_classes | Die weiterverarbeiteten wahren Klassen aus Kapitel X.X als Array | Hier kann ebenfalls die Bedeutung bzw. die Form der Daten im jeweiligen Kapitel nachgesehen werden. |

Tabelle 14: Parameterübergabe Umgebung der Risikoklassen-Klassifikation mit Reinforcement Learning

Das Modell wird so lange trainiert bis keine deutlichen Verbesserungen in der Präzision mehr stattfinden. Danach wird das Modell exportiert und kann jederzeit für spätere Zwecke geladen und wiederverwendet werden.

### 9.2.3 Implementierung Gewöhnliches Feed Forward Netz

Für die Implementierung des gewöhnlichen Feed Forward Netzes wird ähnlich im vorigen Kapitel die hierfür entwickelte Vorlage aus Kapitel X.X verwendet. Demnach wird in diesem Kapitel lediglich eine Parameterfestlegung und Übergabe stattfinden. Folgende Parameter werden an die entwickelte Vorlage Übergeben:

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Number_of_output_neurons | 4 | Anzahl der Ausgabeneuronen, welche mit der Anzahl der Klassifizierungen gleichgestellt werden, kann. Da es vier Klassen gibt, wird es demnach vier Ausgabeneuronen geben. |
| Number_of_layers | 2 | Zwei versteckte Schichten, welche mithilfe von GridSearch erarbeitet wurden. |
| Number_of_neurons | 4 | 4 Neuronen in den versteckten Schichten. Ebenfalls mithilfe von GridSearch erarbeitet. |
| Number_of_input_neurons | 2 | 2 Inputneuronen, welche die weiterverarbeiteten Daten aus Kapitel X.X widerspiegeln. |
| Train_data | Weiterverarbei- | Zur Einsicht der Daten, ihrer Bedeutung |

| | | |
|---|---|---|
| | tete Daten aus Kapitel X.X als Array | und Form wird auf das Kapitel verwiesen. |
| Val_data | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Val_labels | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Epochs | 10 | Es wird für 10 Epochen trainiert. |
| Model_file_name | „risc_classification_model.h5” | Der Dateiname unter dem das Modell gespeichert werden soll. |

Tabelle 15: Parameterübergabe der Risikoklassen-Klassifikation mit gewöhnlichen neuronalen Netzen

Das entstehende Modell wird dabei so lange trainiert bis im MSE keine Verbesserung mehr in den Validierungsdaten stattfindet. Dies dient der Vermeidung von Overfitting. Das entwickelte Modell wird daraufhin gespeichert und kann dann für spätere Zwecke verwendet werden.

## 9.3 Modell zur Prognose von Verkehrsteilnehmeraktion

Als dritter Schritt wurde festgelegt ein Modell zu entwickeln, welches in der Lage ist die nächste Aktion eines Verkehrsteilnehmers zu prognostizieren. Dabei muss erstmals noch spezifiziert werden was genau als Aktion zu verstehen ist. Im Versuch Relevanz zu klassifizieren, wird der TTC-Parameter verwendet, welcher bereits in Kapitel X.X vorgestellt wurde. Damit TTC zwischen zwei Verkehrsteilnehmern berechnet werden kann, werden hauptsächlich zwei Parameter von beiden Parteien benötigt. Der erste Parameter ist die Geschwindigkeit beider Parteien und der zweite Parameter die Distanz zwischen den beiden Parteien. Der zweite Parameter ist allerdings schon vorhanden und kann auch für den nächsten Zeitschritt mit einer Schätzung der nächsten Geschwindigkeit errechnet werden. Demnach ist es von Bedeutung

die nächste Geschwindigkeit eines Teilnehmers vorauszusagen, um daraufhin den zukünftigen TTC-Parameter berechnen und klassifizieren zu können.

### 9.3.1 Entwicklung der Modi

Im vorigen Kapitel wurde vorgestellt das dem Modell, welches noch entwickelt wird, ein Modus übergeben wird. Dieser kann dann beispielsweise bei einem Wert von 0 als „Langsames Beschleunigen“ interpretiert werden. Doch wie genau lässt sich eine langsame Beschleunigung von einem Auto oder einem Fußgänger interpretieren? Diese Frage wird soll in diesem Kapitel behandelt werden und es wird vorgestellt, wie der Modus für jede Verkehrsteilnehmerklasse berechnet werden konnte und die bereits vorhandenen Daten daraufhin mit dem gewünschten Modus versehen werden konnten.

Die Modi wurden entwickelt, indem die Daten genommen und anhand dieser Daten Quartile gebildet wurden. Demnach wurden für diese Daten das 25%, 50% und 75% Quartil errechnet und nach dem folgenden Code fand daraufhin die Zuweisung der Modi statt.

```
clustering_values = []
for i in data:
    if only_acceleration:
        clustering_values.append(float(i[2]) - float(i[1]))
    if only_brake:
        clustering_values.append(float(i[1]) - float(i[2]))

percentile_25 = np.percentile(clustering_values, 25)
percentile_75 = np.percentile(clustering_values, 75)
```

Code 8: Implementierung der Modi

Daraufhin wird im nachfolgenden Code das folgendes Schema verwendet: Liegt die Geschwindigkeit unter dem 25% Prozent Quartil, so findet eine Zuweisung des niedrigsten Modus statt (Das hängt davon ab ob gebremst oder beschleunigt wurde). Datenpunkte zwischen 25% und 75% werden als normal gewertet und alles über 75% wird die Zuweisung als hoch erhalten. Der Modus welcher bislang in den Daten gemäß Kapitel X.X fehlte, wurden daraufhin zu den bestehenden Daten hinzugefügt, sodass die Datensätze nun vollständig sind.

### 9.3.2 Festlegung der optimalen Hyperparameter

Auch für dieses Modell werden, wie in den vorherigen Entwicklungen, die optimalen Hyperparameter für das neuronale Netz festgelegt. Dabei wird der GridSearch Algorithmus ange-

wendet, welcher in Kapitel X.X vorgestellt wurde. Es wird dabei der Wertebereich [4, 8, 16, 32, 64, 128, 256] verwendet und getestet. Dabei wurden die Tests jeweils einmal für eine versteckte Schicht und für zwei versteckte Schichten durchgeführt. Zum Abschluss wurde mithilfe der Bibliothek Matplotlib eine Abbildung erstellt, welche die Ergebnisse zusammenfasst und einen Überblick zur besseren Entscheidungsfindung verschafft.

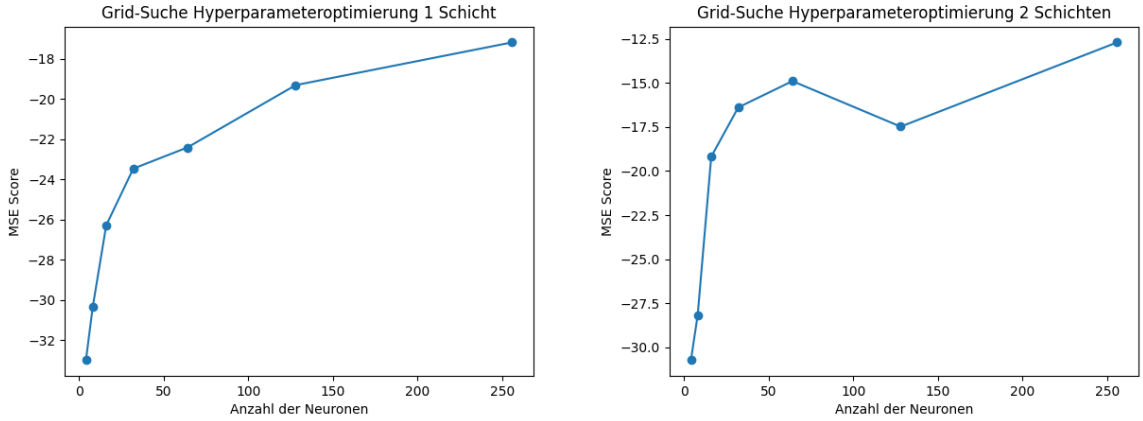


Abbildung 9.5: Grid search Geschwindigkeitsprognose

Bei einer versteckten Schicht konnte das beste Ergebnis mit 256 Neuronen erzielt werden. Dabei wurde ein Mean Squared Error von -17 erreicht. Mit zwei versteckten Schichten wurde das beste Ergebnis ebenfalls mit 256 Neuronen erzielt, allerdings konnte dadurch ein noch besseres Ergebnis erzielt, welches sich bei -12,5 befindet.

Somit macht es für die Entwicklung dieses Netzes Sinn, eine Architektur zu wählen, welche zwei verstecke Schichten mit jeweils 256 Neuronen in diesen Schichten enthält. Eine Abbildung des Netztes wird in diesem Fall allerdings nicht eingefügt, da diese Aufgrund ihrer Größe und der Einschränkung des Dokuments zu ungenau dargestellt wird. Somit wird für einen Einblick auf die Projektabgabe verwiesen.

### 9.3.3 Implementierung

Nachdem die Daten nun fertig vorbereitet wurden, kann die Idee aus Kapitel X.X umgesetzt werden. Wie bereits erwähnt lässt sich das Reinforcement Learning Vorlage nicht anwenden, da diese auf Klassifikation ausgelegt ist. Demnach wird die Die Vorlage aus X.X verwendet und folgende Parameter werden dabei übergeben.

| Parametername | Wertzuweisung | Bedeutung |
|---|---|---|
| Number_of_layers | 2 | Zwei versteckte Schichten, welche mithilfe von GridSearch erarbeitet wurden. |
| Number_of_neurons | 256 | 256 Neuronen in den versteckten Schichten. Ebenfalls mithilfe von GridSearch erarbeitet. |
| Number_of_input_neurons | 3 | 3 Inputneuronen, welche die weiterverarbeiteten Daten aus Kapitel X.X widerspiegeln. |
| Train_data | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Val_data | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Val_labels | Weiterverarbeitete Daten aus Kapitel X.X als Array | Zur Einsicht der Daten, ihrer Bedeutung und Form wird auf das Kapitel verwiesen. |
| Epochs | 10 | Es wird für 10 Epochen trainiert. |
| Model_file_name | „velocity_prediction_model.h5” | Der Dateiname unter dem das Modell gespeichert werden soll. |

Tabelle 16: Parameterübergabe der Geschwindigkeitsprognose

Das Modell wird dabei so lange trainiert bis im Validierungsdatensatz keine erhebliche Verbesserung mehr auftritt. Dies dient dabei der Prävention von Overfitting. Zudem wurde das Modell zum Abschluss gespeichert und kann für spätere Zwecke wiederverwendet werden.

# 10 Evaluation und Diskussion

Nachdem in den vorigen Kapiteln alle Modelle umgesetzt wurden, werden in diesem Kapitel mithilfe der Validierungsdatensätze aus Kapitel X die Modelle evaluiert. Dabei wird vor allem bei Klassifikationsmodellen die Präzision beachtet, um daraus Rückschlüsse ziehen zu können und anschließend die Ergebnisse beurteilen zu können. Bei dem Modell der Geschwindigkeitsvorhersage wird die durchschnittliche Abweichung betrachtet, um beurteilen zu können, wie gut der gewählte Ansatz in den Validierungsdaten funktioniert.

## 10.1 Verkehrsteilnehmerklassifikation

### 10.1.1 Evaluation mit simulierten Validierungsdaten auf simulativ trainierten Modellen

#### 10.1.1.1 Reinforcement Learning

Wendet man die simulierten Validierungsdaten auf das Modell an, welches lediglich mit simulierten Daten trainiert und Reinforcement Learning wurde, so ergibt sich daraus die nachfolgende Präzisionsmatrix:

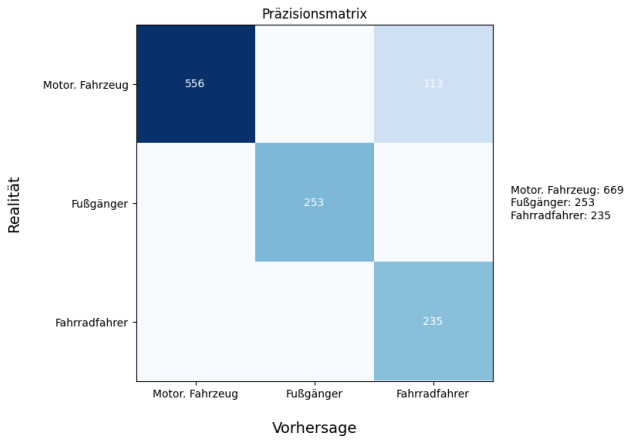


Abbildung 10.1: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit simulierten Validierungsdaten auf simulativ trainiertes Reinforcement Learning Modell

Das Modell konnte mit den Validierungsdaten eine Gesamtpräzision von 90% erzielen, was erstmals auf ein solides Ergebnis hindeutet.

Betrachtet man daraufhin die Präzision der Klassen Fußgänger und Fahrradfahrer, konnte dieses Modell eine Präzision von 100% in beiden Klassen erreichen.

Lediglich bei Erkennung von motorisierten Fahrzeugen gab es einige Abweichungen zur Klassen Fahrrad. Das bedeutet das einige Fahrzeugdatensätze als Fahrräder interpretiert wurden. Hieraus ergibt sich für die Klasse Fahrzeug eine Präzision von 83%.

Aus diesen Ergebnissen kann man daher durchaus interpretieren, dass Reinforcement Learning bei einer ausreichenden Menge an Daten ein geeigneter Ansatz für die Klassifikation von Verkehrsteilnehmern ist. Die Verwechslung zwischen Autofahrern und Fahrradfahrern lässt sich daraus vermuten dass vermutlich eine Grenze in den Daten erreicht wurde. Ein Fahrzeug, welches einem Fahrrad hinterherfährt und dieses nicht überholen kann, nimmt vermutlich sehr ähnliche Werte für Geschwindigkeit oder auch Beschleunigung an. Demnach lässt sich vermuten, dass eine solche Verwechslung zu einem gewissen Maß normal ist.

#### 10.1.1.2 Gewöhnliches Feed Forward Netz

Wendet man die simulierten Validierungsdaten nun auf ein Modell an, welches ebenfalls mit simulierten Daten trainiert wurde, allerdings als Trainingsmethode, die von gewöhnlichen neuronalen Netzen verwendet, so ergibt sich daraus die folgende Präzisionsmatrix.

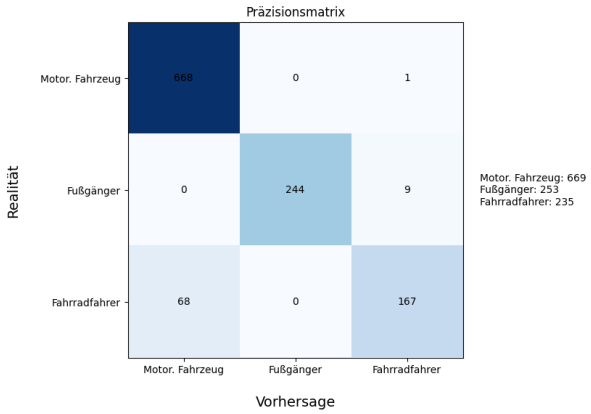


Abbildung 10.2: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit simulierten Validierungsdaten auf simulativ trainiertes gewöhnliches neuronales Netzwerk

Das Modell konnte eine Gesamtpräzision von 93% erzielen, was sich somit als noch etwas besser im Vergleich zu Reinforcement Learning herausstellt.

Betrachtet man die Präzision der Klasse Fußgänger so kann das Modell für diese Klasse eine Präzision 96% vorweisen. Die restlichen 4% der Datensätze wurden mit Fahrradfahrer-Datensätzen verwechselt.

Für die Klasse Fahrrad ergibt sich eine Präzision von 71%. Dies stellt sich als die unpräziseste Klasse heraus und die größte Verwechslung mit 29% fand mit der Klasse motorisiertes Fahrzeug statt.

Die Klasse motorisiertes Fahrzeug schneidet mit einer Präzision von 99,8% am besten ab. Hier gab es lediglich eine Verwechslung mit der Klasse Fahrradfahrer.

Insgesamt betrachtet schneidet dieses Modell somit im Vergleich zu Reinforcement Learning etwas besser ab. Allerdings ist das schlechteste Ergebnis der Klasse Fahrradfahrer mit 71% nochmals um 12% schlechter als das schlechteste Ergebnis aus dem Reinforcement Learning Modell. Somit müsste man an dieser Stelle eine Unterscheidung der Präferenzen festlegen. Legt man mehr wert auf eine höhere Gesamtpräzision, so würde sich dieses Modell besser eignen. Würde man allerdings mehr Wert auf eine bessere Verteilung der Klassifikation setzen, sollte man vermutlich die Reinforcement Lösung wählen.

Die Abweichungen, die in diesem Modell entstanden sind, haben vermutlich ein ähnlicher Grund wie bei Reinforcement Learning. Ein Fahrzeug, dass einem Fahrrad hinterherfährt, hat für diesen Zeitraum vermutlich ähnliche Daten Werte für Geschwindigkeit oder Beschleunigung, sodass hier diese Verwechslungen zu Stande kommen

### 10.1.2 Evaluation mit realen Validierungsdaten auf simulativ trainierten Modellen

#### 10.1.2.1 Reinforcement Learning

Wendet man die realen Validierungsdaten auf das Modell an, welches lediglich mit simulierten Daten und mittels Reinforcement Learning trainiert wurde, so ergibt daraus die nachfolgende Präzisionsmatrix:

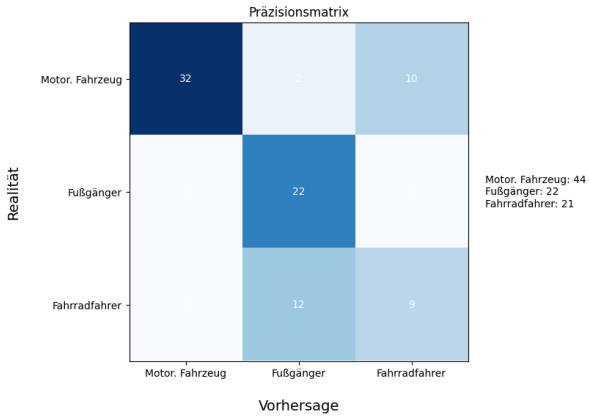


Abbildung 10.3: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf simulativ trainiertes Reinforcement Learning Modell

Die Präzisionsmatrix ergibt eine Gesamtpräzision von 72%, was sich als eine Verschlechterung im Vergleich zu den Simulationsdaten herausstellt.

Die Präzision der Klasse Fußgänger konnte mit 100% beibehalten werden, hieraus ergibt sich also keine Verschlechterung.

In der Klasse Fahrradfahrer konnte allerdings lediglich eine Präzision von 42% beibehalten werden. Im Vergleich zu den Simulationsdaten stellt das eine Verschlechterung um 58% dar.

Die Klasse motorisiertes Fahrzeug kann hingegen noch eine Präzision 72% vorweisen, was allerdings zu den Simulationsdaten ebenfalls eine Verschlechterung um 11% darstellt.

Insgesamt war eine Verschlechterung der Präzision zu erwarten, da auf ein Modell schlichtweg Daten angewendet wurden, die einen ganz anderen Kontext vorweisen und in der echten Welt gewonnen wurden. Trotz alledem konnte sich das Modell sogar als beständig erweisen, vor allem bei der Fußgängerklassifikation. Daraus kann man schließen das Reinforcement Learning ein beständiger Ansatz ist, der auch bei Abweichungen zuverlässige Ergebnisse liefern kann.

#### 10.1.2.2 Gewöhnliche neuronale Netze

Wendet man nun die realen Validierungsdaten auf das Modell an, welches mit simulierten Trainingsdaten und normalem Lernverfahren trainier wurde, so ergibt sich daraus folgende Präzisionsmatrix.

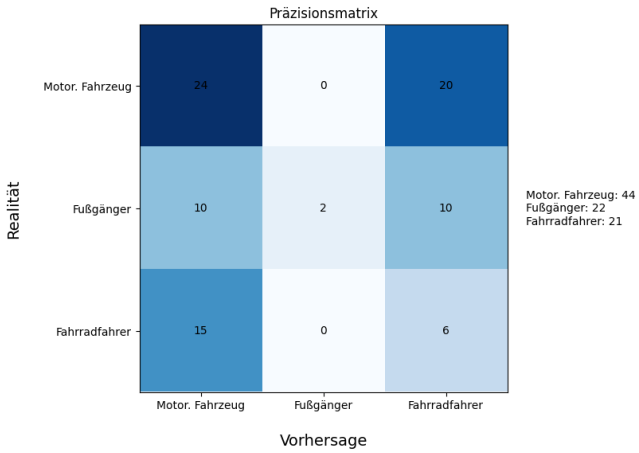


Abbildung 10.4: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf simulativ trainiertes gewöhnliches neuronales Netzwerk

Hieraus ergibt sich eine Gesamtpräzision von 36%, was somit im Vergleich zu allen anderen Verfahren als eine deutliche Verschlechterung hervorsticht.

Die Klasse Fußgänger schneidet hierbei mit einer Präzision von 9% am schlechtesten ab. Im Vergleich zu den Simulationsdaten ist dies eine Verschlechterung um 87%.

Die Klasse Fahrradfahrer hat nur noch eine Präzision von 28% und schneidet im Vergleich zu den Simulationsdaten 48% schlechter ab.

Zuletzt folgt die Klasse motorisiertes Fahrzeug, welche eine Präzision von 54% vorweisen kann. Ebenfalls ist hier im Vergleich zu den Simulationsdaten eine Verschlechterung von 45,8% zu beobachten.

Hieraus lässt sich schließen, dass Reinforcement Learning eine vermutliche beständigere Methode ist, wenn es darum geht auch Daten aus einem anderen Kontext anzubinden. Sollte man daher die Wahl treffen gewöhnliche neuronale Netze zu entwickeln, ist es empfehlenswert den entwickelten Modellen demnach auch nur Daten aus dem Kontext zu geben, mit dem es auch trainiert wurde.

### 10.1.3 Evaluation mit realen Validierungsdaten auf real trainierten Modellen

#### 10.1.3.1 Reinforcement Learning

Verwendet man die realen Validierungsdaten auf das Modell, welches mit realen Trainingsdaten und mittels Reinforcement Learning trainiert wurde, so ergibt sich daraus die nachfolgende Präzisionsmatrix.

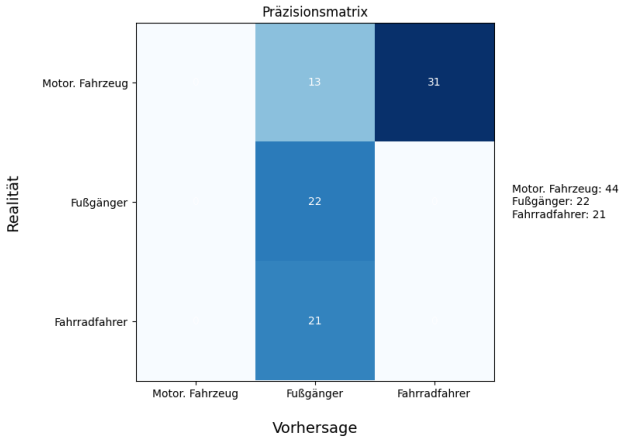


Abbildung 10.5: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf real trainiertes Reinforcement Learning Modell

Es konnte dabei eine Gesamtpräzision von 25% erreicht werden. Das entspricht im Vergleich zur Trainingsdaten-Variante eine Verschlechterung um insgesamt 65%.

Die Klasse Fußgänger konnte dabei allerdings noch mit einer Präzision von 100%, das hohe Niveau halten.

Die Klasse motorisiertes Fahrzeug und Fahrradfahrer hingegen konnten beide eine Präzision von 0% vorweisen, was sich somit für diese beiden Klassen als ein kompletter Ausfall interpretieren lassen kann.

Der Grund für das niedrige Ergebnis wird voraussichtlich an den wenigen Daten liegen, die dem Modell für das Training zur Verfügung standen. Der einzige Unterschied zwischen den anderen Reinforcement Learning Modellen ist nämlich, dass den anderen Modellen deutlich mehr Daten zur Verfügung standen. Daraus lässt sich schließen, dass für Reinforcement Learning voraussichtlich mehr Daten zur Verfügung stehen sollten, um gute Ergebnisse erzielen zu können.

#### 10.1.3.2 Gewöhnliche neuronale Netze

Verwendet man reale Validierungsdaten auf ein Modell, welches mit realen Trainingsdaten und gewöhnlichen neuronalen Netzen trainiert wurde, so ergibt sich daraus die nachfolgende Präzisionsmatrix.

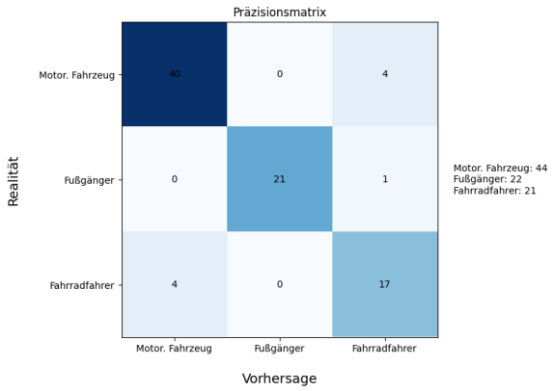


Abbildung 10.6: Verkehrsteilnehmerklassifikation Präzisionsmatrix mit realen Validierungsdaten auf real trainiertes gewöhnliches neuronales Netz

Das Modell kann eine Gesamtpräzision 90% Prozent vorweisen, was im Vergleich zu Reinforcement Learning als eine deutliche Steigerung interpretiert werden kann. Die Steigerung in der Gesamtpräzision beträgt dabei 65%.

Die Klasse Fußgänger kann dabei eine Präzision von 95% vorweisen. Lediglich ein Datensatz wurde als Fahrradfahrer klassifiziert.

Die Klasse Fahrradfahrer kann eine Präzision von 80% vorweisen. Im Vergleich zu Reinforcement Learning stellt dies eine große Steigerung dar, da bei Reinforcement Learning kein einziger Fahrraddatensatz richtig erkannt wurde.

Die Klasse Motorisiertes Fahrzeug kann eine Präzision von 91% vorweisen. Hier ist ebenfalls im Vergleich zu Reinforcement Learning eine große Steigerung zu beobachten, da bei Reinforcement Learning ebenfalls kein einziger Datensatz richtig erkannt wurde.

Dieses Modell konnte im Vergleich zum Reinforcement Learning Modell deutlich besser abschneiden und dass, obwohl die gleichen Trainings- und Validierungsdatensätze verwendet wurden. Hieraus lässt sich möglicherweise schließen dass gewöhnliche neuronale Netze besser geeignet sind, wenn nicht viele Daten zur Verfügung stehen. Sollte also eine geringe Menge an Daten vorliegen, ist es womöglich empfehlenswerter gewöhnliche neuronale Netz anstatt Reinforcement Learning als Trainingsmethode zu verwenden.

## 10.2 Risikoklassen-Klassifikation

Im Bereich der Risikoklassen-Klassifikation konnte nicht mit realen Daten gearbeitet werden, weswegen lediglich eine Evaluation mit simulierten Validierungsdaten auf simulativ trainierte Modelle stattfindet. Dabei gibt es zwei Gründe, weswegen Realdaten nicht verwendet werden konnten. Der erste Grund ist die vorhandene Menge an Realdaten. Es gibt schlichtweg nicht genug Daten, um aussagekräftige Cluster bilden zu können. Der zweite Grund, der sich mit dem ersten Grund ein wenig überschneidet, ist, dass sich die Klassifizierung dieses Kapitels mit Risiko beschäftigt. Die Teilnehmer wurden aus Rücksicht zu sich selbst und anderer Verkehrsteilnehmer gebeten kein Risiko während der Datensammlung einzugehen, sodass die vorhandenen Realdaten ohne großes Risiko aufgezeichnet wurden.

### 10.2.1 Reinforcement Learning

Verwendet man Reinforcement Learning als Trainingsmethode eines Modells, um den Validierungssatz mithilfe der gebildeten Cluster aus Kapitel X.X klassifizieren zu lassen, dann ergibt sich daraus die nachfolgende Präzisionsmatrix.

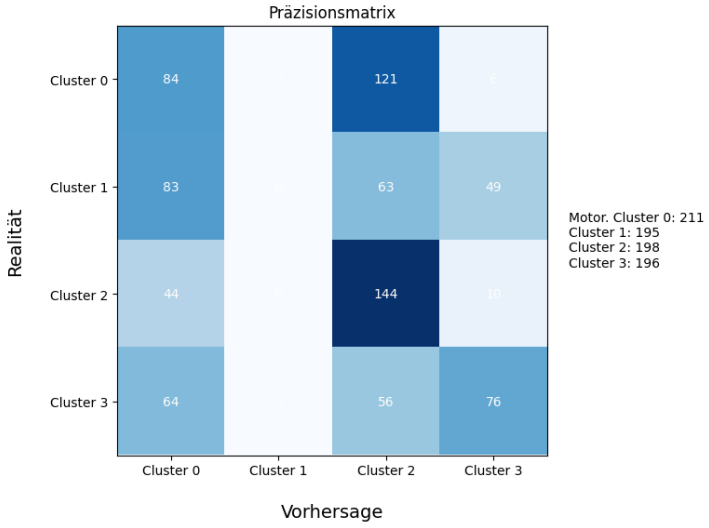


Abbildung 10.7: Risikoklassen-Klassifikation Präzisionsmatrix Reinforcement Learning

Das Modell kann eine Gesamtpräzision von 38% vorweisen.

Cluster 0 weist dabei eine Präzision von 39% auf. Die größte und einzige Verwechslung trat dabei mit Cluster 2 auf.

Cluster 1 hat eine Präzision von 0%. Das Model war nicht in der Lage Datensätze richtig zu Cluster 1 zuzuweisen. Demnach traten nur Verwechslungen auf und diese Verwechslung sind relativ gleichmäßig auf die anderen Cluster verteilt.

Cluster 2 kann mit einer Präzision von 72% die höchste Präzision aufweisen. Dabei trat die höchste und einzige Verwechslung mit Cluster 0 auf.

Cluster 3 weist eine Präzision von 38% auf. Das gibt dem Cluster eine ähnliche Präzision wie Cluster 0. Allerdings treten in diesem Cluster Verwechslungen mit Cluster 0 und Cluster 2 auf.

Insgesamt ist das Ergebnis dieses Klassifikationsmodells eher unpräzise ausgefallen, da nicht einmal die Hälfte der Validierungsdatensätze korrekt klassifiziert werden konnte. Der Grund, weswegen die Präzision so niedrig ausgefallen ist, lässt sich an dieser Stelle allerdings schwer sagen. Die Vermutung lässt daraufsetzen, dass für einige Cluster einfach zu wenig Daten vorhanden waren und einige Datensätze in den Trainingsdaten somit mehrfach auftauchten. Cluster 1 beispielsweise hatte die wenigsten Trainingsdaten vorzuweisen und hat demnach auch am schlechtesten in der Klassifikation abgeschnitten. Diese Aussage bestätigt sich beispielsweise mit Cluster 2, welches mehr verschiedene Daten zum Training bereitstellen konnte und mit 72% ein deutlich besseres Ergebnis erzielte. Cluster 0 hingegen hatte ähnlich viele Datensätze wie Cluster 2, konnte allerdings dennoch nicht so gut vom Modell klassifiziert werden. Eine konkrete Aussage zum ungenauen Ergebnis zu treffen gestaltet sich mit den aktuell vorliegenden Daten als schwierig und es sollte entweder ein andere Lernmethode für das Modell oder ein weiterer Versuch mit mehr Daten vorgenommen werden, um präzisere Aussagen treffen zu können.

### 10.2.2 Gewöhnliche neuronale Netze

Wendet man nun die Validierungsdaten auf ein Modell an, welches gewöhnliche neuronale Netze als Training verwendet hat, so ergibt sich daraus die nachfolgende Präzisionsmatrix.

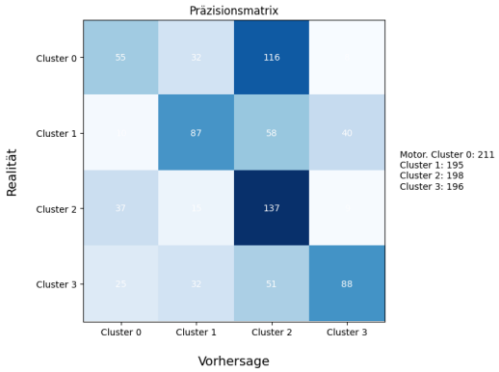


Abbildung 10.8: Risikoklassen-Klassifikation Präzisionsmatrix anhand von gewöhnlichen neuronalen Netzen

Das Modell kann mit den Validierungsdaten eine Gesamtpräzision von 46% aufweisen. Somit schneidet dieses Modell etwa um 16% besser ab als das Modell, welches mit Reinforcement Learning trainiert wurde.

Cluster 0 kann eine Präzision von 26% vorweisen. Dies ist im Vergleich zu Reinforcement Learning eine Verschlechterung um 13%.

Cluster 1 hat nun eine Präzision 44%. Im Vergleich zu Reinforcement Learning eine deutliche Verbesserung, da dort kein einziger Datensatz richtig klassifiziert werden konnte.

Cluster 2 schneidet mit einer Präzision von 69% wieder am besten ab. Dadurch ist im Vergleich zu Reinforcement Learning eine kleine Verschlechterung um 3% entstanden, befindet sich allerdings immer noch grob auf demselben Niveau.

Cluster 3 weist eine Präzision von 45% auf. Das ist im Vergleich zu Reinforcement Learning eine Steigerung von 7%.

Insgesamt schneiden gewöhnliche neuronale Netze im Gesamtergebnis etwas besser ab als Reinforcement Learning. Allerdings sind diese Ergebnisse trotz der Verbesserung immer noch Verbesserungspotential offen, da diese immer noch nicht die 50% Marke an Präzision erreichen. Ein großer Vorteil von gewöhnlichen neuronalen Netzen ist, das diese nun auch Cluster 1 richtig klassifizieren. Dies kam im Vergleich zu Reinforcement Learning überhaupt nicht vor. Allerdings tut sich dieses Modell etwas schwerer in der Unterscheidung zwischen Cluster 0 und Cluster 2, dort konnte das Reinforcement Learning Modell etwas besser überzeugen. Die Ergebnisse dieses Modells decken sich mit denen aus Kapitel X.X. Dort konnten gewöhnliche neuronale Netze mit weniger Datensätzen deutlich bessere Ergebnisse erzielen

als die Reinforcement Learning Alternativen. Dieser große Sprung konnte in der Risikoklassen-Klassifikation zwar nicht erreicht werden, die Aussage, dass gewöhnliche neuronale Netze bei weniger Datensätzen besser abschneiden, konnte dadurch allerdings gestützt werden. Insgesamt lässt sich zu diesem Modell, wie beim vorherigen Sagen, dass ein erneutes Training mit mehr Datensätzen möglicherweise zu besseren Ergebnissen führen könnte, welche sich auf einem Niveau wie in Kapitel X.X bewegen könnten.

## 10.3 Geschwindigkeitsprognose

Die Geschwindigkeitsprognose führt im Vergleich zu den anderen Evaluationen keine Klassifikation durch. Hier wird anhand der Eingabeparameter ein Wert geschätzt. Demnach wurde für die Geschwindigkeitsprognose lediglich gewöhnliche neuronale Netze trainiert, da die umgesetzte Reinforcement Learning Methode, sich lediglich für die Klassifikation eignet. Parallel dazu wurde nur ein Modell für die Geschwindigkeitsprognose mit Simulationsdaten trainiert, da dieser Teil der Relevanzklassifikation ist und für Realdaten aufgrund der geringen vorhandenen Menge keine Relevanzklassifikation durchgeführt werden konnte.

### 10.3.1 Evaluation pro Verkehrsteilnehmerklasse

Die folgenden durchschnittliche Abweichungen in den Prognosen konnten auf den Validierungsdaten erzielt werden, wenn man die Abweichungen pro Verkehrsteilnehmerklasse betrachtet.

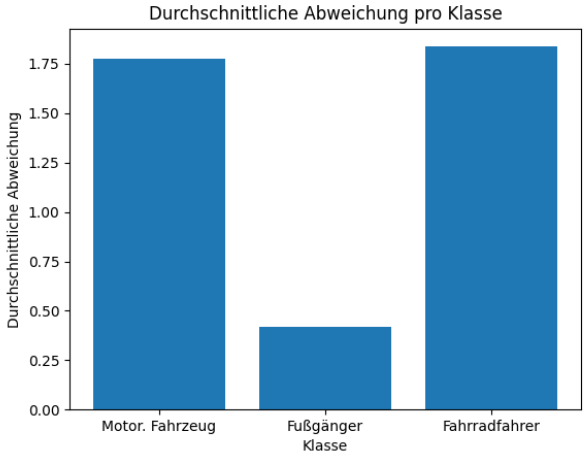


Abbildung 10.9: Geschwindigkeitsprognose durchschnittliche Abweichung pro Klasse

Die Klasse Fußgänger kann dabei mit einer durchschnittlichen Abweichung von 0.45km/h die geringste Abweichung vorweisen.

Motorisierte Fahrzeuge und Fahrradfahrer teilen sich mit einem Wert von ca. 1.75km/h grob dieselbe durchschnittliche Abweichung.

Die Ergebnisse hierzu waren grob in dieser Form zu erwarten, da motorisierte Fahrzeuge und Fahrradfahrer deutlich mehr Spielraum in der Geschwindigkeit haben als Fußgänger und demnach auch mehr Potenzial für Abweichungen bieten können. Etwas überraschend ist an dieser Stelle jedoch, dass Fahrradfahrer und motorisierte Fahrzeuge grob die gleiche durchschnittliche Abweichung haben. Würde man nämlich den Geschwindigkeitsspielraum betrachten, so würde man eher erwarten dass entweder die durchschnittliche Abweichung für Fahrzeuge höher ist oder durchschnittliche Abweichung für Fahrradfahrer niedriger ausfällt. Insgesamt kann das entwickelte Modell in Betrachtung auf diese Auswertung solide Prognosen erstellen.

### 10.3.2 Evaluation pro Modus

Eine andere Betrachtung, die für die Evaluation vollzogen wurde, ist die Betrachtung der durchschnittlichen Abweichung pro Modus. Berechnet man diese mithilfe der Validierungsdaten, so ergibt sich daraus das nachfolgende Bild.

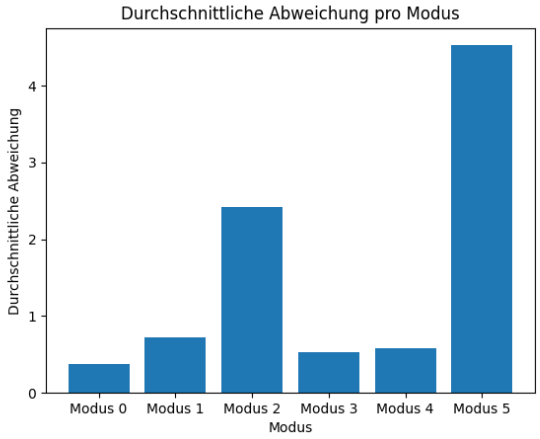


Abbildung 10.10: Geschwindigkeitsprognose durchschnittliche Abweichung pro Modus

Modus 0 – Langsame Beschleunigung, weist mit einer durchschnittlichen Abweichung von ca. 0.3km/h die niedrigste durchschnittliche Abweichung auf.

Modus 1 – normale Beschleunigung, kann eine durchschnittliche Abweichung von ca. 0.75km/h aufweisen.

Modus 2 – starke Beschleunigung, weist mit einer durchschnittlichen Abweichung von ca. 2.5km/h die zweitgrößte Abweichung auf.

Modus 3 – langsame Bremsung, liegt mit einer durchschnittlichen Abweichung von ca. 0.5km/h ebenfalls eher in den niedrigeren Bereichen.

Modus 4 - normale Bremsung, orientiert sich ebenfalls wie Modus 3 mit einer Abweichung von 0.5km/h in den niedrigeren Bereichen.

Modus 5 – starke Bremsung, weist mit einer durchschnittlichen Abweichung von ca. 5km/h die höchste Abweichung in den Validierungsdaten auf.

Die Ergebnisse pro Modus spiegeln in diesem Fall die Erwartungen anhand der Trainingsdaten wider. Einerseits wurden die Erwartungen dadurch widergespiegelt, dass Modus 2 und 5 die höchsten Abweichungen haben, da diese durch Ihre hohen Werte auch den höchsten Spielraum für Variation bieten können und es somit für diese Modi schwerer ist eine genaue Prognose zu treffen. Andererseits spiegeln die Ergebnisse die Daten dadurch wider, dass für die Modi 0, 1, 3 und 4 deutlich mehr Datensätze vorhanden waren als für die Modi 2 und 5. Dadurch hatte das Modell mehr Möglichkeiten aus diesen Modi zu lernen und die Prognosen können demnach genauer ausfallen als für die Modi, in denen weniger Daten vorhanden waren. Insgesamt kann das Modell allerdings auch aus Sicht der Modi solide Ergebnisse vorweisen und für allem für die Modi 0,1,3,4 eingesetzt werden. Für die anderen beiden Modi wäre eventuell ein weiteres Training interessant, sobald für diese Modi mehr Daten vorliegen.

# 11 Zusammenfassung und Ausblick

Das Ziel dieser Masterarbeit war es, ein System zu entwickeln, das in der Lage ist, Verkehrsteilnehmer und deren Relevanz zu klassifizieren. Das System sollte auf der Grundlage von Positionsdaten, die aus GPS-Koordinaten gewonnen wurden, entwickelt werden. Nach Vorstellung der Forschungsfragen und des Grundlagenkapitels wurden die Anforderungen und Visionen für das System vorgestellt. Dabei wurde festgelegt, wie das Endergebnis aussehen sollte. Allerdings ergab sich im Verlauf der Arbeit, dass es sinnvoll wäre, das große Gesamtprojekt in zwei Teilprojekte aufzuteilen.

Bevor die Realisierung der beiden Teilprojekte begann, wurden zunächst die verfügbaren Daten betrachtet. Zwei Quellen wurden genutzt: Zum einen wurden simulierte Daten mithilfe von CARLA, einer Verkehrssimulation, gesammelt. Zum anderen wurden reale Daten von echten Verkehrsteilnehmern mit Hilfe einer App der Hochschule und einiger freiwilliger Teilnehmer gesammelt. Beide Datensätze mussten für die kommenden Realisierungen vorbereitet und weiterverarbeitet werden, was in diesem Kapitel durchgeführt wurde.

Im nächsten Schritt wurde die Entwicklung von zwei maschinellen Lernmethoden angegangen, die als Grundlage für zukünftige Realisierungen dienen sollten. Es wurden sowohl eine generische Implementierung für Reinforcement Learning als auch eine für gewöhnliche neuronale Netze realisiert. Die Idee hinter der generischen Entwicklung war, dass bei der Umsetzung lediglich Parameter übergeben werden müssen, um ein Modell zu entwickeln. Dadurch war es möglich, eine Vielzahl verschiedener Modelle zu entwickeln und später miteinander zu vergleichen und auszuwerten.

Nach der Entwicklung der beiden Methoden wurde das erste Teilprojekt realisiert, welches sich auf die Klassifikation von Verkehrsteilnehmern konzentrierte. Ziel war es, ein System mit neuronalen Netzen zu entwickeln, das Positionsdaten verarbeiten und anhand dieser eine Klassifikation durchführen kann. Dabei wurden drei Klassen definiert: Fußgänger, Fahrradfahrer und motorisierte Fahrzeuge. Es wurden verschiedene Modelle mit unterschiedlichen Datensätzen entwickelt und später im Auswertungskapitel validiert.

Das zweite Teilprojekt widmet sich der Klassifikation von Relevanz aus der Perspektive eines Protagonisten. Die Vision dabei ist, dass aus dieser Perspektive mehrere Prognosen mit anderen Verkehrsteilnehmern erstellt werden können. Mit mehreren Prognosen sollen möglichst viele Szenarien für das Verhalten anderer Verkehrsteilnehmer prognostiziert werden, um dann

den nächsten geplanten Schritt des Protagonisten mit den möglichen nächsten Schritten anderer Verkehrsteilnehmer abzugleichen und das Risiko in allen Szenarien zu minimieren. Hierfür wurden zwei verschiedene Modelle entwickelt. Das erste Modell ist in der Lage, eine Verkehrsteilnehmerklasse, einen Modus und eine Ausgangsgeschwindigkeit zu analysieren und daraufhin eine Prognose für die nächste Geschwindigkeit auszugeben. Diese Prognose kann dann als Basis für den prognostizierten TTC-Parameter dienen, welcher vom zweiten Modell klassifiziert wird. Die Ausgabe vom zweiten Modell sollte eine Risikoklasse sein, welche vorher mittels Clustering gebildet und interpretiert wurden.

In der Verkehrsteilnehmerklassifikation konnte das beste Ergebnis bei einer Gesamtpräzision von 93% für Simulationsdaten erzielt werden, indem ein Modell mit gewöhnlichen neuronalen Netzen trainiert wurde. Bei Realdaten lag die beste Gesamtpräzision ebenfalls bei 90% und konnte ebenfalls mit gewöhnlichen neuronalen Netzen erreicht werden. Zudem zeigte die Auswertung, dass Modelle, die mit Reinforcement Learning trainiert wurden, besser auf Daten mit anderem Kontext abschneiden. Ein Beispiel hierfür ist ein Reinforcement Learning Modell, welches mit Simulationsdaten trainiert wurde und dennoch auf reale Validierungsdaten eine Gesamtpräzision von 75% erreichte.

Für die Relevanzklassifikation wurden die zwei entwickelten Modelle untersucht. Das erste Modell, welches für die Geschwindigkeitsprognose verantwortlich war, lieferte überzeugende Ergebnisse mit nur geringen Abweichungen bis maximal 2,5km/h. Nur in den Modi 2 und 5 waren die Abweichungen etwas höher, wodurch ein zusätzliches Training in diesen Bereichen empfehlenswert wäre. Im Gegensatz dazu konnte das zweite Modell, welches für die Klassifikation der Risikoklassen zuständig war, nicht die gewünschte Zuverlässigkeit erzielen. Mit einer maximalen Präzision von lediglich 46% ist voraussichtlich ein zusätzliches Training mit mehr Daten erforderlich, um eine höhere Zuverlässigkeit zu erreichen.

Insgesamt konnten beide Teilprojekte erfolgreich entwickelt werden und lieferten teilweise zuverlässige und aussagekräftige Ergebnisse. Die Klassifizierung von Verkehrsteilnehmern erwies sich beispielsweise als äußerst zuverlässig, mit einem Spitzenwert von 90%. Ebenso überzeugte die Geschwindigkeitsprognose mit nur geringen Abweichungen. Lediglich die Klassifikation der Risikoklassen zeigte eine unzureichende Präzision. Die Ergebnisse dieser Masterarbeit können jedoch zukünftige Entwicklungen in der Umgebungserkennung unterstützen und weiter vorantreiben. Die entwickelten Methoden offenbarten sowohl Stärken als

auch Schwächen und sollten daher am besten mit anderen Verfahren, wie beispielsweise Kameraverfahren, kombiniert werden, um bestmögliche Ergebnisse zu erzielen.

# Literaturverzeichnis


[Althoff+2009] Althoff, M., Stursberg, O., & Buss, M. (2009). Model-based probabilistic collision detection in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, *10*(2), 299–310.

[Bergstra+2012] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

[CARLA 2023] CARLA Dokumentation https://carla.org/

(Zugriff 18.03.2023).

[Döhmen 2016] Döhmen, T. (2016). Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files [PhD Thesis]. Master's thesis, Vrije Universiteit Amsterdam, www. cwi. nl/boncz/msc/2016 ….

[Dörn 2023] Dokumentation zu neuronalen Netzen von Sebastian Dörn https://sebastiandoern.de/neuronale-netze/

(Zugriff 18.03.2023).

[Goodfellow+2020] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.

[Google 2023] Dokumentation Google Trends https://trends.google.com/home

(Zugriff 10.01.2023).

[Green 2000] Green, M. (2000). „ How long does it take to stop?" Methodological analysis of driver perception-brake times. *Transportation human factors*, *2*(3), 195–216.

[Han+2001] Han, J., & Kamber, M. (2001). Data mining: Concepts and techniques. Elsevier.

[He+2015] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Supasing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

[Jiménez+2013] Jiménez, F., Naranjo, J. E., & García, F. (2013). An improved method to calculate the time-to-collision of two vehicles. *International Journal of Intelligent Transportation Systems Research*, *11*, 34–42.

[Kronach 2016] Landkreis Kronach (2016). 5G-gestützte autonom fahrende Einzel- oder Flottenfahrzeuge für den Einsatz als öffentliche Verkehrsmittel im ländlichen Raum.

[Matplotlib 2023] Matplotlib Dokumentation https://matplotlib.org/

(Zugriff 18.03.2023).

[Nielsen 2015] Nielsen, M. A. (2015). *Neural networks and deep learning* (Bd. 25). Determination press San Francisco, CA, USA.

[Numpy 2023] Numpy Dokumentation https://numpy.org/

(Zugriff 18.03.2023).

[Opitz 2022] Opitz, P. (2022). Klassifizierung der Relevanz von Verkehrsteilnehmern zur Erweiterten Umfeldwahrnehmung Autonomer Fahrzeuge. Hochschule für angewandte Wissenschaften Coburg.

[Python 2023] Python Dokumentation https://www.python.org/

(Zugriff 18.03.2023).

[scikit-learn 2023] Scikit-learn Dokumentation https://scikit-learn.org/stable/

(Zugriff 18.03.2023).

[Sherstinsky 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306.

[Simoncini+2018] Simoncini, M., Taccari, L., Sambo, F., Bravi, L., Salti, S., & Lori, A. (2018). Vehicle classification from low-frequency GPS data with recurrent neural networks. *Transportation Research Part C: Emerging Technologies*, *91*, 176–191.

[Sohl 2022] Sohl, M. (2022). Klassifizierung der Bewegungsmuster von Mobilfunkteilnehmern zur erweiterten Umfeldwahrnehmung autonomer Fahrzeuge. Hochschule für angewandte Wissenschaften Coburg.

[Sutton+2018] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Tensorflow 2023] Tensorflow Dokumentation https://www.tensorflow.org/

(Zugriff 18.03.2023).

[Torlak 2022] Torlak, R. (2022). Detektion der Bewegung von Verkehrsteilnehmern aus Positionsdaten. Hochschule für angewandte Wissenschaften Coburg.

[von der Hude 2020] von der Hude, M. (2020). *Predictive Analytics und Data Mining: Eine Einführung mit R*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-30153-8

[Xiong+2018] Xiong, X., Chen, L., & Liang, J. (2018). Vehicle driving risk prediction based on Markov chain model. *Discrete Dynamics in Nature and Society*, *2018*, 1–12.

# Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine **Bachelorarbeit** mit dem Titel

---

**Verwendung von Positionsdaten zur automatisierten Klassifizierung von Verkehrsteilnehmern mittels maschinellen Lernverfahren**

---

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie nicht an anderer Stelle als Prüfungsarbeit vorgelegt habe.

Ort

Datum

Unterschrift