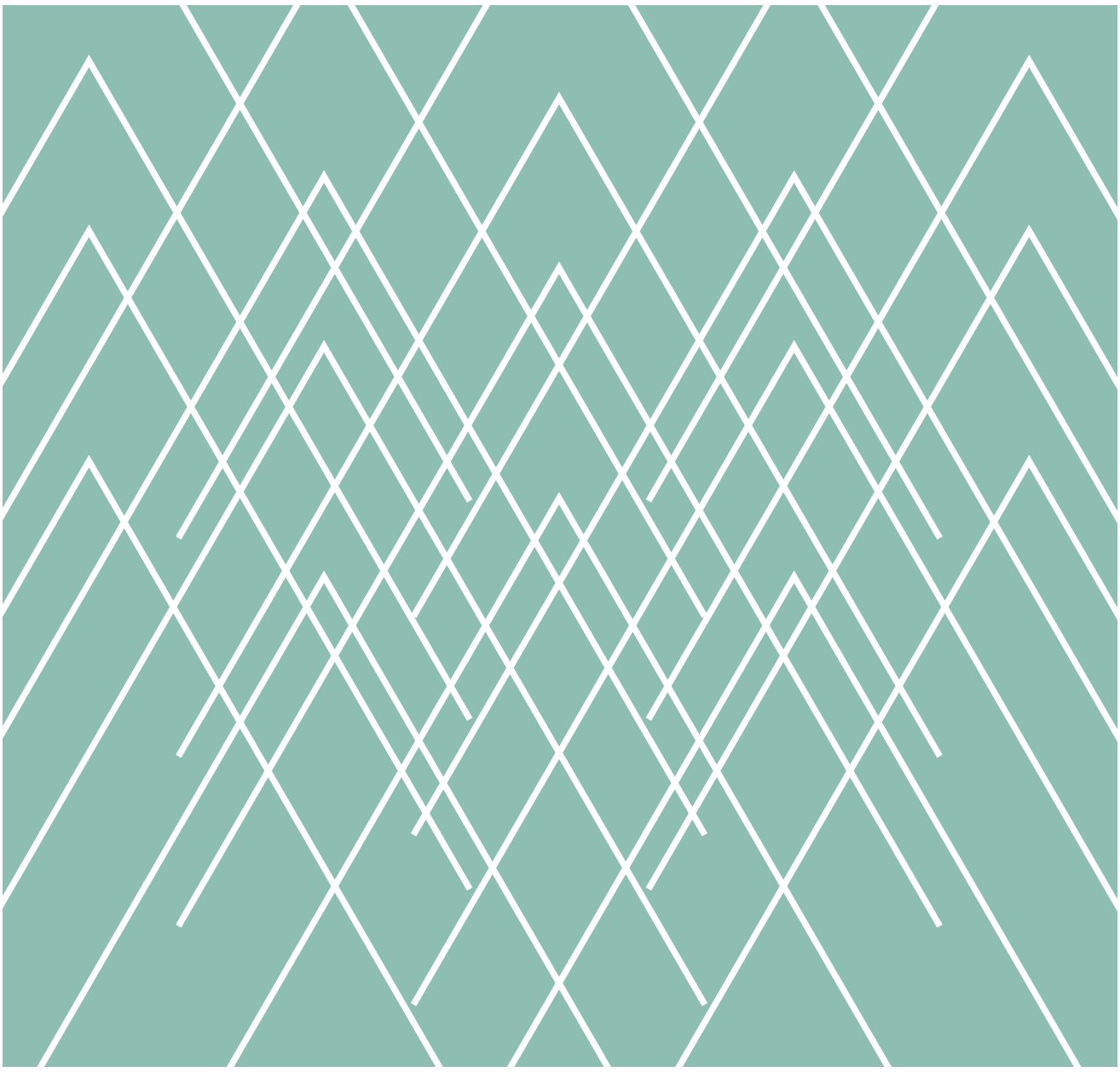
Analyse von Smartphone-Sensordaten mit Rekurrenten Neuronalen Netzen zur Klassifikation von Fahrmanövern

Oliver Otto



▲ Hochschule Harz
Hochschule für angewandte Wissenschaften

WAIT

Impressum

Inhaltlich verantwortlich

Autor/-in der Abschlussarbeit

Institution

Der Fachbereich Automatisierung und Informatik ist ein Fachbereich der Hochschule Harz. Die Hochschule Harz ist eine Körperschaft des öffentlichen Rechts. Sie wird durch den Rektor Prof. Dr. Folker Roland gesetzlich vertreten: info@hs-harz.de.

Umsatzsteuer-Identifikationsnummer

DE231052095

Adresse

Hochschule Harz
Fachbereich Automatisierung und Informatik
Friedrichstraße 57-59
38855 Wernigerode

Kontakt

Dekanin des Fachbereiches Automatisierung und Informatik
Prof. Dr. Andrea Heilmann
Tel.: +49 3943 659 300
Fax: +49 3943 659 300
E-Mail: dekanin-ai@hs-harz.de

Aufsichtsbehörde

Das Ministerium für Wirtschaft, Wissenschaft und Digitalisierung des Landes Sachsen-Anhalt (MW), Hasselbachstraße 4, 39104 Magdeburg, ist die zuständige Aufsichtsbehörde.

ISSN 2702-2293

Haftungsausschluss

Die Hochschule Harz weist auf Folgendes hin:

Die Hochschule Harz ist lediglich für die Veröffentlichung der einzelnen Werke zuständig, sie übernimmt keinerlei Haftung. Vielmehr gilt Folgendes:

- für den Inhalt der Publikation ist der/die Autor/-in verantwortlich
- mit der Erfassung in der Schriftenreihe Wernigeröder Automatisierungs- und Informatik-Texte verbleiben die Urheberrechte beim Autor/bei der Autorin
- die Einhaltung von Urheber- und Verwertungsrechten Dritter liegt in der Verantwortung des Autors/der Autorin

Vor Veröffentlichung bestätigte der/die Autor/-in,

- dass mit der Bereitstellung der Publikation und jedes Bestandteils (z.B. Abbildungen) nicht gegen gesetzliche Vorschriften verstoßen wird und Rechte Dritter nicht verletzt werden
- dass im Falle der Beteiligung mehrerer Autoren am Werk der/die unterzeichnende Autor/-in stellvertretend im Namen der übrigen Miturheber/-innen handelt
- im Falle der Verwendung personenbezogener Daten den Datenschutz (durch Einholen einer Einwilligung des Dritten zur Veröffentlichung und Verbreitung des Werks) zu beachten
- dass im Falle einer bereits erfolgten Veröffentlichung (z.B. bei einem Verlag) eine Zweitveröffentlichung dem Verlagsvertrag nicht entgegensteht
- dass die Hochschule Harz von etwaigen Ansprüchen Dritter (z.B. Mitautor/-in, Miturheber/-in, Verlage) freigestellt ist

Hochschule Harz – Hochschule für angewandte Wissenschaften

Fachbereich Automatisierung und Informatik

Masterarbeit
Ausgezeichnet mit dem Walter Gießler Preis 2019

Analyse von Smartphone-Sensordaten mit Rekurrenten Neuronalen Netzen zur Klassifikation von Fahrmanövern

Autor:

Oliver Otto

10. Oktober 2018

Betreuer:

Prof. Dr. Frieder Stolzenburg

Fachbereich Automatisierung und Informatik

Dr. Kiril Schröder

Otto, Oliver:
Analyse von Smartphone-Sensordaten mit Rekurrenten Neuronalen Netzen zur Klassifikation von Fahrmanövern
Masterarbeit, Ausgezeichnet mit dem Walter Gießler Preis 2019
Hochschule Harz – Hochschule für angewandte Wissenschaften, 2018.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Hochschule Harz
Fachbereich Automatisierung und Informatik

Thema und Aufgabenstellung der Masterarbeit
MA AI 23/2018

für Oliver Otto

Analyse von Smartphone-Sensordaten mit Rekurrenten Neuronalen Netzen zur Klassifikation von Fahrmanövern

Trotz des Rückgangs der Verkehrsunfälle in den vergangenen Jahren kommt es regelmäßig zu einer hohen Zahl von Verkehrsopfern. Allein in Deutschland wurden 2017 beinahe 3000 Menschen durch Verkehrsunfälle getötet. Die Ursachen, die zu Verkehrsunfällen führen, sind vielfältig. Einige sind u.a. Selbstüberschätzung, aggressives Fahrverhalten, Müdigkeit oder Ablenkung beim Fahren.

Einen innovativen Ansatz können Fahrerassistenzsysteme bieten, die das Fahrverhaltens analysieren und so Rückschlüsse auf Fahrmanöver ziehen. Sie könnten Ablenkung oder Müdigkeit erkennen und den Fahrer, der evtl. die Lage nicht immer objektiv einschätzen kann, auf ein gestiegenes Risiko hinweisen. Diese Informationen sind auch für Versicherer von Bedeutung, die so eine aktuelle und individuelle Risikobewertung des Fahrers vornehmen können.

Ziel dieser Arbeit ist die Analyse, inwieweit sich Rekurrente Neuronale Netze zur Klassifikation von Fahrmanövern, repräsentiert durch Zeitreihenabschnitte, einsetzen lassen.

Die Masterarbeit beinhaltet folgende Teilaufgaben:

- Entwicklung und Durchführung eines Experiments zur Datenerhebung,
- Analyse des Stands der Technik und Auswahl eines geeigneten Verfahrens zur Klassifikation der erhobenen Daten,
- Implementierung, Identifikation entscheidender Faktoren für die Klassifikationsaufgabe und Analyse des gewählten Verfahrens,
- Bewertung der Ergebnisse und Ausblick.

Prof. Dr. Frieder Stolzenburg
1. Prüfer

Dr. Kiril Schröder
2. Prüfer

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mir bei der Erstellung dieser Masterthesis durch ihre fachliche und persönliche Unterstützung geholfen haben.

Ich danke Herr Prof. Dr. Stolzenburg für die sehr gute Betreuung, die vielen interessanten Gespräche und Anregungen zu meiner Arbeit. Aber auch für die freundliche Bereitstellung diversen Quellcodes.

Ebenso danke ich Herr Dr. Schröder für die intensive Unterstützung bei der Themenfindung, Hinweise und Anregungen zu meiner Arbeit und die Bereitschaft, die Rolle des Zweitprüfers zu übernehmen.

Weiterhin möchte ich mich bei meiner Familie bedanken, die das Verständnis für so manche Stunden der Abwesenheit aufgebracht hat und es mir ermöglicht hat, mich mit freiem Kopf auf meine Arbeit konzentrieren zu können.

1. Einführung

Dieses Kapitel vermittelt die grundlegende Motivation, einen Ansatz zu untersuchen, mit dem Fahrerassistenzsysteme zur Unfallvermeidung realisiert werden könnten. Die Relevanz von Technologien zur Vermeidung von Verkehrsunfällen und der Minderung von Unfallfolgen wird deutlich, schaut man sich die Anzahl der betroffenen Personen und die Unfallfolgen an. Technologien zur Unfallvermeidung haben daher schon seit Jahrzehnten eine enorme Bedeutung und bilden einen komplexen Forschungszweig. Dieses Kapitel zeigt die grundlegende Problematik und Ziele dieser Arbeit auf sowie deren Aufbau.

1.1 Relevanz der Analyse von Sensordaten

Das Statistische Bundesamt (Destatis) weist in seiner Statistik über Straßenverkehrsunfälle [2, S. 5] für das Jahr 2017 vorläufig 393340 verunglückte Personen aus. Davon verunglückten 3186 tödlich. Betrachtet man die Entwicklung dieser Zahlen, beginnend im Jahr 1970, ist damit ein deutlicher Rückgang verunglückter und getöteter Personen im Straßenverkehr zu verzeichnen. Wird dagegen die Anzahl polizeilich gemeldeter Verkehrsunfälle betrachtet, ist festzustellen, dass diese im Jahr 2017 mit ca. 2,6 Millionen einen neuen Höchststand erreicht hat. Weiterhin bleibt festzustellen, der Fahrzeugbestand in Deutschland wächst kontinuierlich. Waren es am Stichtag zum 01.01.2010 noch 45,7 Millionen Kraftfahrzeuge, so waren es am 01.01.2018 bereits 56,4 Millionen Kraftfahrzeuge [3, S. 92].

Damit zeigt sich ein Rückgang im Straßenverkehr verunglückter Personen, obwohl die Anzahl der Verkehrsteilnehmer im gleichen Zeitraum gestiegen ist. Ein Grund für den Rückgang der schwer verletzten und getöteten Personen ist u.a. die kontinuierliche Weiterentwicklung der Sicherheitssysteme in den Fahrzeugen. Prägnante Beispiele für Sicherheitssysteme sind die Knautschzone, der Airbag und der Sicherheitsgurt [4, S. 36 f.]. Durch den Sicherheitsgurt allein können bereits potentiell tödliche Verletzungen verhindert werden. Er zählt zu den zentralen Sicherheitselementen im Fahrzeug und rettete Schätzungen zu Folge bereits über eine Million Menschenleben [4, S. 3]. Neben diesen und weiteren passiven Sicherheitssystemen

verfügen moderne Fahrzeuge über eine Reihe weiterer aktiver Sicherheitssysteme wie z.B. das Antiblockiersystem, das Elektronische Stabilitäts-Programm oder eine aktive Bremsunterstützung. Während passive Systeme Verletzungen im Falle eines Unfalls vermeiden helfen, sollen aktive Sicherheitssysteme bereits das Entstehen von Unfällen vermeiden [5, S. 11]. Zieht man in Betracht, dass moderne Fahrzeuge über deutlich mehr Sicherheitssysteme als die bereits genannten verfügen, wird die Entwicklung, der verringerten Anzahl verunglückter Personen im Straßenverkehr bei gleichzeitiger Zunahme der Verkehrsteilnehmer, plausibel.

Diese Vielzahl an technischen Lösungen können aber nur bedingt zur Sicherheit beitragen. Einen entscheidenden Beitrag zur Sicherheit muss der Fahrer selbst leisten. So haben die Fähigkeiten eines Fahrers und seine Befindlichkeiten einen wesentlichen Einfluss auf die sichere Teilnahme am Straßenverkehr [5, S. 10]. Bei Unfällen mit Personenschaden ist die überwiegende Unfallursache auf das Fehlverhalten von Personen zurückzuführen, wobei eine nicht angepasste Geschwindigkeit die Hauptunfallursache ist [5, S. 13].

Eine unangepasste Geschwindigkeit kann u.a. aus einer Ablenkung heraus entstehen. So z.B. bei der Bedienung eines Navigationssystems, bei der Verwendung eines Smartphones oder durch weitere Insassen. Ist der Fahrer in diesen Momenten abgelenkt, ist dieser womöglich nicht mehr in der Lage auf sich ändernde Situationen angemessen reagieren zu können. Ein weitere Ursache für eine unangepasste Geschwindigkeit ist die Fehleinschätzung von Fahrern. Mögliche Ursachen können u.a. sein: Selbstüberschätzung der eigenen Fähigkeiten, das Fahrzeug steuern zu können oder auch mangelnde Kenntnisse über physikalische Zusammenhänge. Hinzu kommen das Fahren unter Einfluss von Medikamenten, Alkohol oder illegalen Drogen. All diese Ursachen können zu einer unangepassten Geschwindigkeit oder einer unangepassten Fahrweise führen.

Möchte man den Fahrer nun weiter unterstützen, mit dem Ziel Unfälle im Allgemeinen zu vermeiden, wäre ein denkbarer Ansatz die Verwendung eines Fahrerassistenzsystems, welches den Fahrer auf ein gestiegenes Risiko während der Fahrt hinweist. Da die erwünschte angepasste Geschwindigkeit auch auf selbigen Straßenabschnitten nicht konstant sein muss und nicht mit der erlaubten Höchstgeschwindigkeit zusammenfallen muss, scheint die alleinige Verwendung der Geschwindigkeit als Indikator für ein Risiko behaftetes Fahren als ungeeignet. Vielmehr könnte solch ein Fahrerassistenzsystem während einer Fahrt Anhaltspunkte im Fahrverhalten suchen, die auf ein gesteigertes Risiko hindeuten. Eine Möglichkeit bestünde darin das aktuelle Fahrverhalten des Fahrers mit bekanntem, unfallfreiem Fahrverhalten abzugleichen. Eine weitere Möglichkeit bestünde in der Suche nach Anzeichen für Ablenkung beim Fahren.

Solch ein Assistenzsystem könnte den Fahrer, dessen Fahrt aktuell analysiert wird, auf ein gestiegenes Risiko hinweisen, welches aus seinen aktuellen Fahrverhalten resultiert. Der Fahrer könnte daraufhin seine Aufmerksamkeit wieder auf den Straßenverkehr und sein Verhalten lenken, sein Fahrverhalten neu bewerten und den aktuellen Bedingungen im Straßenverkehr anpassen. Dies wäre in Situationen hilfreich, in denen der Fahrer eine Situation falsch einschätzt oder durch Ablenkung sein Fahrverhalten nicht der aktuellen Situation anpasst. Der Fahrer ist u.U. gedanklich nicht beim Fahren, sondern beschäftigt sich mit anderen Dingen, steht unter Stress

oder ist wütend und beurteilt die Verkehrssituation anders als sonst. Ein einfaches Beispiel ist die Fahrt zur Arbeit, die zwei regelmäßige Zwischenstopps am Kindergarten und Schule enthält. Diese Fahrt findet regelmäßig statt und ist alltägliche Routine, sodass sich hier bereits Routinefehler einschleichen können. Wird diese Routine, die es ermöglicht pünktlich zum Schulstart und Arbeitsbeginn vor Ort sein zu können, gestört, wird der Fahrer versuchen dies zu kompensieren. Eventuell sogar dann, wenn dies nicht notwendig ist. Findet der Fahrtbeginn z.B. verspätet statt, wird der Fahrer wahrscheinlich dazu neigen schneller zu fahren. Fährt er sonst zu Gunsten der Sicherheit eher 5 $\frac{\text{km}}{\text{h}}$ langsamer als notwendig, sind es jetzt womöglich 5 $\frac{\text{km}}{\text{h}}$ mehr als erlaubt. Da der Fahrer aber eine Verspätung befürchtet, bewertet er das Verhalten als angemessen und geht das erhöhte Unfallrisiko ein. Dabei steigt gerade in dieser vermeintlich kleinen Geschwindigkeitssteigerung die Todeswahrscheinlichkeit für Fußgänger im Falle eines Zusammenstoßes deutlich. Für Fußgänger zwischen 15 und 59 Jahren beträgt diese bei 45 $\frac{\text{km}}{\text{h}}$ noch 4,7%. Bei einem Zusammenstoß mit 55 $\frac{\text{km}}{\text{h}}$ sind es bereits 15,2% [6].

Eine weitere Möglichkeit der Verwendung solch eines Systems wäre die zusätzliche Bereitstellung und Auswertung der Informationen zum Fahrverhalten durch eine weitere Instanz. Dies könnten z.B. Versicherer sein. Risikobewusste Fahrer könnten auf diese Weise belohnt werden, wenn sie unnötige Risiken beim Fahren vermeiden. Aktuelle Ansätze dazu werden von Kfz-Versicherern bereits in Form von Telematik-Tarifen angeboten [7].

1.2 Ziel dieser Arbeit

Damit ein System feststellen kann, ob während einer Fahrt ein erhöhtes Risiko besteht, muss es geeignete Parameter messen und auswerten können. Da, wie einleitend bereits festgestellt, die aktuelle Fahrgeschwindigkeit allein ungenügend ist, müssen weitere Parameter herangezogen werden. Hinzu kommt die Randbedingung, dass alle Messungen und Auswertungen ohne Eingriff in das Fahrzeug oder den Fahrer stattfinden müssen. Damit stehen Werte, die das Fahrzeug selbst generiert und über den internen Bus auslesbar wären nicht zur Verfügung. Es sollen auch keine Sensoren am Fahrer angebracht werden die z.B. Puls, Hautleitfähigkeit oder die Augen- sowie Kopfbewegungen des Fahrers registrieren. Mit diesen Einschränkungen soll zunächst erreicht werden, dass keine zusätzlichen Manipulationen am Fahrzeug oder Fahrer notwendig werden und keine ungewohnten und somit störenden Einflüsse den Fahrer ablenken. Um diese Einschränkungen einzuhalten wurde für diese Arbeit der Einsatz eines Smartphones gewählt, welches ebenfalls über eine Auswahl von Sensoren verfügen.

Die daraus resultierende Problematik besteht nun darin, Ablenkung und Befinden des Fahrers nicht direkt messen zu können. Torkkola et al. [8] haben in ihren Experimenten untersucht, inwieweit Ablenkung und Fahrverhalten durch bereits im Fahrzeug vorhandene Sensoren detektiert werden können. Für diesen Zweck untersuchten sie Daten von Standardsensoren und komplexeren Sensoren, die in Systemen zur Kollisionsvermeidung verwendet werden. Standard-Sensoren erfassen u.a. den Winkel des Lenkradeinschlags oder die Betätigung des Gaspedals. Komplexere Sensoren zur Kollisionsvermeidung erkennen u.a. die Begrenzungen der eigenen

Fahrbahn. In diesen Experimenten mussten die Fahrer im Verlauf der Fahrt, die in einem Simulator durchgeführt wurde, zufällig gezeigte Fahrzeuge beobachten und anschließend entsprechende Fragen beantworten. Für jede korrekt beantwortete Frage konnten sie zusätzliche Bonuspunkte erhalten. Die daraus resultierenden Daten wurden für weitere Analysen verwendet. Die Daten der Standardsensoren wurden dabei sowohl allein (z.B. Betätigung des Gaspedals), als auch in Kombination mit den Daten der komplexeren Sensoren zur Kollisionsvermeidung betrachtet. Torkkola et al. zeigten in ihrer Arbeit, dass eine Erkennung im gewissen Rahmen mit diesen Sensoren möglich ist. Die Erkennungsrate von Unaufmerksamkeit mit Sensoren zur Kollisionsvermeidung lag bei 80%, unter ausschließlicher Verwendung von Standardsensoren bei 67%. Betrachtet man lediglich die Standardsensoren, waren die Informationen zum Einschlagwinkel des Lenkrades, gefolgt von denen zum Gaspedal, für die Erkennung ausschlaggebend. Da durch die Daten ein Rückschluss auf den Zustand des Fahrers bzw. sein Fahrverhalten möglich ist, muss ein Fahrmanöver, in dem der Fahrer abgelenkt ist, bzw. ein erhöhtes Risiko eingeht, Merkmale aufweisen, die es als solches identifizierbar machen.

Geht man davon aus, dass während einer Fahrt ein entsprechender Stellwinkel des Lenkrades zu einer Auslenkung des Fahrzeugs führt, werden sich auch die auf das Fahrzeug einwirkenden Kräfte ändern. Dies wird beim Einfahren in eine Kurve deutlich, bei der die seitlich wirkenden Kraftvektoren ihren Betrag ändern. Dieser Zusammenhang ist jedoch begrenzt und nicht mehr in Situationen anwendbar, in denen das Fahrzeug nicht entsprechend reagiert. So z.B. auf Glatteis. In diesem Fall würde die Aktion des Fahrers (Änderung des Lenkradwinkels) nicht registriert werden. Die Betätigung des Gaspedals resultiert schließlich ebenfalls in der Änderung der einwirkenden Kräfte. Dabei kann das Fahrzeug beschleunigt werden, oder es wird eine konstante Geschwindigkeit gehalten, bei der die Beschleunigung durch das Gaspedal die einwirkenden Bremskräfte (negative Beschleunigung z.B. durch Luftwiederstand und Rollreibung) gerade ausgleicht. Auch hier sind den Zusammenhängen Grenzen gesetzt. Resultiert z.B. die Betätigung des Gaspedals in einem Schlupf der Räder, wird das Fahrzeug nur wenig oder gar nicht beschleunigt. Durch die fehlende Traktion kann damit auch in diesem Fall nicht auf das eigentliche Fahrverhalten geschlossen werden. Unter diesen Annahmen und Berücksichtigung der Grenzen kann geschlussfolgert werden, dass durch die Messung der Geschwindigkeit und Trägheitskräfte auf die Bedienung des Lenkrads und des Gaspedals geschlossen werden kann. Mit einem Smartphone lassen sich beide Messungen realisieren. Die Trägheitskräfte können mit einem G-Sensor direkt und die Geschwindigkeit über die Auswertung des GPS-Signals indirekt gemessen werden.

Ein weiterer Aspekt, der für die Verwendung eines Smartphones spricht, ist neben der sensorischen Ausstattung die zur Verfügung stehende Rechenleistung. Damit bietet sich die Möglichkeit mit Hilfe moderner Technologien komplexe Auswertungen direkt auf dem Smartphone durchführen zu können. So steht z.B. mit TensorFlow ein Framework zur Verfügung, welches es erlaubt, Anwendungen mit künstlichen Neuronalen Netzen selbst auf mobilen Endgeräte, wie Smartphones, implementieren [9] zu können.

Durch die einleitend dargestellte Problematik allein besteht bereits ein grundlegendes Motiv, sich mit möglichen Lösungen auseinanderzusetzen, wie es Fahrerassis-

tenzsysteme sein könnten. Und mit den nun zur Verfügung stehenden Möglichkeiten, Fahrverhalten messen und Smartphones mit entsprechender Sensorik einsetzen zu können, können grundlegende Fragen zu möglichen Lösungsansätzen untersucht werden. Um einen Fahrer auf ein erhöhtes Risiko hinweisen zu können, muss sein Fahrverhalten während der Fahrt analysiert werden können. Es muss in den Messungen, die während der Fahrt stattfinden, nach Merkmalen gesucht werden, die als Indikator für ein erhöhtes Risiko dienen. Ebenfalls denkbar ist der Vergleich ausgewählter Manöver, bei dem die aktuellen Messdaten mit denen verglichen werden, die ein risikobehaftetes Fahren repräsentieren.

Das Ziel dieser Arbeit ist es, zu klären, ob Fahrmanöver, die mit einem Smartphone sensorisch erfasst wurden, mit Hilfe rekurrenter neuronaler Netze grundsätzlich klassifiziert werden können. Das künstliche neuronale Netz muss eine Messreihe, die ein Fahrmanöver repräsentiert, einer bestimmten Klasse von Fahrmanövern zuordnen können. Dieser Fragestellung wird experimentell nachgegangen. Zwischenziele, die im Rahmen dieser Arbeit realisiert wurden, sind daher die Entwicklung einer Smartphone-App, die während der Fahrmanöver geeignete Messreihen aufzeichnen kann, das Erfassen und Aufbereiten von Daten zu Fahrmanövern und die Implementierung eines Klassifikators unter Verwendung künstlicher neuronaler Netze.

Gelingt eine erfolgreiche Klassifikation, wird das Ziel als erreicht erachtet. Ist eine Klassifikation nicht möglich, kann die Frage im Rahmen dieser Arbeit nicht abschließend beantwortet werden. Dies liegt in der Tatsache begründet, dass dieses Experiment mit einem konkret implementierten Klassifikator und einer bestimmten Art von neuronalem Netz keine generelle Aussage über die Technologie im ganzen treffen kann. So könnte im Falle eines Scheiterns eine alternative Implementierung oder Verwendung anderer neuronaler Netze, eine Klassifikation ermöglichen.

1.3 Verwandte Arbeiten

Einleitend wurde bereits in Abschnitt 1.2 auf die Arbeit von Torkkola et al. [8] eingegangen, in der in Simulationen untersucht wurde welche Möglichkeiten der Messungen bestehen, um Ablenkung beim Fahrer feststellen zu können. Aus dieser Arbeit geht hervor, dass Daten zur Betätigung des Gaspedals und Lenkrades wichtige Informationen über den Fahrer liefern können. Ausgehend von diesen Erkenntnissen wurden die Fahrmanöver, die später in dieser Arbeit erläutert werden, festgelegt, die im Rahmen dieser Arbeit aufgenommen und weiter betrachtet werden.

In [10] stellen Hong et al. ein System zur Bewertung des Fahrverhaltens vor, um aggressives Fahrverhalten detektieren zu können. Hauptbestandteil dieses Systems ist ebenfalls ein Smartphone, welches Daten über die internen Sensoren (GPS-Sensor, G-Sensor, Kompass, etc.) aufzeichnet. Zusätzlich werden jedoch auch Daten vom fahrzeugeigenem Diagnosesystem (OBD2) und einer inertialen Messeinheit (IMU) aufgezeichnet. Zur Beurteilung des Fahrverhaltens und zur Erstellung eines Fahrverhaltensmodells wurden Sensordaten zu bestimmten Fahrmanövern ausgewertet. So z.B. beim Anfahren, Stoppen, schneller Fahrt (schneller als 50 km/h) und Kurven. Mit den Informationen aus diesen Daten, z.B. der Höchstgeschwindigkeit, Längs- und Querbeschleunigung wurde wiederum ein Merkmalsvektor (feature vector) entwickelt. Dieser bildet die Grundlage für ein Fahrermodell und dient der Analyse,

ob es sich um einen aggressiven Fahrer handelt. In eigenen Experimenten konnten sie mit einer Genauigkeit von bis zu 90% aggressives Fahrverhalten identifizieren. Johnson et al. [11] verwenden ebenfalls ein Smartphone um Sensordaten zu erfassen. Um ein aggressives Fahrverhalten zu detektieren verwenden sie einen Algorithmus zur dynamischen Zeitnormierung (DTW, dynamic time warping). You et al. [12] verwenden Sensordaten eines Smartphones, zusätzlich jedoch auch beide Kameras (Vorder- und Rückseite) um gefährliche Situationen während der Fahrt detektieren zu können. Vaiana et al. [13] verwenden lediglich Beschleunigungsdaten, die sie mit einem Smartphone aufzeichnen. Die Auswertung erfolgt mit einem G-G-Diagramm, in welchem die längs und quer wirkenden Beschleunigung eingetragen werden. Befinden sich die Messpunkte innerhalb eines zuvor definierten Bereichs, ist dies ein Indiz für sicheres Fahren. Meseguer et al. [14] stellen eine Anwendung zur Analyse des Fahrverhaltens vor, in der sie die Sensordaten vom Fahrzeug und Smartphone mittels Datamining und einem künstlichen neuronalen Netz, bestehend aus mindestens drei Schichten, auswerten.

Keine der vorgestellten Arbeiten verwendet zur Analyse der Sensordaten, die häufig mit Daten die direkt aus dem Fahrzeug stammen angereichert wurden, ein rekurrentes neuronales Netz.

Die Analyse der Messreihen, bei denen es sich um Zeitreihen von Sensordaten handelt, soll mittels Zeitreihenklassifikation erfolgen. Zur Klassifikation von Zeitreihen existiert eine Vielzahl von Ansätzen [15] mit unterschiedlich leistungsfähigen Algorithmen. Da im Zentrum des Interesses dieser Arbeit jedoch die Klassifikation von Zeitreihen mit Hilfe rekurrenter neuronale Netze steht, wurde für die vorliegende Arbeit der von Jaeger vorgestellte Klassifikator [16, S. 74 bis 80] verwendet. Dieser basiert auf Konzeptoren und rekurrenten neuronalen Netzen, verfügt über eine Leistungsfähigkeit, die dem Stand der Technik entspricht [16, S. 74] und konnte bereits zur Klassifikation von Zeitreihen in verschiedenen Bereichen erfolgreich eingesetzt werden. Jaeger beschreibt die erfolgreiche Klassifikation japanischer Vokale und die Erkennung einzelner Sprecher [16, S. 74], wohingegen Bao et al. dieses Konzept zur Erkennung von Bewegungsabläufen einsetzen [17]. Gast et al. entwickeln ein Modell, welches komplexe sensorische Stimulationen lernen und erkennen kann [18].

1.4 Abgrenzung der Klassifikationsaufgabe

Einige, u.a. in Abschnitt 1.3 vorgestellte, Arbeiten beschäftigen sich bereits mit der Erkennung von aggressivem Fahrverhalten und Möglichkeiten Assistenzsysteme zu entwickeln. Auch in Abschnitt 1.1 wurde ein Fahrerassistenzsystem vorgeschlagen, um so dem Fahrer Rückmeldungen über sein Fahrverhalten geben zu können. Die Entwicklung solch eines Assistenzsystem ist zunächst als Vision zu sehen und ist nicht Bestandteil dieser Arbeit. Ebenso ist die Beurteilung des Fahrverhaltens nicht Bestandteil dieser Arbeit. Demnach wird in den Messreihen und Fahrmanövern dieser Arbeit nicht nach Indikatoren besonders risikobehafteten oder aggressiven Fahrens gesucht. Auch der implementierte Klassifikator wird nicht auf solche Kriterien hin implementiert und evaluiert werden. Die Anforderungen, die an den Klassifikator im Rahmen dieser Arbeit gestellt werden, bestehen vielmehr darin, einzelne, voneinander verschiedene Fahrmanöver erkennen und einer Klasse von Fahrmanövern zuordnen zu können.

1.5 Aufbau dieser Arbeit

Zunächst werden in Kapitel 2 die theoretischen Grundlagen zu künstlichen neuronalen Netzen vorgestellt. Beginnend bei den natürlichen neuronalen Netzen, von denen auf die künstlichen neuronalen Netze geschlossen wird, werden grundlegende Funktionen der Neuronen erläutert. Weiter wird erläutert, wie Neuronen in Netzen funktionieren. Weiterführend werden spezielle künstliche neuronale Netze vorgestellt, wobei der Schwerpunkt auf die rekurrenten neuronalen Netze liegt.

Die Anforderungen, die an die Datenerhebung gestellt wurden, werden in Kapitel 3 erläutert. Es werden die Fahrmanöver beschrieben, die mit dem Smartphone erfasst werden sollen und Apps analysiert, mit denen Sensordaten aufgezeichnet werden können. Die vom Autor entwickelte App wird vorgestellt, sowie der experimentelle Aufbau im Fahrzeug zur Datenerfassung beschrieben.

Das Kapitel 4 beschreibt den Aufbau und die Funktion des implementierten Klassifikators und wie Fahrmanöver erkannt und zugeordnet werden können. Dabei wird die konkrete Implementierung, die im Rahmen dieser Arbeit in Octave realisiert wurde, detailliert erläutert. Die Durchführung von Experimenten, die automatisiert werden können und für diese Arbeit implementiert wurden, wird ebenfalls beschrieben.

Die Ergebnisse der Klassifikationsaufgabe werden in Kapitel 5 vorgestellt. Zunächst wurde experimentell ermittelt, wie gut die einzelnen Messreihen den Fahrmanövern zugeordnet werden können. Dabei wurde zunächst die Anzahl der Neuronen variiert, um eine eventuelle Auswirkung der Netzwerkgröße zu ermitteln. Anschließend wurde die Anzahl der Messreihen variiert, die für das Training der neuronalen Netze verwendet wurden. Abschließend wurde untersucht, wie der Klassifikator mit Messreihen umgeht, die einer unbekannten Klasse (Fahrmanöver) angehören.

In Kapitel 6 werden ausgewählte Faktoren des Klassifikators modifiziert, um deren Auswirkungen zu ermitteln. Zum einen wird die Datenaufbereitung betrachtet und welchen Einfluss diese auf eine richtige Klassifikation hat. Weiterhin wird die Aktivierungsfunktion modifiziert, um auch hier die Auswirkungen zu ermitteln.

Abschließend werden in Kapitel 7 die Ergebnisse in ihrer Gesamtheit betrachtet und diskutiert, inwiefern das Ziel dieser Arbeit erreicht wurde. Mögliche Auswirkungen, die eine Einführung solcher Systeme haben könnten, werden betrachtet. Weiterführende Arbeiten erläutert, sowie Fragestellungen ergänzt, die im Rahmen dieser Arbeit nicht oder unzulänglich bearbeitet werden konnten.

2. Theoretische Grundlagen

In diesem Kapitel werden notwendige theoretische Grundlagen eingeführt. Es wird zunächst erklärt was ein Neuron und ein neuronales Netz ist. Zunächst ausgehend vom biologischem Vorbild aus der Natur, der Nervenzelle, bis hin zum künstlichen Neuron und künstlichen Netzen. Da die Anwendungsbereiche für neuronale Netze sehr vielfältig sind und deren theoretischen Grundlagen sowie praktischen Umsetzungen ein breites Spektrum an Möglichkeiten abdecken, werden nur elementare Grundlagen behandelt. Der Schwerpunkt dieses Kapitels liegt in der Betrachtung der rekurrenten neuronalen Netze, da sich diese besonders für die Analyse von Daten eignen, die in Zeitreihen vorliegen [1].

2.1 Das Neuron

Nervenzellen (Neuronen), die in ihrer Gesamtheit durch vielfache Verknüpfungen schließlich komplexe Netze bilden, sind bereits aus der Natur bekannt. Die mögliche Komplexität zeigt sich am Beispiel des menschlichen Gehirns, welches Teil des zentralen Nervensystems ist. Ein menschliches Gehirn besteht aus etwa 86 Milliarden Nervenzellen [19, S. 7], dabei kann eine Nervenzelle allein mit bis zu 100.000 weiteren Zellen verbunden sein [20, S. 82].

2.1.1 Die Nervenzelle aus der Natur als Vorbild

Eine Nervenzelle, wie sie in Abbildung 2.1 dargestellt ist, besteht aus einem Zellkörper mit Zellkern und Neuriten, welche die Ausläufer am Zellkörper bilden [20, S. 82]. Neuriten, die wiederum in Dendriten und Axone unterteilt werden, sind die Verbindungsstellen zu weiteren Zellen. Über die Dendriten werden Signale von anderen Zellen empfangen. Eine Nervenzelle kann, wie in Abbildung 2.1 dargestellt, mehrere Dendriten ausbilden, die dann verästeln können und mit den sendenden Zellen synaptische Verbindungen eingehen. Dagegen hat eine Nervenzelle immer nur ein Axon, welches dann allerdings ebenfalls verzweigt und mit anderen Zellen synaptische Verbindungen eingeht. Über das Axon werden Impulse wiederum von der Nervenzelle selbst an empfangende Zellen gesendet [20, S. 94].

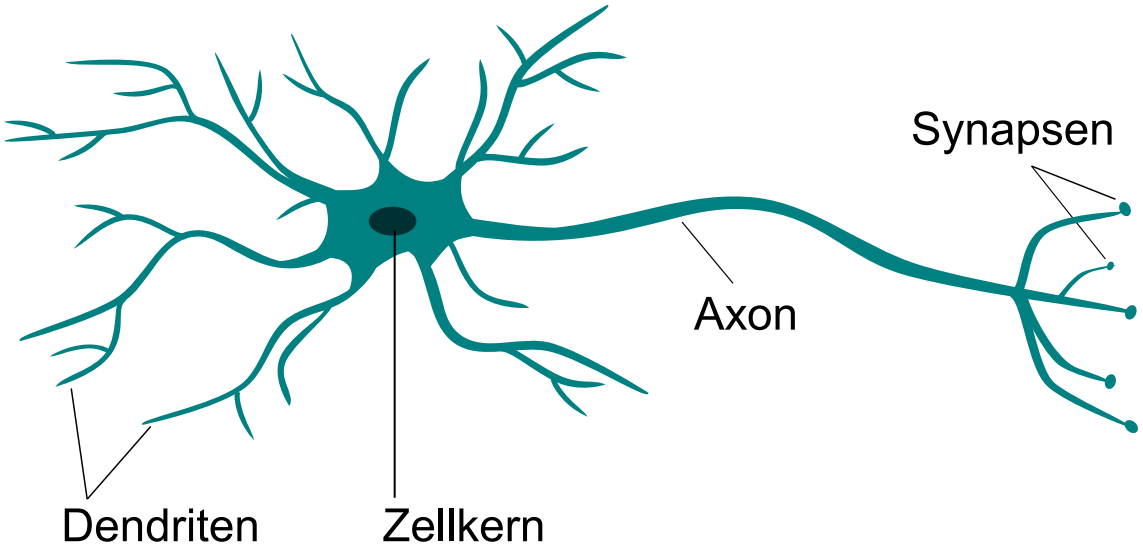


Abbildung 2.1: Neuron, Quelle: Eigene Darstellung nach [20, S. 82]

Die Übertragung von Informationen über die Nervenzelle wird in Form von Nervenimpulsen realisiert, die wiederum selbst Aktionspotentiale an einer Nervenfaser sind. Bei diesem Vorgang wird die Membran an einer Stelle für positiv geladene Ionen durchlässig. Diese Ionen gelangen in das, zumeist negativ geladene, Innere der Zelle und bewirken so eine kurzfristige und lokale Umkehr des elektrischen Feldes. Dies regt wiederum die Nachbarregion an, bei der die Membran nun ebenfalls positiv geladene Ionen durchlässt. Ionenpumpen sorgen anschließend für die Wiederherstellung des ursprünglichen elektrischen Feldes. Weiterhin gibt es Mechanismen die verhindern, dass eine kurz zuvor erregte Region erneut angeregt werden kann. Der Nervenimpuls läuft so entlang der Nervenfaser immer nur in eine Richtung. Diese robuste Form der Impulsübertragung ist binär und kann eine maximale Frequenz von 500 Hz erreichen. Erzeugt wird der Nervenimpuls am Axonhügel, dem Beginn des Axons. Der einmal erzeugte Nervenimpuls kann sich über die Verzweigungen des Axons ausbreiten und sorgt in den Synapsen für das Ausschütten von Neurotransmittern. Rezeptoren der empfangenden Zelle binden diese Neurotransmitter und lösen so einen weiteren Nervenimpuls aus [20, S. 108 bis 115].

2.1.2 Mathematisches Modell zur Nervenzelle

Zur Modellierung und Simulation künstlicher neuronaler Netze mit Hilfe von Computern wird zunächst ein mathematisches Modell [21, S. 268 bis 271] benötigt, welches die Funktionen des Neurons möglichst genau abbildet. Der Übergang vom natürlichen Neuron zum mathematischen Modell ist abstrahiert in Abbildung 2.2 dargestellt. Für die Realisierung mit Computern wird zunächst aus dem kontinuierlichen Zeitbereich in den diskreten Zeitbereich gewechselt.

Natürliche Neuronen sind über Synapsen miteinander verbunden, über die sie ein Signal x weitergeben können. Die Stärke des ausgelösten Aktionspotentials an den Dendriten des empfangenden Neurons ist abhängig von der synaptischen Verknüpfung. Je mehr Synapsen zwei Neuronen verbinden, desto stärker, also schwerer gewichtet, ist das Aktionspotential. Um dies auszudrücken wird das Signal x mit einem

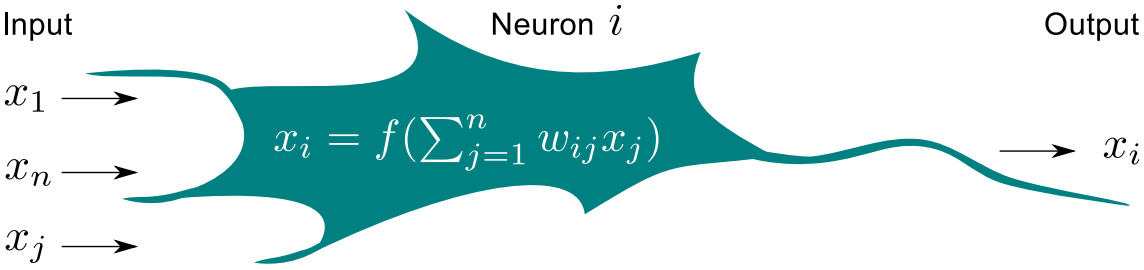


Abbildung 2.2: Formales Neuron, Quelle: Eigene Darstellung nach [21, S. 269] sowie [20, S. 82]

spezifischen Gewicht w beaufschlagt. Zu jeder Verbindung zwischen Neuronen existiert so ein Signal x_j mit einem dazugehörendem Gewicht w_{ij}. Existiert zwischen zwei Neuronen keine Verbindung ist in diesem mathematischen Modell das entsprechende Gewicht null. Natürliche Neuronen verarbeiten die eingehenden Signale um bei Bedarf wiederum selbst ein entsprechendes Signal am Axonhügel zu generieren. Im Modell wird zu diesem Zweck zunächst die gewichtete Summe berechnet, die schließlich über eine Aktivierungsfunktion ausgewertet wird [21, S. 269]. Damit erhält man für die Ausgabe an das empfangende Neuron Formel 2.1.

$$x_i = f(\sum_{j=1}^{n} w_{ij}x_j) \tag{2.1}$$

Die Aktivierungsfunktion hat einen relevanten Einfluss auf die neuronale Aktivität. Beispiele für Aktivierungsfunktionen sind die Heaviside-Funktion (Formel 2.2), die logistische Funktion (Formel 2.3), der Tangens hyperbolicus und die Sinus-Funktion wie in Abbildung 2.3 dargestellt. So kann über diese Funktion beeinflusst werden wie sensitiv ein Neuron ist, also ob bereits kleine Werte zu einer Aktivierung führen können. Mit ihr kann aber ebenso die neuronale Dynamik beschränkt werden. Wird im einfachsten Fall die Identität mit $f(x) = x$ als Aktivierungsfunktion verwendet, kann dies in einigen Fällen zu Konvergenzproblemen führen da die Werte unbeschränkt wachsen können [21, S. 269]. Dies kann wiederum zu einem instabilen neuronalen Netz führen.

$$H_\Theta(x) = \begin{cases} 0 & \text{falls } x < \Theta \\ 1 & \text{sonst} \end{cases} \tag{2.2}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

2.2 Neuronale Netze

Werden Neuronen miteinander verbunden, entstehen neuronale Netze. Innerhalb dieses Netzes übernehmen Neuronen verschiedenste Aufgaben. Die Eingabeneuronen dienen zur Informationsaufnahme, durch sie wird es möglich dem Netz Informationen zuführen zu können. Ein Beispiel aus der Natur ist eine Sinneszelle wie

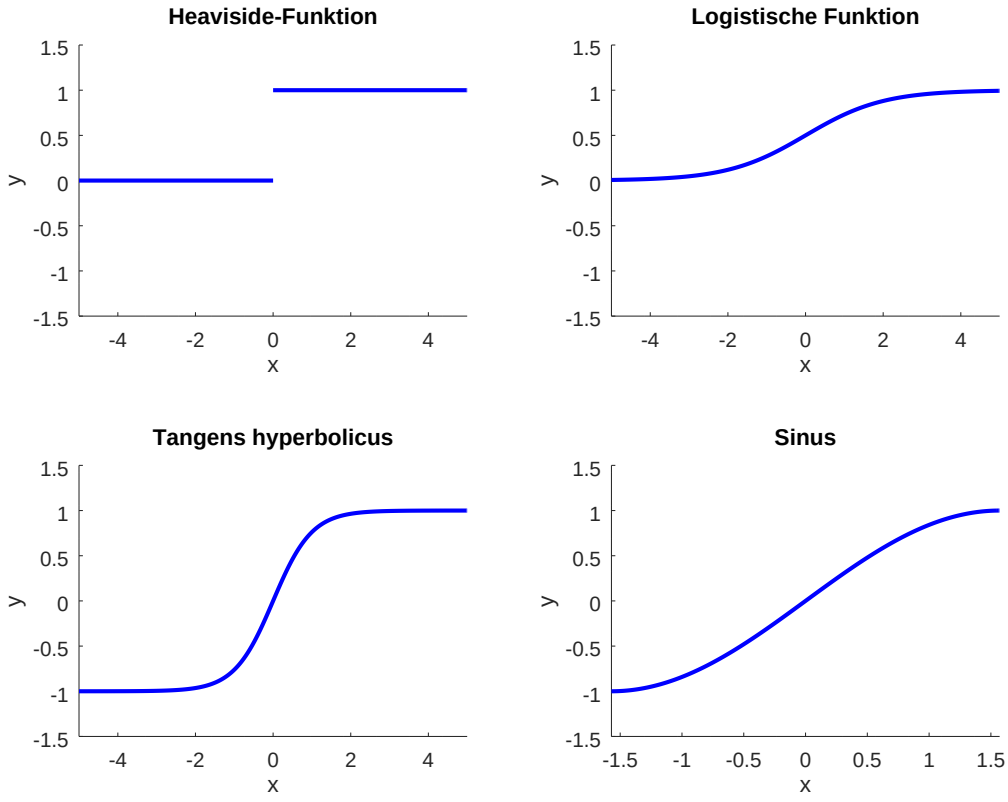


Abbildung 2.3: Aktivierungsfunktionen, Quelle: Eigene Darstellung

sie u.a. im Auge vorkommt. Durch äußere Reize, im konkreten Beispiel das Auftreffen von Licht auf die Netzhaut, werden die dortigen Sinneszellen erregt und lösen Aktionspotentiale aus. Dieser Impuls wird schließlich über den Sehnerv an das Gehirn weitergereicht [20, S. 148 f.]. Neuronen, die Informationen aufnehmen werden der Eingabeschicht eines Netzes zugeordnet. Die eigentliche Verarbeitung erfolgt anschließend im Gehirn. Das Gesehene könnte weiter analysiert werden, was schließlich zu einer Entscheidung und einer Reaktion führen kann. Dies geschieht mit Neuronen im Gehirn, die ausschließlich Impulse des Nervensystems verarbeiten. Sie sind nicht dafür ausgelegt mit ihrer Umwelt in Interaktion zu treten und werden der verdeckten Schicht zugeordnet. Schließlich kann ein neuronales Netz über eine Ausgabeschicht verfügen. Neuronen dieser Schicht geben die verarbeiteten Informationen wieder aus. Im aktuellen Beispiel könnten die eingefallenen Lichtstrahlen von einem Buch herrühren und das Ende einer gelesenen Seite zeigen. Im Gehirn wird die Entscheidung zum umblättern getroffen. Über die Aktivierung der komplexen Bewegungsprogramme im Gehirn gelangen die Impulse schließlich mit Hilfe von Neuronen der Ausgabeschicht und deren motorischen Endplatte zu den Muskeln, um die Aktion ausführen zu können [20, S. 184 f.]. Der gesamte Ablauf ist natürlich deutlich komplexer als hier beschrieben, veranschaulicht allerdings die Strukturierung in die genannten Ebenen wie in Abbildung 2.4 gezeigt. Diese Darstellung ist stark vereinfacht um die Funktionsweisen klar herauszustellen und deckt nicht das Spektrum möglicher Verknüpfungen ab. So können prinzipiell Neuronen der Eingabeschicht mit Neuronen der Ausgabeschicht direkt verbunden sein. Es ist ebenso möglich,

dass ein Neuron sowohl zu Eingabe- als auch zur Ausgabeschicht gehört. Ebenso sind mehrere verdeckte Schichten möglich, die hintereinander geschaltet sind.

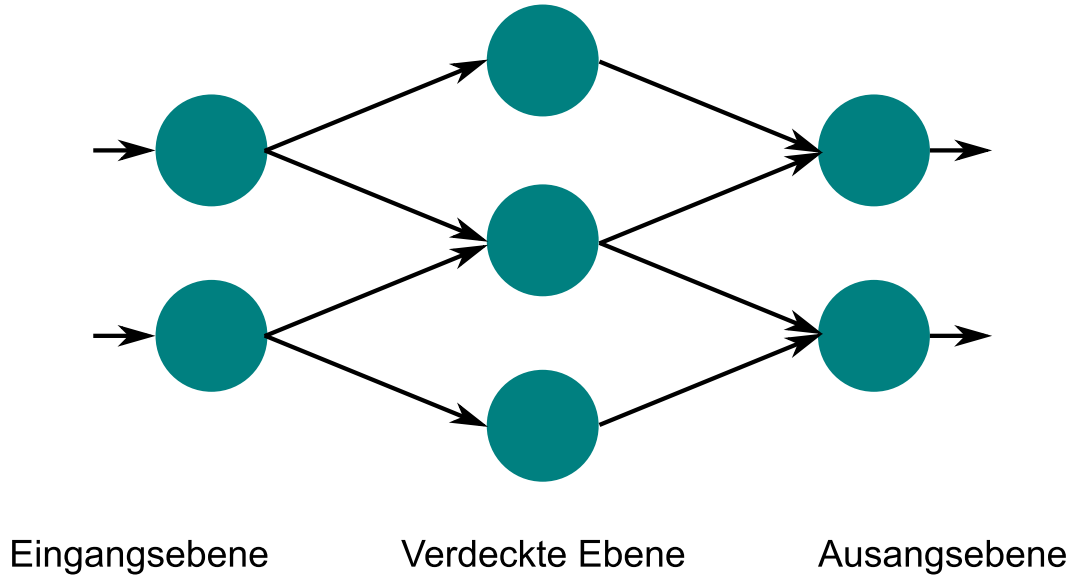


Abbildung 2.4: Ebenen neuronaler Netze, Quelle: Eigene Darstellung

Die Art wie die Neuronen miteinander verbunden sind, hat wiederum auch Auswirkungen auf die Eigenschaften des neuronalen Netzes. Es werden dabei vier Verbindungen, wie sie in Abbildung 2.5 dargestellt sind, unterschieden. Im einfachsten Fall besteht zwischen zwei Neuronen nur eine einfache Verbindung. Das Ausgabesignal des Neurons x_j wird mit dem Gewicht w_{ij} beaufschlagt und wird somit zum Eingangssignal des Neurons x_i in der nächsten Schicht. Bei der direkten Rückkopplung wird die Ausgabe das Neurons x_i direkt als Eingabe zurückgeführt. Somit erhält es im konkret dargestellten Fall neben dem eigenen Ausgabewert auch den des vorgeschalteten Neurons. Die Rückkopplung geschieht hier innerhalb einer Schicht. Im Gegensatz dazu führt die indirekte Rückkopplung den Ausgabewert nicht auf dasselbe Neuron zurück, sondern auf eines einer vorgelagerten Schicht. Abhängig von der Anzahl der Schichten im neuronalen Netz ist die Rückkopplung auch über mehr als nur eine Schicht möglich. Die seitliche Rückkopplung betrifft wiederum nur Neuronen derselben Schicht. So erhält das Neuron x_j die Ausgabewerte der Neuronen der vorgelagerten Schicht und den Ausgabewert von Neuron x_i derselben Schicht.

Abhängig von der Art der neuronalen Verknüpfung wird zwischen zwei Netzwerkstrukturen unterschieden. Den vorwärts gerichteten Netzen (Feedforward Neural Network) und rückgekoppelten Netzen (Rekurrent Neural Network, RNN) [22, S. 35]. In einem vorwärts gerichteten Netzwerk existieren lediglich einfache, vorwärts gerichtete Verbindungen, wie es z.B. bei einem Perzeptron der Fall ist. Bei dem von Rosenblatt 1958 [23] vorgestellten, einschichtigem Perzeptron handelt es sich um die einfachste Struktur eines neuronalen Netzes [24, S. 97 bis 103]. In solch einem Perzeptron existiert nur eine Schicht von Eingabeneuronen, die über Gewichte mit einem Ausgabeneuron verbunden sind. Allerdings sind auch mehrschichtige Perzeptronen-Netze (multi layer perceptron, MLP) möglich. In einem RNN existieren dagegen auch verschiedene Rückkopplungen, wobei hier verschiedene Formen von RNNs existieren zwischen denen unterschieden wird. Da sich RNN u.a. für die

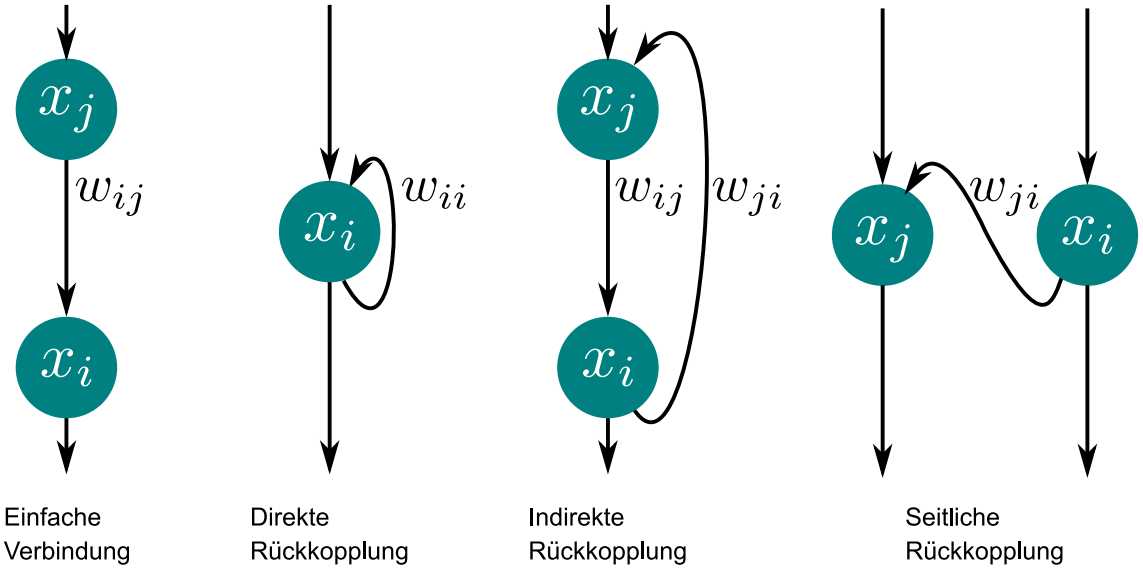


Abbildung 2.5: Neuronale Kopplungen, Quelle: Eigene Darstellung

Analyse von Zeitreihen eigen, wird auf diese Art neuronale Netze in Abschnitt 2.5 gesondert eingegangen. Die Faltungsnetze (convolutional neural networks), die u.a. zur Bild- und Objekterkennung verwendet werden, seien an dieser Stelle nur ergänzend genannt. Im Rahmen dieser Arbeit wird auf diese Netzart nicht weiter eingegangen.

2.3 Mathematische Darstellung von neuronalen Netzen

Die bisher zur Veranschaulichung gewählte Darstellungsform wie in Abbildung 2.4 bietet einige Vorteile. Die Neuronen und deren Verbindungen sind direkt erkennbar. Mit entsprechender Bezeichnung und zusätzlichen Informationen lassen sich diese Netze im geringen Umfang sogar manuell nachvollziehen. Dennoch kann selbst ein kleines Netz mit lediglich vier Neuronen schnell unübersichtlich werden, so zu sehen in Abbildung 2.6. Darüber hinaus ist sie für die maschinelle Verarbeitung ungeeignet. In diesem Kapitel wird daher eine mathematische Darstellungsform vorgestellt, die solch ein Netz beschreibt und sich auch für die Computer-gestützte Auswertung eignet.

Der Skalar N repräsentiert die Anzahl der Neuronen eines Netzes. Jedes Neuron x_i mit $1 \leq i \leq n$ und $n = N$ befindet sich in einem definierten Erregungszustand. Dieser ist abhängig vom initial gewählten Wert und der neuronalen Aktivität, also wie lange das Netz aktiv ist und welche Werte es über die Verknüpfungen von den anderen Neuronen erhält. Werden alle Neuronen-Zustände gemeinsam betrachtet erhält man den Zustand des neuronalen Netzes, dargestellt durch den Zustandsvektor X wie in Formel 2.4. Z.B. ergibt sich der Zustandsvektor X für das in Abbildung 2.6 gezeigte Netz wie in Formel 2.5 gezeigt.

$$X = \begin{pmatrix} x_1 \\ x_i \\ x_n \end{pmatrix} \tag{2.4}$$

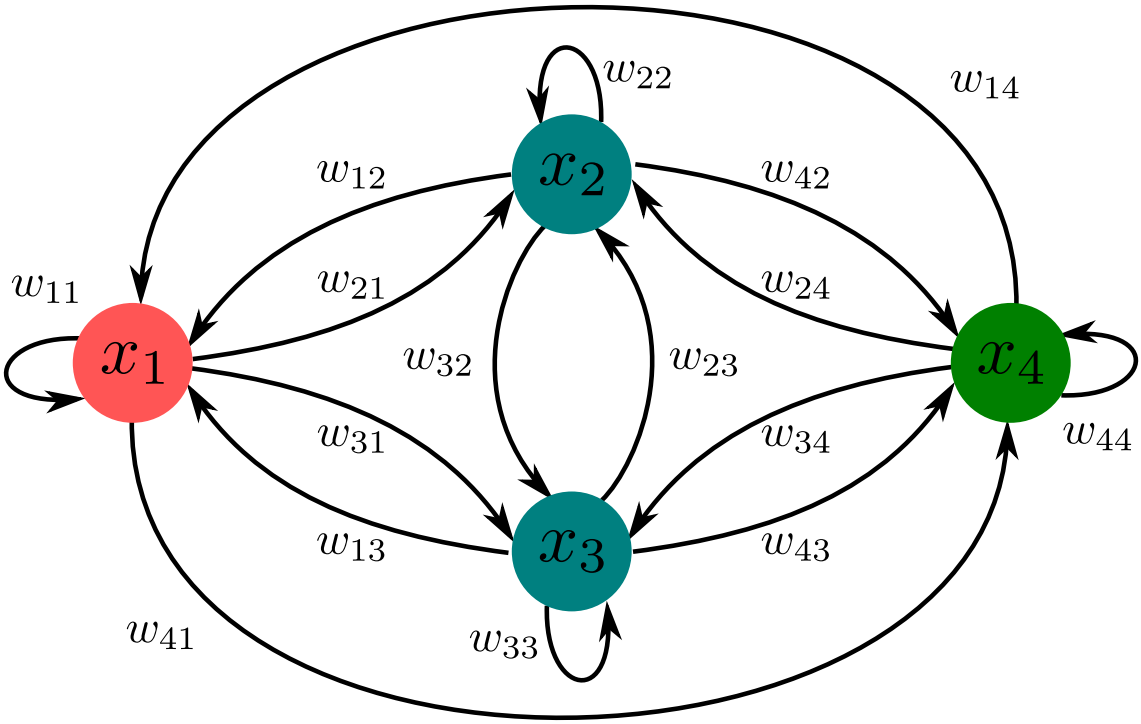


Abbildung 2.6: Neuronales Netz mit vollständiger Vernetzung, Quelle: Eigene Darstellung

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \tag{2.5}$$

Die Gewichte, die die Signalweitergabe zwischen jeweils zwei Neuronen beeinflussen, werden zu einer Gewichtsmatrix W wie in Formel 2.6 gezeigt zusammengefasst. Die Indizes i und j identifizieren das Gewicht zwischen zwei Neuronen eindeutig, wobei $1 \leq i, j \leq n$ und $n = N$ gilt. Diese Darstellungsform erlaubt es zudem markante Gewichte einfach zu identifizieren, vorausgesetzt es werden kontinuierlich in aufsteigender Folge zunächst die Eingabeneuronen, danach die Neuronen der verdeckten Schichten und abschließend die Ausgabeneuronen nummeriert. In diesem Fall sind die Gewichte der Eingabeneuronen in den ersten Spalten der Gewichtsmatrix zu finden (die Anzahl der Spalten entspricht dabei die der Eingabeneuronen). Ähnlich verhält es sich bei den Neuronen der Ausgabeschicht. Gewichte der Verbindungen, die aus der Ausgabeschicht heraus führen, sind, entsprechend der Anzahl der Ausgabeneuronen, in den letzten Spalten zu finden. Die Gewichte der direkten Rückkopplungen befinden sich auf der Hauptdiagonalen von W. Die Gewichte der Verbindungen, die in die Ausgabeschicht hinein führen, also auf die Ausgabeneuronen wirken, ordnen sich in den letzten Zeilen der Gewichtsmatrix an. Bei einem Netz mit einem Eingabe- und einem Ausgabeneuron sind demnach alle Eingabegewichte in der ersten Spalte und alle Ausgabegewichte in der letzten Zeile in W zu finden. Bei zwei Eingabeneuronen entsprechend die ersten zwei Spalten usw. Entsprechend verhält es sich mit der Anzahl der Ausgabeneuronen und der Anzahl Zeilen.

$$W_{Netz} = \begin{pmatrix} w_{i,j} & w_{i,j+1} & w_{i,N} \\ w_{i+1,j} & w_{i+1,j+1} & w_{i+1,N} \\ w_{N,j} & w_{N,j+1} & w_{N,N} \end{pmatrix} \tag{2.6}$$

Für ein vollständig vernetztes neuronales Netz wie in Abbildung 2.6 ergibt sich die Gewichtsmatrix wie in Formel 2.7 gezeigt. Zur Vereinfachung wurden die Kommas zwischen den Indizes weggelassen. Wird im Netz aus Abbildung 2.6 Neuron 1 zum Eingabeneuron und Neuron 4 zum Ausgabeneuron, lässt sich der geschilderte Zusammenhang mit Formel 2.7 leicht nachvollziehen. Ist keine vollständige Vernetzung gewünscht, werden die betreffenden Gewichte zu 0. Somit sind in dieser Darstellung auch unverbundene Neuronen leicht zu identifizieren.

$$W_{Netz} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \tag{2.7}$$

Da bei der vorgestellten Vorgehensweise zum Aufstellen der Gewichtsmatrix die Gewichte der Neuronen der einzelnen Schichten einfach identifiziert werden können, lässt sich die Gewichtsmatrix komfortabel wie in Formel 2.8 zerlegen. Die Gewichte, die den Eingabeneuronen zugeordnet sind, werden in der separaten Matrix W_{in} zusammengefasst. Die der Ausgabeneuronen in W_{out}. Ferner werden die Skalare K und L eingeführt. Der Skalar K steht dabei für die Anzahl der Eingabeneuronen und der Skalar L für die der Ausgabeneuronen. Die Anzahl der verdeckten Neuronen ergibt sich zu $N_{verdeckt} = N - K - L$.

$$W_{Netz} = \begin{pmatrix} \begin{pmatrix} w_{i,j} & \cdots & w_{i,K} \\ \vdots & \ddots & \vdots \\ w_{N-L,j} & \cdots & w_{N-L,K} \end{pmatrix} \begin{pmatrix} w_{i,K+1} & \cdots & w_{i,N} \\ \vdots & \ddots & \vdots \\ w_{N-L,K+1} & \cdots & w_{N-L,N} \end{pmatrix} \\ \begin{pmatrix} w_{N-L+1,j} & \cdots & w_{N-L+1,N} \\ \vdots & \ddots & \vdots \\ w_{N-L+1,j} & \cdots & w_{N,N} \end{pmatrix} \end{pmatrix} \tag{2.8}$$

$$W_{Netz} = \begin{pmatrix} W_{in} \; W_{verdeckt} \\ \\ W_{out} \end{pmatrix}$$

Prinzipiell können Neuronen auch nach einem anderem Schema benannt werden und in einer einmal aufgestellten Gewichtsmatrix Manipulationen wie z.B. ein Zeilentausch vorgenommen werden. Dennoch empfiehlt sich diese Darstellungsform, da die einzelnen Gewichtsmatrizen komfortabel abgeleitet werden können, was im Verlauf der weiteren Anwendung von neuronalen Netzen die Arbeit erheblich erleichtert.

2.4 Lernen

Das Lernen, auch Trainieren, neuronaler Netze wird realisiert, indem die Verbindungen zwischen den Neuronen modifiziert werden und die Anzahl der Neuronen verändert wird. Das Gehirn z.B. ist in der Lage, sich immer wieder auf neue Situationen einzustellen und zu lernen. Eine Möglichkeit des Lernens besteht in der Modifikation synaptischer Verbindungen. Wird eine Synapse häufig aktiviert, registriert sie dies und reagiert darauf mit Wachstum. Sie wird größer und schüttet bei zukünftigen Reizen mehr Neurotransmitter aus. Werden Synapsen dagegen nur ungenügend genutzt, können diese auch wieder verschwinden [20, S. 211]. Dieses Verhalten formulierte D. Hebb bereits 1949 und ist als Hebb'sche Lernregel bekannt. Diese besagt, dass durch eine wiederholte und dauerhafte Signalweitergabe eines Neurons x_j an x_i Veränderungen stattfinden, die in einer Verstärkung der betroffenen Neuronenverbindung resultiert [21, S. 270]. Eine weitere Möglichkeit zu lernen besteht in der Veränderung der Netzstruktur, indem die Anzahl der Neuronen verändert wird. Das Gehirn, selbst das erwachsene, ist in der Lage, mindestens am Hippocampus, regelmäßig neue Neuronen zu bilden [25]. Der Hippocampus spielt bei der Gedächtnisbildung und beim Lernen eine wesentliche Rolle. Künstliche neuronale Netze bei denen im Rahmen der Lernens ebenfalls die Struktur verändert wird, sind die vorhersagenden neuronalen Netze (Abschnitt 2.5.5).

Bei künstlichen neuronalen Netzen werden entsprechend die Gewichte w der neuronalen Verbindungen angepasst. Eines der bekanntesten Lernverfahren ist die Fehler-Rückübertragung (Backpropagation). Hierbei werden zunächst die Eingabewerte des Netzes in einer Aktivierungsphase verarbeitet. Die Ein- und Ausgabewerte des Netzes werden über den negativen Gradientn der summierten quadratischen Fehlerfunktion (Formel 2.9) ausgewertet [21, S. 291 bis 294].

$$E = \frac{1}{2} \sum_{i=1}^{n} (t_i - x_i)^2 \tag{2.9}$$

Dieses Lernverfahren eignet sich vor allem für das Training vorwärts gerichteter Netze. Bei Rekurrenten Neuronalen Netzen kann dieses Verfahren auf Grund der Rückkopplungen nicht direkt angewendet werden. Diese müssen zunächst aufgelöst werden. Erreicht wird dies durch eine Ausfaltung des Netzes in der Zeit (backpropagation through time) [22, S. 151 bis 153]. Der Wert, der über eine Rückkopplung zurückgegeben wird, ist über den zeitlichen Verlauf T hinweg, abhängig von der neuronalen Aktivität, Änderungen unterworfen. Es ist durchaus möglich, dass zu jedem beliebigen Aktivierungszeitpunkt t_n eines Netzes ein anderer Wert rückgekoppelt wird. Wird eine Rückkopplung in Zeit aufgefaltet ist diese, wie in Abbildung 2.7 dargestellt, von der Anzahl n der Aktivierungen des Netzes abhängig. Aus einem Netz, bestehend aus einem Neuron A und einer direkten Rückkopplung, entsteht für lediglich $n = 3$ Aktivierungen ein Netz aus drei Neuronen A_n. Das erste Neuron A_1 erhält zwei Eingabewerte. Der erste $x(t_0)$ repräsentiert den ersten Wert des Eingangssignals. Für den Fall, dass der Initialwert von A_1 ungleich 0 ist, erhält es auch den Wert $y(t_0)$, der zum Zeitpunkt vor der ersten Aktivierung bereits auch der Ausgabewert von A_1 war und somit die Rückkopplung ersetzt. Nach der ersten Aktivierung des Netzes wurde durch A_1 der Ausgabewert $y(t_1)$ berechnet. Dieser

steht jetzt dem Neuron A_2 als Eingabewert zur Verfügung und ersetzt ebenfalls die ursprüngliche Rückkopplung. Zusätzlich erhält A_2 den zweiten Wert des Eingangssignals $x(t_1)$. Entsprechend fortgesetzt für das Neuron A_3 steht schließlich der Wert $y(t_3)$ zur Verfügung. Auf dieses ausgefaltete Netz kann nun auch das Verfahren der Fehler-Rückübertragung angewendet wird. Zu beachten ist allerdings, dass es sich bei dem ursprünglichen Netz nur um ein Gewicht w von nur einer Rückkopplung handelt. Im ausgefalteten Netz wird dieses durch drei Gewichte w_n repräsentiert. Die Fehler-Rückwirkung muss daher zunächst vollständig durchgeführt und alle berechneten Änderungen zusammengefasst werden, bevor dies zu einer Anpassung des Gewichtes führt.

Bereits die Ausfaltung eines einzelnen Neurons für lediglich drei Eingabewerte führt zu deutlich größeren Netzen. Steigt die Anzahl der Eingabewerte deutlich an führt dies auch zu einem enormen Wachstum des Netzes. Ebenso steigt mit wachsender Anzahl von Neuronen die Komplexität des Netzes. Häufig werden jedoch gerade Netze mit einer größeren Anzahl Neuronen benötigt damit die gewünschte Funktionalität realisiert werden kann. Ein Hopfield-Netz, welches ebenfalls zu den Rekurrenten Neuronalen Netzen gehört, benötigt zur Erkennung von Mustern in einem Bild mit 10 mal 10 Pixeln bereits 100 Neuronen [21, S. 272]. Insgesamt existieren in diesem Beispiel-Netz dann 4950 Gewichte. Es gibt jedoch alternative Lernverfahren, auf die im Rahmen der Hopfield-Netze und Echo State Netze später eingegangen wird.

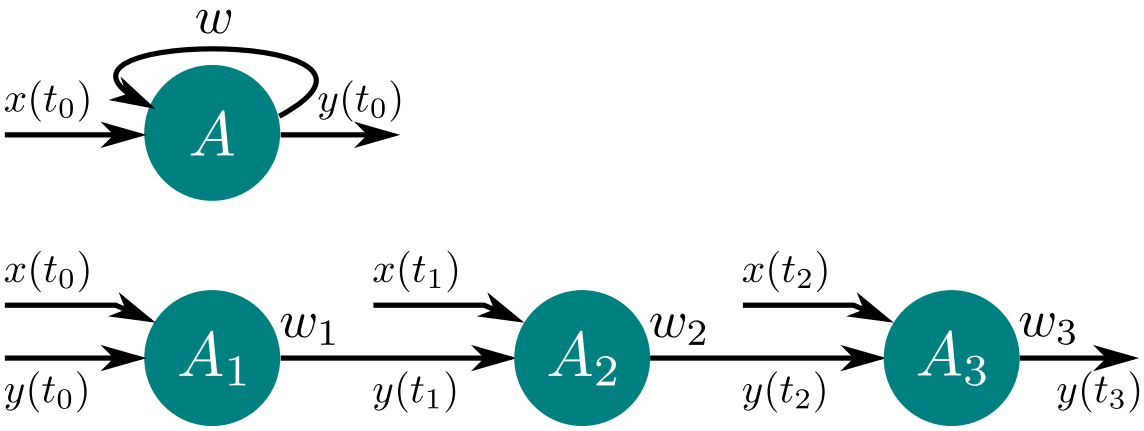


Abbildung 2.7: Ausfaltung in der Zeit, Quelle: Eigene Darstellung in Anlehnung an [22, S. 152]

2.5 Rekurrente Neuronale Netze

Rekurrente Neuronale Netze kommen den Vorbildern aus der Natur auf Grund ihrer Verknüpfungen am nächsten. Durch die Rückkopplungen besitzen sie zudem Eigenschaften, die sie für verschiedene Anwendungsfälle interessant machen. Diese Netze sind u.a. in der Lage Muster zu speichern und zu reproduzieren. Eine Eigenschaft, die an ein Gedächtnis erinnert. Sie eignen sich aber auch zur Zeitreihenanalyse, bei der Muster identifiziert werden können. Ebenso sind Vorhersagen zum Verlauf von Zeitreihen im entsprechenden Rahmen möglich. Auf Grund dieser Bedeutung soll auf diese Netzart tiefer eingegangen werden. Es wird in diesem Kapitel auf verschiedene Rekurrente Neuronale Netze gesondert eingegangen, deren Aufbau und Funktion wird vorgestellt sowie das Training dieser Netze näher betrachtet.

2.5.1 Hopfield-Netze

In seiner, 1982 veröffentlichten, Arbeit über künstliche neuronale Netze [26] ging Hopfield der zentralen Frage nach, ob es bei der Interaktion zwischen künstlichen Neuronen, also den elementaren Einheiten künstlicher neuronaler Netze, zu „nützlichen“ Phänomenen kommt. Und ob gewisse Leistungen lediglich daraus resultieren, dass sehr viele Neuronen miteinander interagieren. Als Beispiele für solche Phänomene nannte er das Entstehen remanenter Erinnerungen und die Entstehung von generalisierten Kategorien. Eigenschaften, die natürliche neuronale Netze besitzen. Eine Motivation dieses Verhalten zu untersuchen lag darin begründet, dass zwar das menschliche Verständnis in der Lage ist komplexe Netze und entsprechende Hardware zu planen, die Evolution jedoch keinen festen dafür Plan hat [26].

Physikalische Systeme in denen es durch die gemeinsame Interaktion ihrer elementaren Einheiten zu Phänomenen kommt, sind u.a. Magnetfelder. Elementarmagneten sind kleinste, magnetisierte Materialien, die einen magnetischen Dipol besitzen und damit über ein Magnetfeld verfügen. Durch dieses magnetische Feld üben sie eine Kraft auf andere Elementarmagneten aus, sind aber auch den Kräften anderer Magnetfelder ausgesetzt. Auf diese Weise besteht zwischen ihnen eine Wechselwirkung. Liegen die Elementarmagnete im System frei beweglich vor können sie sich ausrichten und es entsteht ein nach außen wirksames Magnetfeld.

Das von Hopfield aufgestellte Modell besteht aus einer Schicht Neuronen, die damit sowohl als Eingabe- und als Ausgabeschicht fungiert. Alle Neuronen sind miteinander verbunden, jedoch existieren keine direkten Rückführungen. Weiteres Merkmal ist die Symmetrie der Gewichte zwischen zwei Neuronen x_i und x_j, es gilt $w_{ij} = w_{ji}$. Die Gewichtsmatrix eines Hopfield-Netzes enthält auf der Hauptdiagonalen ausschließlich Nullen und ist entlang dieser Hauptdiagonalen symmetrisch.

Neuronen eines Hopfiled-Netzes können lediglich binäre Zustände annehmen. Ist es aktiv so ist $x_i = 1$. Bei Inaktivität ist $x_i = 0$. Zur Aktivierung eines Neurons x_i wird zunächst die Summe aller ihr zugesandten Werte (die der anderen Neuronen) gebildet. Diese wird dann mit dem zu x_i gehörenden Schwellwert u_i verglichen. Ist die Summe größer als der Schwellwert wird das Neuron aktiviert, andernfalls wird es deaktiviert. Die entsprechende Vorschrift zu Aktivierung ist in Formel 2.10 [26] zusammengefasst. Hopfield wählte zur Aktivierung ursprünglich einen asynchronen Ablauf. Das bedeutet, dass nicht alle Neuronen gleichzeitig ihren Status berechnen und entsprechend anpassen. Vielmehr wurde ein Neuron zufällig ausgewählt. Diese Entscheidung beruht auf der Tatsache, dass es bis dahin keinen Beweis für eine global gesteuerte und synchrone Aktivierung natürlicher Neuronen gab. Mittlerweile ist bekannt, dass für die Erfüllung spezifischer Funktionen, wie sie z.B. beim Gedächtnis benötigt werden, durchaus synchrone Aktivierungen erfolgen [27].

$$\begin{matrix} x_i = 1 \\ x_i = 0 \end{matrix} \quad \text{wenn} \quad \sum_{j \neq i} w_{ij} x_j \quad \begin{matrix} > u_i \\ < u_i \end{matrix} \tag{2.10}$$

Beim Training von Hopfield-Netzen findet eine erweiterte Form der Hebb'schen Lernregel statt (siehe Abschnitt 2.4). Haben zwei Neuronen den gleichen Status führt dies, wie bereits bekannt, zu einer Verstärkung, also einer Zunahme der Gewichte zu

den Verbindungen zwischen diesen zwei Neuronen. Hinzu kommt jedoch nun, dass Gewichte auch kleiner werden können und die Wertigkeit so abgeschwächt wird. Haben in einem Hopfield-Netz zwei Neuronen x_i und x_j einen unterschiedlichen Status, führt dies zu einer Abschwächung ihrer Verbindungen. Durch die Architektur und dem Lern- sowie Aktivierungsverhalten stellen Hopfield-Netze auch neuronale Autoassoziativspeicher dar, in denen binär codierte Muster gespeichert und abgerufen werden können. Um eine Anzahl s an Mustern, wobei jedes Muster durch einen Zustandsvektor x^s repräsentiert wird, zu speichern, müssen die notwendigen Gewichte w zwischen den Neuronen x berechnet werden. Dies geschieht nach der Vorschrift in Formel 2.11. In neuerer Literatur [21, 22] werden zur Darstellung der binären Zustände auch -1 und 1 statt 0 und 1 verwendet. So erfolgt in [21, S. 272] die Berechnung für die Aktivierung nach Formel 2.12 und für die Gewichtsbestimmung nach Formel 2.13.

$$w_{ij} = \sum_s (2x_i^s - 1)(2x_j^s - 1) \tag{2.11}$$

$$\begin{array}{ll} x_i = -1 & \\ x_i = 1 & \text{wenn} \quad \sum_{j \neq i} w_{ij} x_j \quad \begin{array}{l} < 0 \\ \geq 0 \end{array} \end{array} \tag{2.12}$$

$$w_{ij} = \frac{1}{n} \sum_s x_i^s x_j^s \tag{2.13}$$

Die Speicherkapazität eines Netzes ist u.a. von der Anzahl der Neuronen abhängig. Bei 100 Neuronen ermittelte Hopfield [26] eine maximale Speicherkapazität von 13 Mustern. Auch die Art der Muster spielt eine Rolle. Sind diese zu ähnlich neigt das Netz dazu diese zu vermischen und kann diese dann nicht mehr sicher unterschieden.

2.5.2 Elman-Netze

Elman setzte sich in seiner Arbeit „Finding Structure in Time“ [28] mit der Problematik auseinander, wie zeitlich abhängige Eingabewerte mit neuronalen Netzen verarbeitet werden können. Eine mögliche Darstellung, die es ermöglicht, eine zeitliche Abfolge eines Wertes zu verarbeiten, ist die Überführung in eine räumliche Struktur. Bei diesem Vorgang wird jeder einzelne Zeitschritt durch eine separate Dimension dargestellt. Mathematisch erhält man auf diese Weise von einer zeitlichen Abfolge T mit n Zeitschritten einen Vektor V mit n Dimensionen, sodass $n = \dim(V)$ mit $n \in N$ gilt. Für die Verarbeitung dieses Vektors kann damit ein Netz mit n Eingabeneuronen modelliert werden. Auch wenn dies für ausgewählte Szenarien praktikabel ist bringt diese Darstellung bei der Analyse zeitlich abhängiger Eingabewerte eine Reihe von Problemen mit sich. Ein Bsp. ist die zeitliche Verschiebung eines zu analysierenden Musters innerhalb einer Datenreihe, wie in Abbildung 2.8 dargestellt.

In Abbildung 2.8 ist das Basismuster selbst in den davon abgeleiteten Mustern A, B und C enthalten. Durch die zeitliche Einordnung und Verschiebung des Basismusters innerhalb der Datenreihe kommt es auch zu einer Verschiebung innerhalb der

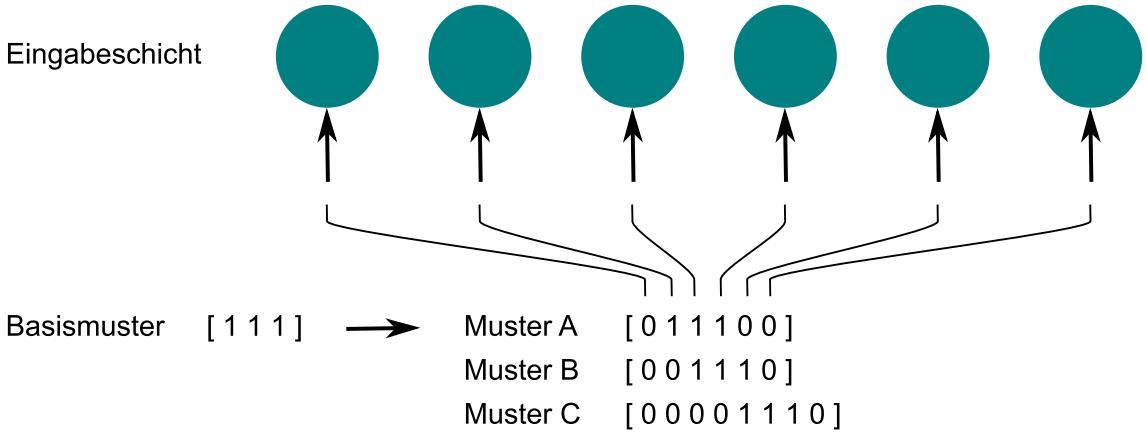


Abbildung 2.8: Darstellung von Zeit als räumliche Struktur, Quelle: Eigene Darstellung nach [28]

Dimensionen. Das Ergebnis einer geometrischen Interpretation der räumlichen Darstellung von Muster A und B wäre, dass diese nicht ähnlich und räumlich getrennt sind. Zwar können künstliche neuronale Netze lernen, dass es sich um ähnliche Muster handelt, dies wäre dann jedoch das Ergebnis eines externen Trainers, der dies dem Netz vorgibt [28, S. 181]. Die Ähnlichkeit wäre dann nicht aus der Struktur der Datenreihe selbst geschlossen worden. Ein weiteres Problem ist die Limitierung der Dimensionen eines Modells. Modellierte man ein neuronales Netz zur Erkennung des Basismusters in den Mustern A und B und beschränkte es auf $n = 6$, könnten höher dimensionierte Datenreihen wie z.B. Muster C damit nicht mehr verarbeitet werden. Entweder müsste zuvor Muster C analysiert oder das Modell adaptiert werden. Jordan und Elman analysierten daher, wie Muster in Zeitreihen in sequentieller Darstellung mit rekurrenten neuronalen Netzen verarbeitet werden können. Die Daten einer Zeitreihe werden bei diesen Netzen nicht mehr räumlich abgebildet und den Eingabeneuronen seriell zugeführt.

Das von Jordan entwickelte Netz [29] (Jordan-Netz), dargestellt in Abbildung 2.9, besitzt neben einer Eingabe-, Ausgabe- und verdeckten Schicht eine Kontextschicht. In [29] wurde für die Neuronen dieser Kontextschicht die Bezeichnung Statuseinheiten gewählt, die einen zeitlichen Kontext abbilden. Da Jordan diese Schicht später als Kontextschicht bezeichnet, soll dieser Begriff hier weiter Anwendung finden.

In einem Jordan-Netz sind die Neuronen der Eingabeschicht direkt und ausschließlich mit den verdeckten Neuronen verbunden, es existieren keine Rückkopplungen. Die verdeckten Neuronen sind mit allen Schichten verbunden. Sie erhalten Informationen sowohl aus der Eingabe- als auch aus der Kontextschicht. Ihre Ausgabesignale sind ausschließlich mit des Ausgabeneuronen verbunden. Auch hier existieren keine Rückkopplungen innerhalb der Schicht. Die Neuronen der Ausgabeschicht geben schließlich die Informationen nach außen und zur Kontextschicht weiter, sie bilden damit eine rückgekoppelte Struktur. Zu jedem Neuron der Ausgabeschicht existiert ein Neuron in der Kontextschicht. Das Gewicht der Rückkopplung beträgt 1 und führt daher zu keiner Änderung der zurückgeführten Werte. Innerhalb der Kontextschicht verfügt jedes Neuron über eine direkte Rückkopplung mit einem variablen Gewicht μ. Über diese direkten Rückkopplungen kann die Bestimmung der Ähnlich-

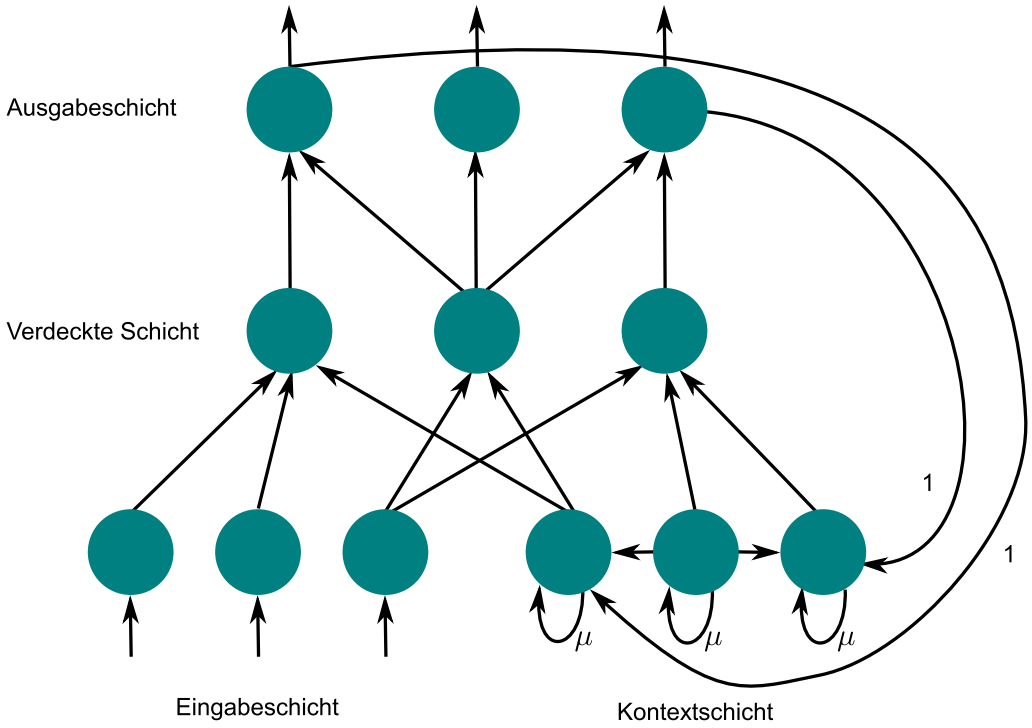


Abbildung 2.9: Jordan-Netz (nur ausgewählte Verbindungen sind dargestellt), Quelle: Eigene Darstellung nach [29, S. 10]

keit zwischen zwei Zuständen beeinflusst werden. Mit entsprechend großen Gewichten μ können Ähnlichkeiten auch in zeitlich größerer Ausdehnung gefunden werden [29, S. 11]. Das Netz verfügt auf diese Weise über eine Art Kurzzeitgedächtnis.

Elman modifizierte schließlich das Jordan-Netz und entwickelte daraus das in Abbildung 2.10 dargestellte Netz (Elman-Netz). Prinzipiell verfügt auch das Elman-Netz über die bereits bekannten Schichten und Verknüpfungen. Allerdings gehen die Rückkopplungen nun nicht mehr von den Ausgabeneuronen, sondern von den Neuronen der verdeckten Schicht aus. Die Anzahl der Neuronen in der Kontextschicht ist damit nicht mehr abhängig von der Anzahl der Neuronen der Ausgabeschicht, was das Elman-Netz flexibler als das Jordan-Netz macht [24, S. 128].

Beim Training der Elman- und Jordan-Netze werden die Rückkopplungen weggelassen. Damit entsteht ein vorwärts gerichtetes Netz welches wieder mit dem Backpropagation-Verfahren trainiert werden kann [24, S. 128]. Die Gewichte der Rückkopplungen werden nicht trainiert.

2.5.3 Langes Kurzzeitgedächtnis - LSTM-Netze

Am Beispiel des Jordan- und Elman-Netzes wurde bereits gezeigt, wie mit rekurrenten neuronalen Netzen temporäre Speicher realisiert werden können. Damit stehen dem neuronalen Netz sowohl aktuelle Informationen durch die Eingabeneuronen, als auch ältere Informationen durch die Neuronen der Kontextschicht zur Verfügung. Stehen diese chronologisch verschobenen Informationen im Zusammenhang, können diese durch das Netz ausgewertet werden. Problematisch wird es, wenn zwischen den zusammenhängenden Informationen sehr viel Zeit liegt und die gespeicherten

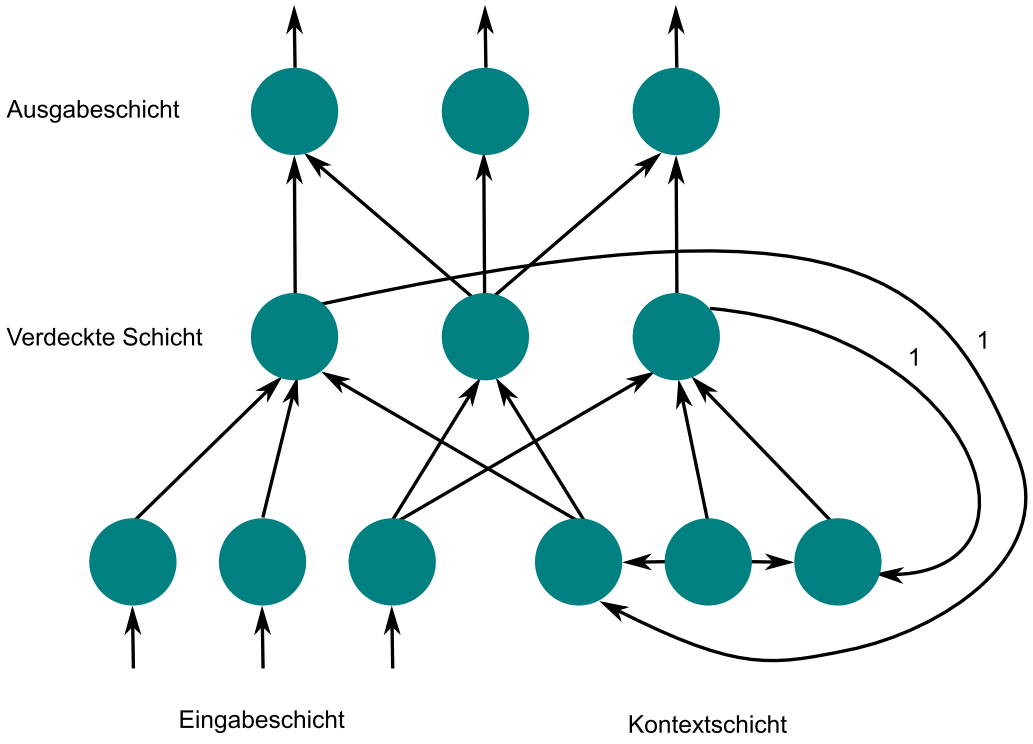


Abbildung 2.10: Elman-Netz (nur ausgewählte Verbindungen sind dargestellt), Quelle: Eigene Darstellung nach [28]

Informationen schwächer werden, so wie eine Erinnerung verblasst. Eine weitere Problematik entsteht beim Training, wenn die Netze über viele verdeckte Schichten verfügen, wie dies bei rekurrenten neuronalen Netzen durch die Ausfaltung in der Zeit geschehen kann. Hier kann dann das Backpropagation-Verfahren dazu führen, dass die Gewichtsanpassungen zu groß bzw. zu klein sind und die Korrektur oszillierend fortgeführt wird ohne eine optimale Anpassung zu erreichen. Oder es werden die falschen Gewichte adaptiert und der eigentliche Fehler damit nicht korrigiert. Ursprung dieser Phänomene sind u.a. die schwächer werdenden Fehlerinformationen bei jeder Iteration bzw. jeder zusätzlichen Neuronenschicht. Diese Zusammenhänge wurden von Hochreiter und Schmidhuber analysiert, die zu dem Ergebnis kamen, dass die Fehlerrückführung in der Zeit für mehrschichtige Netze ungeeignet ist. Im Ergebnis dieser Analysen entwickelten sie das lange Kurzzeitgedächtnis (Long Short-term Memory, LSTM) mit Strukturen, mit denen das Speichern und Abrufen von Informationen gesteuert werden kann [30].

Zentrales Element eines LSTM-Netzes ist das konstante Fehlerkarussell (constant error carousel, CEC). Hierbei verfügt ein Neuron über eine direkte Rückkopplung mit dem Gewicht $w_{jj} = 1$ und erlaubt bei linearer Aktivierung eine Rückführung. Daraus resultieren jedoch Konflikte für die Optimierung der Eingabe- und Ausgabegewichte eines Neurons j. Das Eingabegewicht w_{ji} ist konkurrierenden Optimierungen ausgesetzt. Einerseits muss es für das Speichern des Signals von i in j optimiert werden, andererseits auch für das Blockieren genau dieses Signals, wenn es für j irrelevant ist. Analog existieren Konflikte für die Ausgabegewichte von j. Einerseits müssen diese für den Zugriff auf den Inhalt von j durch ein nachfolgendes Neuron k optimiert

werden, andererseits dürfen aber auch keine irrelevanten Daten an k weitergegeben werden [30, S. 5 f.].

Zur Auflösung dieser Konflikte und Steuerung der internen Information eines Neurons führen Hochreiter et al. das Gate ein. Ein Gate besteht selbst aus einem neuronalen Netz, welches über eine sigmoidale Funktion (Abbildung 2.3 in Abschnitt 2.1.2) aktiviert wird. Das Input- und Output-Gate erweitern das CEC zu einer Gedächtniszelle, wobei nicht zwingend beide Gates vorhanden sein müssen. Das Input-Gate steuert in welchem Ausmaß ein Neuron j Informationen speichert, die es von anderen Neuronen erhält. Das Output-Gate steuert in welchem Ausmaß ein Neuron j seine gespeicherten Informationen an die empfangenden Neuronen weitergegeben wird [30, S. 6 bis 8]. Gers et al. erweiterten das LSTM-Netz schließlich um das Forget-Gate [31]. Mit diesem Gate kann der gespeicherte Wert einer Gedächtniszelle gezielt zurückgesetzt werden. Damit wird verhindert, dass intern gespeicherte Werte unendlich wachsen können und zu einem instabilen Netz führen.

2.5.4 Echo State Network

Eine weitere Struktur rekurrenter neuronaler Netze stellen die Echo State Netze (ESN) [32] dar. Der Aufbau folgt dabei weitestgehend der bekannten Struktur eines mehrschichtigen Netzes. Es existieren, wie in Abbildung 2.11 dargestellt, Eingabeneuronen sowie Ausgabeneuronen und verdeckte, nicht linear aktivierte Neuronen, die zusammengefasst als Reservoir bezeichnet werden. Innerhalb dieses Reservoirs werden alle Verbindungsgewichte zwischen den Neuronen zufällig festgelegt und im Rahmen eines Lernverfahrens auch nicht mehr verändert. Dies ist für alle Echo State Networks essentiell notwendig und eine zentrale Eigenschaft. Die Verbindungen der Eingabeneuronen können dagegen variabler gestaltet werden. Zwar werden auch diese über zufällige Gewichte mit allen Reservoirneuronen verbunden, wirken jedoch optional nur auf das Reservoir oder auch direkt auf die Ausgabeneuronen. Ebenso sind die rückwirkenden Verbindungen der Ausgabeneuronen optional. Die Reservoirneuronen werden mit einem zufälligen Wert initialisiert.

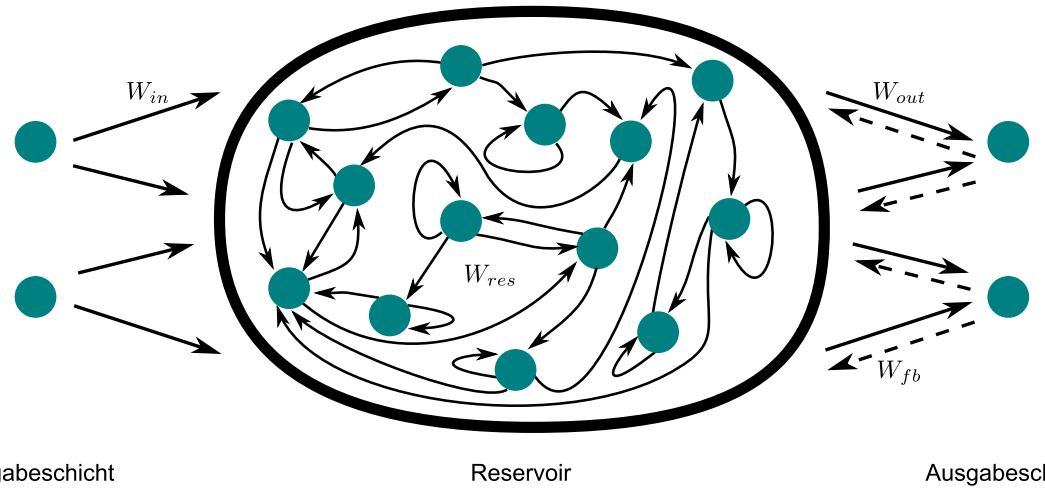


Abbildung 2.11: Echo State Network, Quelle: Eigene Darstellung nach [32]

Formal lässt sich ein Echo State Network ebenso mit der in Abschnitt 2.3 vorgestellten Matrix- und Vektorform beschreiben. Die Gewichte für das in Abbildung 2.11

dargestellte Netz können separat durch drei Matrizen beschrieben werden. Sei N die Anzahl der Neuronen im Reservoir, K die Anzahl der Eingabeneuronen und L die der Ausgabeneuronen. Die Gewichte der Reservoirneuronen werden in der Matrix W_{res} der Größe $N \times N$ zusammengefasst. Die Gewichte der Eingänge werden in der Matrix W_{in} der Größe $N \times K$ und die der rückwirkenden Verbindungen in der Matrix W_{fb} der Größe $N \times L$ zusammengefasst. Der aktuelle Zustand des Reservoirs zum Zeitpunkt n wird durch den Vektor $x(n)$ mit $\dim(x) = N$ repräsentiert. Entsprechend repräsentieren der Vektor $u(n)$ mit $\dim(u) = K$ zum Zeitpunkt n das Eingabesignal und der Vektor $y(n)$ mit $\dim(y) = L$ das Ausgabesignal. Sei f eine sigmoidale Funktion, können mit Hilfe der Berechnungsvorschrift in Formel 2.14 [32] die neuen Zustände der Neuronen zum Zeitpunkt $n + 1$ berechnet werden.

$$x(n+1) = f(W_{res}x(n) + W_{in}u(n+1) + W_{fb}y(n)) \tag{2.14}$$

Die Ausgabegewichte, welche als einzige in einem ESN trainiert werden, werden in der Matrix W_{out} der Größe $L \times (K+N)$ zusammengefasst. Zusammen mit dem erweiterten Systemzustand $z(n)$, welcher sich aus den Zuständen der Eingabe- und Reservoirneuronen entsprechend Formel 2.15 [32] zusammensetzt, wird der Wert der Ausgabeneuronen $y(n)$ nach Formel Formel 2.16 [32] berechnet. Existieren keine direkten Verbindungen zwischen der Eingabe- und Ausgabeschicht sind die entsprechenden Werte in W_{out} auf Null zu setzen. Typischerweise wird für die Aktivierungsfunktion g eine Sigmoid- oder die Identitätsfunktion verwendet (siehe Abschnitt 2.1.2).

$$z(n) = \begin{pmatrix} x(n) \\ u(n) \end{pmatrix} \tag{2.15}$$

$$y(n) = g(W_{out}z(n)) \tag{2.16}$$

Für das Trainieren der Ausgabegewichte W_{out} werden zunächst alle erweiterten Systemzustände $z(n)$ zum Eingangssignal $u(n)$ unter Verwendung von Formel 2.14 berechnet. Die Zustandsvektoren $z(n)$ werden zeilenweise in der Zustandsmatrix Z der Größe $n_{max} \times (N + K)$ zusammengefasst. Entsprechend werden die zugehörigen bekannten Ausgabesignale (Teacher Values) $d(n)$ zeilenweise in der Matrix D der Größe $n_{max} \times L$ zusammengefasst. Sollten im ESN Rückkopplungen durch W_{fb} existieren muss in den Ausgabeneuronen das jeweils zu n gehörende Ausgabesignal $d(n)$ geschrieben werden (Teacher forced Values). Die Ausgabegewichte können nun mit Formel 2.17 [32] berechnet werden. Das Ausgabesignal entsteht schließlich durch die mit W_{out} gelernte Linearkombination der Reservoirneuronen. Bei der Berechnung von W_{out} können die anfänglich gesammelten Systemzustände $z(n)$ aus Z auch entfernt werden (entsprechend auch die zugehörenden Signale d_n). Dieser „Washout“ ist dadurch begründet, dass in den gesammelten Systemzuständen $z(n)$ zu Beginn auch Zustände enthalten sein können, die nicht ausschließlich vom Eingangssignal $u(n)$, sondern auch vom zufälligen Initialzustand abhängig sind.

$$W_{out} = (Z^{-1}D)^{\top} \tag{2.17}$$

Das „Vergessen“ des Initialzustandes $x(0)$, sodass nach einem Washout der interne Zustand $x(n)$ des Reservoirs ausschließlich vom Eingangssignal $u(n)$ abhängt und sich diesem asymptotisch nähert, ist eine elementare Eigenschaft von Echo State Netzwerken und wird als Echo State Property bezeichnet. Diese Eigenschaft wird durch eine entsprechende Modellierung der Gewichtsmatrix W_{res} erreicht. Empirische Beobachtungen zeigen, dass die Echo State Property für jedes $u(n)$ erreicht wird wenn der Spektralradius von W_{res} kleiner als eins ist [32].

2.5.5 Vorhersagende neuronale Netze

Mit vorhersagenden neuronalen Netzen (Predicitive Neural Networks, PrNN), die zu den rekurrenten neuronalen Netzen gehören, stellen Stolzenburg et al. [1] einen Ansatz vor, in dem nicht nur die Gewichte von Neuronenverbindungen gelernt werden, sondern auch die Netzwerkarchitektur. Im Rahmen einer Zeitreihenanalyse werden damit zwei Ziele verfolgt. Zum einen sollen diese Reihen möglichst genau gelernt werden, um deren weiteren Verlauf bestimmen zu können. Werte sollen damit vorhersagbar werden. Zum anderen soll das neuronale Netz auf das für die Vorhersage notwendige begrenzt werden. Dies bedeutet, dass das neuronale Netz, wenn möglich, verkleinert wird. Einzelne Neuronen werden jedoch nicht schlicht entfernt. Vielmehr wird die gesamte Netzwerkarchitektur gelernt und es werden relevante Neuronen sowie deren Verbindungen identifiziert, um mit diesen ein neues Netz zu generieren.

Ein PrNN kann als Reservoir verstanden werden. Es existieren immer noch Neuronen für die Eingabe und Ausgabe, jedoch wird zwischen diesen nicht explizit unterschieden. Ein Neuron kann sowohl zur Eingabe als auch zur Ausgabe dienen. In Abbildung 2.12 sind exemplarisch zwei solcher Neuronen, die als Schnittstelle verstanden werden können, durch x_{I/O_1} und x_{I/O_2} gekennzeichnet, dargestellt. Auch wenn prinzipiell alle Neuronen als dem Reservoir zugehörig betrachtet werden können, werden weiterhin nur solche Neuronen als Reservoirneuronen bezeichnet denen keine Eingabe- oder Ausgabefunktion zukommt. Die Gewichte, die von diesen Schnittstellen-Neuronen wegführen, werden als Eingabegewichte in W_{in} zusammengefasst. Alle Gewichte die zu diesen Neuronen hinführen entsprechend in W_{out}. Alle Gewichte der übrigen Neuronenverbindungen werden in W_{res} zusammengefasst. Zusammen ergeben die Gewichtsmatrizen die Transitionsmatrix W des Netzes entsprechend Formel 2.18. Der Zustand der Eingabeneuronen wird durch den Vektor $x_{I/O}$ und der der Reservoirneuronen durch den Vektor x_{res} erfasst. Der Zustand des Netzes wird durch den Vektor x_n erfasst und folgt dem Aufbau entsprechend Formel 2.19. Der Initialzustand ist demnach x_0.

$$W = \begin{bmatrix} W_{out} \\ W_{in} \;\; W_{res} \end{bmatrix} \tag{2.18}$$

$$x_n = \begin{bmatrix} x_{I/O} \\ x_{res} \end{bmatrix} \tag{2.19}$$

Im Gegensatz zum ESN werden in einem PrNN die Neuronen linear aktiviert (Identitätsfunktion). Entsprechend kann der nächste Netzzustand im Wiedergabemodus mit Formel 2.20 berechnet werden. In diesem Modus berechnet das Netz seinen

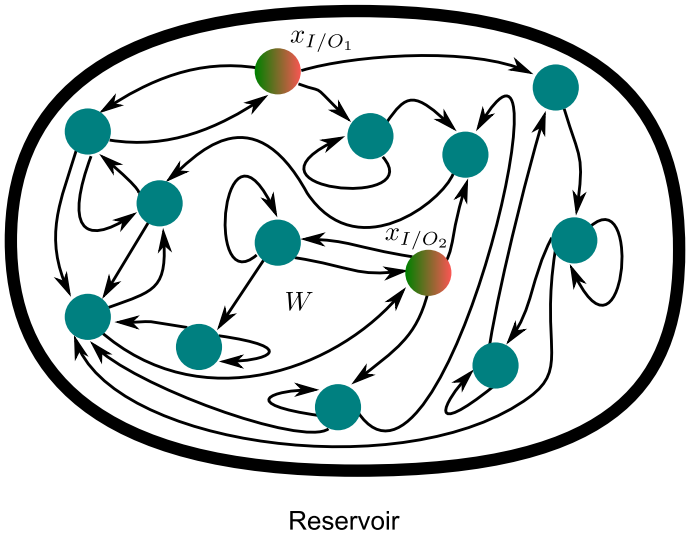


Abbildung 2.12: PrNN (Verbindungen sind nur exemplarisch dargetellt), Quelle: Eigene Darstellung nach [1]

nächsten Eingabewert selbst. Vor der Netzaktivierung dienen die Werte der Neuronen $x_{I/O}$ als Eingabewerte. Nach der Netzaktivierung werden diese überschrieben (wie bei Ausgabeneuronen) und stehen dem Netz als Eingabe wieder zur Verfügung. Auf diese Weise wird eine gelernte Funktion vom Netz nachgebildet und es kann zukünftige Werte berechnen. Dafür müssen jedoch die Gewichte in W_{out} bekannt sein. Um diese zu lernen, wird das Netz im Empfangsmodus betrieben. Hierbei sind alle Gewichte von W_{out} Null und es werden nach jeder Aktivierung n die dazu bekannten Werte der Zeitreihe $S(n)$ in die Neuronen $x_{I/O}$ geschrieben (Teacher forced Values).

$$\begin{array}{ll} x_n = x_0 & \\ x_n = W x_{n-1} & \text{wenn} \quad \begin{array}{l} n = 0 \\ n > 0 \end{array} \end{array} \tag{2.20}$$

$$R(n+1) = \begin{bmatrix} W_{in} & W_{res} \end{bmatrix} \begin{bmatrix} S(n) \\ R(n) \end{bmatrix} \tag{2.21}$$

Da durch $S(n)$ sowohl der initiale Wert als auch alle zukünftigen Werte bis einschließlich $S(n_{max})$ bekannt sind, müssen diese nicht berechnet werden. Es genügt die Berechnung der Zustände $R(n)$ der Reservoirneuronen (Formel 2.21). Wie bereits bei ESN werden die Zustände sequentiell gesammelt und zu einer Matrix zusammengefasst. So enthält die Matrix X aus Formel 2.22, mit Ausnahme des letzten Wertes, spaltenweise alle Werte der Zeitreihe $S(n)$ und Reservoirzustände $R(n)$. Der jeweils letzte Wert wird ausgelassen, da dieser als Eingabewert für den nächsten Ausgabewert fungiert. Demnach resultiert schließlich $S(n_{max})$ aus der Berechnung mit $S(n_{max}-1)$. Dagegen ist $S(n_{max}+1)$, der aus der Berechnung mit $S(n_{max})$ resultieren würde, unbekannt. Da alle Werte von $S(n)$ sowohl Eingabe- als auch Ausgabewerte sind, sind mit diesen die zu X korrespondierenden Ausgabewerte bekannt und werden entsprechend Formel 2.23 zu Y_{out} zusammengefasst. Der erste Wert wird ausgelassen, da dieser nicht berechnet (vorhergesagt) werden kann. Auf diese Weise enthält jede Spalte j in X den Eingabewert und Reservoirzustand, die schließlich

zur Berechnung der Ausgabe in Spalte j von Y_{out} führen. Unter Voraussetzung eines linearen Zusammenhangs zwischen X und Y_{out}, der durch die Gewichtsmatrix W_{out} realisiert wird (Formel 2.24), lässt sich schließlich W_{out} mit Formel 2.25 berechnen. Dies entspricht dem Lösen eines lineares Gleichungssystems.

$$X = \begin{bmatrix} S(0) & \cdots & S(n-1) \\ R(0) & \cdots & R(n-1) \end{bmatrix} \tag{2.22}$$

$$Y_{out} = \begin{bmatrix} S(1) & \cdots & S(n) \end{bmatrix} \tag{2.23}$$

$$Y_{out} = W_{out} X \tag{2.24}$$

$$W_{out} = Y_{out} / X \tag{2.25}$$

Bei der anschließenden Netzwerkreduktion werden für das Erlernen einer Zeitreihe $f(t)$ alle dafür notwendigen Komponenten im Netz identifiziert. Dies führt über die Eigenwertzerlegung der Transitionsmatrix W entsprechend Formel 2.26, wobei V die Eigenvektoren $v_1, ..., v_N$ und die Diagonalmatrix D die Eigenwerte $\lambda_1, ..., \lambda_N$ der Größe nach in absteigender Reihenfolge sortiert enthält. Mit diesen Eigenvektoren kann durch Linearkombination der Startvektor $x_0 = x_1 v_1 + ... + x_N v_N = Vx$ mit $x = [x_1 \cdots x_N]^\top$ dargestellt werden. Da W eine lineare Abbildung für die Eigenvektoren v_k und Eigenwerte λ_k mit $1 \leq k \leq N$ ist, gilt $Wv_k = \lambda_k v_k$ [1, S. 3 f.]. Ferner erhält man mit $Wx_0 = W(x_1 v_1 + \ldots + x_N v_N) = x_1 \lambda_1 v_1 + \ldots + x_N \lambda_N v_N$ für die Ausgabe Formel 2.27.

$$W = VDV^{-1} \tag{2.26}$$

$$f(t) = W^t x_0 = x_1 \lambda_1^t v_1 + \ldots + x_N \lambda_N^t v_N = VD^t x \tag{2.27}$$

Unter diesen Rahmenbedingungen und der Annahme von vorerst nur einer Dimension gilt $f(t) = \alpha_1 \lambda_1^t + \ldots + \alpha_N \lambda_1^N$, wobei α aus Formel 2.27 ermittelt werden kann und λ die Eigenwerte von W sind. Die Relevanz der Komponenten α_k und λ_k kann für n Trainingsfälle über die Fehlersumme mit Formel 2.28 bestimmt werden [1, S. 7]. Es werden anschließend alle Komponenten k verwendet, die die größten Werte E_k haben, wobei deren kumulierte Summe einen Schwellwert $\theta \leq 1$ nicht überschreiten darf. Sind die relevanten Komponenten identifiziert, werden mit deren Hilfe aus den Matrizen V und D sowie dem Vektor x komprimierte Versionen $\overset{*}{V}$, $\overset{*}{D}$ und $\overset{*}{x}$ wie folgt abgeleitet. Aus V werden Zeilen verwendet, die mit Ein-Ausgabekomponenten korrespondieren. Ebenso aus V solche Spalten, die relevante Netzwerkkomponenten enthalten. Aus D werden Zeilen und Spalten entsprechend relevanter Komponenten verwendet. Aus x werden Zeilen weiter verwendet, die relevante Komponenten enthalten. Dieses Verfahren ist sowohl für reelle, als auch für komplexe Transformationsmatrizen W definiert. Im Falle einer reellen Matrix W sind zusätzliche Transformationen notwendig [1, S. 5 bis 7].

$$E_k = \sum_{t=1}^{n} |\alpha_k \lambda_k^t|^2 = \begin{cases} \frac{|\alpha_k \lambda_k|^2 (1-|\lambda_k|^{2n})}{1-|\lambda_k|^2} & wenn |\lambda_k| \neq 1 \\ n|\alpha_k|^2 & sonst \end{cases} \tag{2.28}$$

2.5.6 Konzeptoren

Mit Konzeptoren [33, 16] stellt Jaeger einen Mechanismus vor, mit welchem neuronale Netze, ebenfalls auf dem Prinzip des Reservoirs basierend, trainiert, modifiziert und kombiniert werden können. Konzeptoren bieten ein breites Anwendungsfeld und können u.a. zur Mustererkennung eingesetzt werden [33, S. 1].

Konzeptoren adressieren das Problem, Muster in einem Reservoir speichern und wieder abrufen zu können. Es wird hierbei die Tatsache ausgenutzt, dass sich beim Training eines Reservoirs die Änderungen im Reservoir, die durch die Eingabewerte (Zeitreihe bzw. Muster) entstehen, auf einem linearen Unterraum des Gesamtraumes des Reservoirs beschränken. Dieser Unterraum ist wiederum charakteristisch für die jeweilig gelernten Eingabewerte bzw. Muster. Dargestellt ist dies für drei Muster p_1, p_2, p_3 mit einem Netz mit drei Reservoirneuronen in Abbildung 2.13. Jeder schwarze Punkt stellt dabei einen Reservoirzustand x_n zum Zeitpunkt n dar. Die unterschiedlichen Muster führen zu unterschiedlichen Reservoirzuständen, die zusammengenommen geometrisch interpretiert werden können und eine für das Muster charakteristische Form aufweisen. Diese Form kann als Ellipsoid interpretiert werden, welcher durch die Hauptkomponenten der Korrelationsmatrix R charakterisiert ist [16, S. 35].

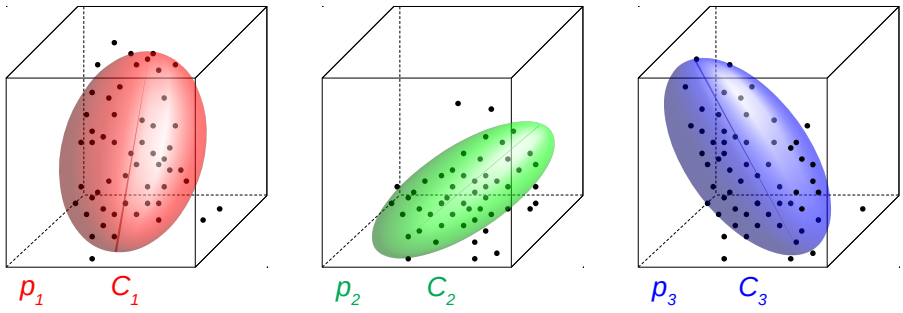


Abbildung 2.13: Konzeptoren C_1, C_2, C_3 für drei exemplarische Muster p_1, p_2, p_3 für ein Netzwerk mit N = 3 Reservoirneuronen, Quelle: Eigene Darstellung nach [33, S. 2]

Die Korrelationsmatrix R (Formel 2.29) ergibt sich wiederum aus den Reservoirzuständen x_n. Werden die Zustände x_n für ein Muster j spaltenweise zusammengesetzt, erhält man eine Reservoirzustandsmatrix X^j der Form $X = (x(1), \ldots, x(L))$ mit $L = n$. Der Konzeptor wird schließlich mit Hilfe der Kostenfunktion $\mathfrak{L} = (C|R^j, \alpha)$ definiert, dessen Minimierung schließlich $C(R^j, \alpha)$ liefert [16, S. 36]. Die erste Komponente C von $\mathfrak{L}$ berücksichtigt dabei, dass der Konzeptor als Projektionsmatrix fungieren soll und ergibt sich aus $E[||x - C_x||^2]$ [16, S. 36], mit $R = E[xx']$. Mit der zweiten Komponente wird beeinflusst, wie viele Dimensionen Einfluss auf die Projektion haben sollen. Sie ergibt sich durch $\alpha^{-2}||C||^2_{\text{fro}}$ [16, S. 36]. Insgesamt ergibt sich der Konzeptor damit durch die in Formel 2.30 [16, S. 36] gezeigte Vorschrift.

Der Konzeptor kann auch, wie in Formel 2.31 gezeigt, direkt mit R und α berechnet werden [16, S. 36]. Ebenso lässt sich R auch wieder durch C und α berechnen (Formel 2.32 [16, S. 36]). Der Parameter α dient zur Skalierung der Signalstärke [16, S. 43] und kann zur Modifizierung eines Konzeptors entsprechend Formel 2.33 angewendet werden [16, S. 44]. Ist α zunächst unbekannt wird als Standardwert 1 verwendet. Abhängig von der Verwendung der Konzeptoren existieren unterschiedliche, heuristische Ansätze zur Bestimmung von α [16, S. 49]. Auf das Gradienten-Kriterium [16, S. 49 f., 77 f.], welches im Rahmen dieser Arbeit Anwendung findet, wird in Abschnitt 4.1.1 gesondert eingegangen.

$$R^j = \frac{X^j (X^j)^\top}{L} \tag{2.29}$$

$$C(R, \alpha) = \text{argmin}_C \; E[||x - C_x||^2] + \alpha^{-2} ||C||^2_{\text{fro}} \tag{2.30}$$

$$C(R, \alpha) = R(R + \alpha^{-2} I)^{-1} = (R + \alpha^{-2} I)^{-1} R \tag{2.31}$$

$$R = \alpha^{-2} (I - C)^{-1} C = \alpha^{-2} C (I - C)^{-1} \tag{2.32}$$

$$\varphi(C, \alpha) = C(C + \alpha^{-2}(I - C))^{-1} \tag{2.33}$$

Ein Konzeptor kann nach Formel 2.34 [16, S. 37] schließlich dazu verwendet werden ein Muster j, welches in einem Reservoir gespeichert ist, aus den Reservoiraktivitäten zu filtern und auf die Ausgabeneuronen abzubilden (auf das Speichern von Mustern in einem Reservoir wird im Rahmen dieser Arbeit nicht weiter eingegangen, es sei auf [33, 16] verwiesen). Ein Konzeptor fungiert demnach als Projektionsmatrix.

$$x(n + 1) = C^j tanh(W x(n) + b) \tag{2.34}$$

Für Konzeptoren sind weiterhin die boolesche Operationen AND, OR und NOT definiert [16, S. 50 bis 57], auf die u.a. auch die De Morganschen Gesetze [34, S. 56 bis 58] angewendet werden können [16, S. 57]. Da sowohl die OR- als auch die NOT-Operation später im Klassifikator Anwendung finden, werden Berechnungsgrundlagen zu den booleschen Operationen im Folgenden ebenfalls vorgestellt.

Die OR-Operation kann entweder auf Basis der Korrelationsmatrizen oder direkt mit den zu verknüpfenden Konzeptoren durchgeführt werden. Bei Verwendung der Korrelationsmatrizen, wobei für zwei Konzeptoren gleicher Dimension C und B, $C = C(R, 1) = R(R + I)^{-1}$ und $B = B(Q, 1) = Q(Q + I)^{-1}$ gilt, I die Einheitsmatrix gleicher Dimension ist, findet Formel 2.35 Anwendung [16, S. 51]. Wird die Verknüpfung dagegen direkt mit C und B berechnet, findet Formel 2.36 Anwendung [16, S. 52]. Auch hier sind wieder C und B von gleicher Dimension und I sei die Einheitsmatrix. Voraussetzung ist, dass die Matrizen nicht singulär sind und damit

ihre Inversen definiert sind. Wird die OR-Operation auf Datenebene interpretiert, entspricht diese einer Addition der entsprechenden Korrelationsmatrizen [16, S. 52].

$$C \vee B := (R + Q)(R + Q + I)^{-1} \tag{2.35}$$

$$C \vee B = (I + (C(I - C)^{-1} + B(I - B)^{-1})^{-1})^{-1} \tag{2.36}$$

Die NOT-Operation kann ebenfalls auf Basis der Korrelationsmatrix (Formel 2.37) oder auf Basis des Konzeptors (Formel 2.38) durchgeführt werden [16, S. 52]. I sei wieder die Einheitsmatrix und ebenfalls vorausgesetzt ist wieder eine reguläre, invertierbare Korrelationsmatrix R. Die NOT-Operation kann auf Datenebene als die Berechnung des inversen Elements betrachtet werden [16, S. 52].

$$\neg C := R^{-1}(R^{-1} + I)^{-1} \tag{2.37}$$

$$\neg C = I - C \tag{2.38}$$

Die AND-Operation kann, wie die OR- und NOT-Operationen auch, auf Basis der Korrelationsmatrix (Formel 2.39) oder auf Basis des Konzeptors (Formel 2.40) durchgeführt werden [16, S. 52]. Dabei wird die AND-Operation zunächst mit Hilfe der De Morganschen Gesetze umgeformt. So erhält man für $a \wedge b = \neg(\neg a \vee \neg b)$, worüber Formel 2.39 schließlich hergeleitet wird. I sei wieder die Einheitsmatrix und ebenfalls vorausgesetzt ist wieder eine reguläre, invertierbare Korrelationsmatrix R.

$$C \wedge B := (R^{-1} + Q^{-1})^{-1}((R^{-1} + Q^{-1})^{-1} + I)^{-1} \tag{2.39}$$

$$C \wedge B = (C^{-1} + B^{-1} - I)^{-1} \tag{2.40}$$

2.6 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Begriffe zum Neuron (Abschnitt 2.1.1) und neuronalem Netz (Abschnitt 2.2) eingeführt und deren Funktionsweise, ausgehend vom Vorbild aus der Natur, erläutert. Anschließend wurden das Neuron und das neuronale Netz mathematisch beschrieben und die Begriffe des künstlichen Neurons und künstlichen neuronalen Netzes eingeführt (Abschnitt 2.1.2 und Abschnitt 2.3). Dabei wurde auf die verschiedenen Möglichkeiten der neuronalen Verbindungen eingegangen, die einen signifikanten Einfluss auf die Netzstruktur haben und schließlich bestimmen, ob es sich bei dem Netz um ein vorwärts gerichtetes oder rückgekoppeltes Netz handelt. Insbesondere wurde erläutert, was unter dem Training bzw. unter Lernen (Abschnitt 2.4) im Zusammenhang mit Neuronen zu verstehen ist. Auf Besonderheiten des Lernens wurde bei Bedarf in der Beschreibung der Netze gesondert eingegangen.

Im Speziellen wurde auf das Hopfield-Netz eingegangen, welches sich u.a. zur Mustererkennung eignet und in welchem die erweitere Hebb'sche Lernregel zur Anwendung kommt (Abschnitt 2.5.1). Es folgen die Jordan- und Elman-Netze (Abschnitt 2.5.2), die sich u.a. mit der Problematik der Zeitreihenanalyse auseinandersetzen. Zeitlich abhängige Daten werden nicht mehr als räumliche Struktur interpretiert, sondern den neuronalen Netzen sequentiell zugeführt. Um Zeitreihen zu verarbeiten verwenden Jordan und Elman rückgekoppelte neuronale Netze, die über eine Art Gedächtnis verfügen. Eine weitere, komplexere Netzart sind die LSTM-Netze (Abschnitt 2.5.3) von Hochreiter und Schmidhuber, die sich speziell mit dem Lern- und Speichervermögen von rekurrenten neuronalen Netzen auseinandergesetzt haben. Schließlich wurde das Echo State Network (Abschnitt 2.5.4) vorgestellt, bei dem es sich um einen weiteren Ansatz eines rekurrenten neuronalen Netzes handelt. Diese verfügen über eine große Anzahl zufällig generierter Neuronen, die als Reservoir zusammengefasst werden. Darüber hinaus werden nicht mehr potentiell alle Neuronen eines Netzes trainiert, sondern lediglich die Ausgabeneuronen. Weiterhin wurden die, ebenfalls auf einem Reservoir basierenden, vorhersagenden Netze (Abschnitt 2.5.5) und Konzeptoren (Abschnitt 2.5.6) vorgestellt, die im Rahmen dieser Arbeit von weiterer Bedeutung sind.

3. Datenerhebung

In diesem Kapitel wird die Datenerhebung, welche mit Hilfe eines Smartphones realisiert wurde, erläutert. Es wird geklärt welche Messgrößen erfasst und welche Fahrmanöver aufgezeichnet wurden. Ebenso wird dargestellt wie die Daten erhoben wurden, welche Software verwendet wurde und wie die Experimente, das eigentliche Durchführen der Fahrmanöver, zur Rohdatenerhebung realisiert wurden. Das Kapitel schließt mit der Erläuterung zur ersten Datenaufbereitung sowie einer Zusammenfassung.

3.1 Auswahl der Messgrößen und Sensoren

Zur Bestimmung welche Messgrößen zur Detektion von Fahrmanövern relevant sein können, wurde bereits in Abschnitt 1.2 auf die Ergebnisse Torkkolas et al. [8] eingegangen. Diese legen nahe, dass bereits Fahrzeug-eigene Sensoren, die die Betätigung des Gaspedals sowie den Stellwinkel des Lenkrades registrieren, einen Rückschluss auf den Grad der Ablenkung bzw. Aufmerksamkeit eines Fahrers erlauben. Da diese Daten nur intern im Fahrzeug zur Verfügung stehen und nicht ohne weiteren Aufwand ausgelesen werden können, darüber hinaus die Daten mit einem Smartphone möglichst flexibel erfasst werden sollen, werden Ersatzgrößen benötigt. Mit diesen Ersatzgrößen ist anschließend eine indirekte Messung zur Datenerhebung möglich.

An Stelle des Lenkradeinschlags wird die Beschleunigung über die wirkende Trägheitskraft erfasst, welche quer zum Fahrzeug wirkt. Im Falle eines Stillstands oder einer geradlinigen Vorwärts- bzw. Rückwärtsfahrt ist diese Null. Erst während einer Kurvenfahrt wirkt eine Zentrifugalkraft. Diese Trägheitskraft kann über den Beschleunigungssensor im Smartphone erfasst werden. Abhängig davon, ob es sich dabei um eine Links- oder eine Rechtskurve handelt ändert sich das Vorzeichen der gemessenen Beschleunigung. Die Vermutung ist, dass über den zeitlichen Verlauf der gemessenen Beschleunigung erste Rückschlüsse auf das Fahrverhalten gezogen werden können. Von Interesse wären z.B. wie stark und wie lange beschleunigt wird. So wird angenommen, dass ein Ausweichmanöver einen anderen Messwerte-Verlauf zeigen könnte als eine normale Kurvenfahrt.

Ebenso wie der Lenkradeinschlag ist auch die Erfassung des Gaspedals nicht ohne Weiteres möglich. Es wird stattdessen die Beschleunigung erfasst, welche längs zum Fahrzeug wirkt. Diese Kraft ist Null, wenn das Fahrzeug stillsteht oder die gleichmäßige Beschleunigung betragsmäßig gleich der Summe der Bremskräfte ist und das Fahrzeug eine konstante Geschwindigkeit fährt. Über das Vorzeichen allein kann nicht zweifelsfrei auf eine Fahrtrichtung bzw. auf ein Manöver geschlossen werden. So wirkt die Trägheitskraft bei einer vorwärts gerichteten Beschleunigung in die gleiche Richtung, wie bei der Bremsung einer Rückwärtsfahrt. Analog gilt dies für die Bremsung einer Vorwärtsfahrt und Beschleunigung einer Rückwärtsfahrt. Wollte man dies unterscheiden, würden zusätzliche Kriterien benötigt. Im Rahmen dieser Arbeit wird mit diesem Messwert zwischen einer vorwärts gerichteten Beschleunigung und der Abbremsung einer Vorwärtsfahrt unterschieden, die Manöver zur Messwertaufnahme wurden entsprechend gestaltet. Rückschlüsse auf das Fahrverhalten sollten sich dadurch ebenso durch den zeitlichen Verlauf realisieren lassen. So würden große gemessene Kräfte auf eine starke Beschleunigung bzw. Bremsung schließen lassen.

Alle vorgestellten Trägheitskräfte lassen sich mit Hilfe des Beschleunigungssensors, auch G-Sensor genannt, wie er in einem Smartphone verbaut ist, messen. Mit diesem Sensor ist eine Messung über drei Achsen x, y, z im Raum möglich. Das entsprechende Koordinatensystem (Smartphone-KOS), welches bei der Messung in Relation zum Smartphone betrachtet werden muss, ist in Abbildung 3.1 dargestellt. Alle Messungen von Beschleunigungen werden in der Einheit $\frac{\mathrm{m}}{\mathrm{s}^2}$ durchgeführt. Darstellungen von Messreihen in Diagrammen zur Beschleunigung erfolgen ebenfalls in dieser Einheit.

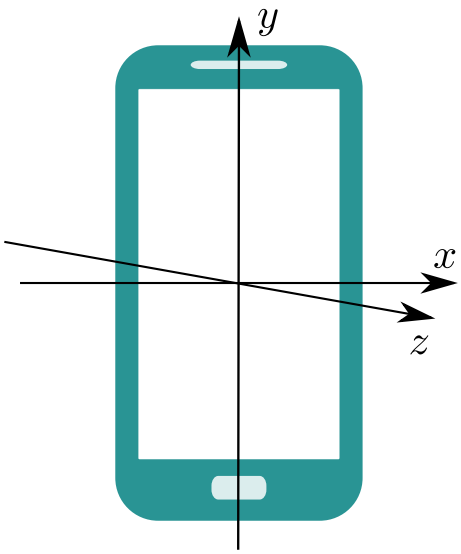


Abbildung 3.1: Koordinatensystem des Smartphones, Quelle: Eigene Darstellung

Als dritte Messgröße wird die Geschwindigkeit des Fahrzeuges erfasst. Da auch diese nicht direkt gemessen werden kann, wird über das Smartphone eine Positionsbestimmung mit Hilfe des empfangenen GPS-Signals realisiert. Über die kontinuierliche Positionsbestimmung und Messung der benötigten Zeit zwischen zwei Positionen kann die Geschwindigkeit berechnet werden. Diese wird in der Einheit $\frac{\mathrm{m}}{\mathrm{s}}$ erfasst. Darstellungen von Messreihen in Diagrammen zur Geschwindigkeit erfolgen ebenfalls in dieser Einheit.

Durch die Bestimmung der Geschwindigkeit können einige der zuvor genannten Probleme gelöst und die gemessenen Beschleunigungswerte in einem Kontext betrachtet werden. So kann durch Zusammenführen aller Daten zwischen Stillstand und einer gleichförmigen Bewegung unterschieden werden, was mit einer Kraftmessung allein nicht möglich wäre. Im gewissen Rahmen kann auch die Problematik abgemildert werden, dass z.B. nicht zwischen einer vorwärts gerichteten Beschleunigung und der Abbremsung einer Rückwärtsfahrt unterschieden werden kann, wenn die Geschwindigkeit einen Schwellwert überschreitet und eine Rückwärtsfahrt unplausibel wird.

3.2 Festlegung relevanter Manöver

Die Fahrmanöver, die erkannt werden sollen, wurden so gewählt, dass sie sich möglichst deutlich voneinander unterscheiden. Dies soll im Rahmen dieser Arbeit die Analyse erleichtern, in der lediglich die prinzipielle Machbarkeit einer Klassifikation im Zentrum des Interesses steht. Gleichzeitig sollten die gewählten Manöver allerdings auch elementare Szenarien einer Autofahrt repräsentieren. Zu den gewählten Szenarien gehört der Stillstand eines Fahrzeugs, das gleichförmige geradlinige Fahren, das positive und negative Beschleunigen sowie das Fahren einer Kurve. Die daraus resultierenden Fahrmanöver sind in Tabelle 3.1 aufgeführt und entsprechen im Wesentlichen auch den Manövern, wie sie z.B. auch in [10, 11, 13] verwendet wurden. Alle Fahrmanöver unterliegen der Einschränkung, dass die Fahrbahn gerade und trocken ist.

Tabelle 3.1: Relevante Szenarien und Manöver

Szenario	**Fahrmanöver**
Stillstand	Stillstand
Gleichförmiges geradliniges Fahren	Geradeaus
Beschleunigung (positive und negativ)	Anfahren Normalbremsung Vollbremsung
Fahren einer Kurve	Linkskurve Rechtskurve

Das erste Szenario ist der Stillstand eines Fahrzeugs. Im Rahmen dieser Arbeit wird dieser Zustand ebenfalls als Manöver betrachtet. Ein Klassifikator sollte den Zustand eines fahrenden Fahrzeuges von dem eines stehenden Fahrzeuges unterscheiden können. Gleichzeitig soll es als Referenz für alle folgenden Manöver dienen. Ergänzend dazu wird das Manöver Geradeaus in das Experiment aufgenommen. Es stellt eine gleichförmige Fahrt dar. Die Beschleunigung in Fahrtrichtung entspricht betragsmäßig der einwirkenden Bremskräfte (z.B. durch Rollreibung), sodass im Idealfall keine Beschleunigungen auf der X-Ache und Z-Achse gemessen werden.

Das Szenario Beschleunigung wird durch die Manöver Anfahren, Normalbremsung und Vollbremsung abgebildet. Beim Anfahren wird das Fahrzeug aus dem Stillstand heraus möglichst gleichmäßig beschleunigt, bis eine Geschwindigkeit von $20\frac{km}{h}$ erreicht ist. Dieses Manöver wird ausschließlich auf einer ebenen und geraden Strecke

durchgeführt. Um eine möglichst gleichmäßige Beschleunigung zu erreichen wird das Fahrzeug aus dem Stillstand heraus auf einer festgelegten Streckenlänge beschleunigt, die Endgeschwindigkeit muss innerhalb dieser Strecke erreicht sein. Das zweite Manöver ist die Normalbremsung. Hierbei muss, analog zum Manöver Anfahren, die (negative) Beschleunigung zum Abbremsen möglichst gleichmäßig stattfinden. Zu diesem Zweck wird ausgehend von einer gleichförmigen Fahrt bei einer Geschwindigkeit von $20\frac{\text{km}}{\text{h}}$ über einen festgelegten Streckenabschnitt abgebremst. Am Ende des Streckenabschnitts muss das Fahrzeug stehen. Das dritte Manöver ist die Vollbremsung. Dieses Manöver wird aus der gleichen Ausgangssituation heraus gestartet wie die Normalbremsung. Lediglich das Bremsen selbst ändert sich. Bei erreichen einer Streckenmarkierung wird eine Vollbremsung ausgelöst, auf die das Fahrzeug mit dem Einsatz des Antiblockiersystems reagiert. Das Manöver ist ebenfalls bei Erreichen des Stillstands abgeschlossen.

Das Szenario Kurvenfahrt wird mit zwei separaten Manövern berücksichtigt. So wird zwischen dem Fahren einer Rechts- und einer Linkskurve unterschieden. Die Fahrt findet bei einer konstanten Geschwindigkeit von $20\frac{\text{km}}{\text{h}}$ statt. Sie beginnt mit dem Einschlagen des Lenkrads (Einfahrt in die Kurve) und endet nach dem Durchfahren der Kurve auf gerader Strecke mit dem Erreichen der Mittelstellung des Lenkrads.

Insgesamt sind damit sieben Manöver identifiziert, die jeweils eine Klasse repräsentieren und für die Klassifizierung verwendet werden sollen. Eine Auflistung der Manöver mit der schließlich verwendeten Anzahl von Messreihen enthält Tabelle 3.6.

3.3 Datenerhebung mit einem Smartphone

Die zur Datenerfassung eingesetzte App muss diverse Anforderungen erfüllen, um im Experiment sicher verwendet werden zu können. Das reine Auslesen der Sensordaten genügt nicht. Ebenso wichtig wie das zuverlässige Erfassen der Messdaten ist auch die Bedienung der App während der Manöver. Zu diesem Zweck folgt eine Anforderungsanalyse um grundlegende Eigenschaften festzulegen, welche die App aufweisen muss. Abschließend werden in diesem Abschnitt drei recherchierte Apps und die schließlich selbst entwickelte App vorgestellt.

3.3.1 Anforderungsanalyse

Grundlegend lassen sich für die App zwei wichtige Bereiche festlegen, denen wichtige Eigenschaften zuzuordnen sind. Das ist einmal die Datenerfassung selbst und die Bedienung der App.

Zur Datenerfassung muss die App gleichzeitig die Daten des Beschleunigungssensors in $\frac{\text{m}}{\text{s}^2}$ und des GPS-Sensors auslesen. Letzterer dient zur indirekten Messung der Geschwindigkeit in $\frac{\text{m}}{\text{s}}$. Die Datenerfassung muss so realisiert sein, dass in einem Intervall von 100 ms aktuelle Messdaten vorliegen, was einer Abtastung von 10 Hz entspricht und 10 Messwerte pro Sekunde liefert. Diese Festlegung findet in Anlehnung an [8] statt. Weiterhin müssen die Messwerte angezeigt werden, dabei genügt jedoch zur einfachen Überprüfung, während das Smartphone im Fahrzeug ausgerichtet wird, eine Anzeige der aktuellen Messwerte. Eine grafische Visualisierung in Form einer Messkurve wird nicht benötigt. Die App muss bei Bedarf alle Messwerte

in chronologischer Reihenfolge aufzeichnen und in einer Datei ablegen können. Diese Datei muss leicht zugänglich sein. Das bedeutet sie muss vom Smartphone auf ein anderes Gerät (z.B. PC) leicht transferiert werden können. Die Daten sollen strukturiert in einer Textdatei, idealerweise als CSV, abgelegt werden. Damit soll eine weitere Verarbeitung mit anderen Programmen wie z.B. Notepad++ oder Octave ohne zusätzliche Konvertierung möglich sein. Idealerweise werden die Dateinamen automatisch mit Datum und Uhrzeit im Dateinamen gespeichert.

Neben der zuverlässigen Datenerfassung muss die App im Verlauf des Experiments sicher bedient werden können. Die Anzeige der Messdaten soll sich daher auf das minimal notwendige beschränken. Die Beschleunigungen auf den einzelnen Achsen müssen nur bei der Einrichtung des Smartphones gelesen werden können. Bei dieser Einrichtung wird das Smartphone möglichst senkrecht im Fahrzeug befestigt (in diesem Fall misst das Smartphone die ungefähre Erdanziehungskraft von 9,81 $\frac{\mathrm{m}}{\mathrm{s}^2}$ auf der Y-Achse, wenn es wie in Abbildung A.1 befestigt ist). Ebenso muss das GPS-Signal nur zur Kontrolle des Empfangs und der aktuellen Genauigkeit bei der Einrichtung abgelesen werden können. Es werden Bedienelemente benötigt um das Auslesen der Sensoren starten und stoppen zu können. Der Dateiname sollte frei gewählt werden können. Damit die Manöver bei der späteren Auswertung sicher und schnell innerhalb einer Messreihe identifiziert werden können, müssen diese markiert werden. Die App muss daher die Möglichkeit bieten das aktuelle Manöver angeben zu können. Diese Angabe muss die App zu jedem Messwert mit aufnehmen und in der Datei ablegen.

Da die Messungen idealerweise im normalen Verkehr stattfinden sollen und das Smartphone in einer definierten Position, wie in Abbildung A.1 zu sehen, im Fahrzeug befestigt sein wird, müssen die Bedienelemente so gestaltet werden, dass ein Beifahrer oder Fahrer die App bedienen kann. Dabei soll dieser das angezeigte Fenster nicht wechseln und nicht in Menüs navigieren müssen.

3.3.2 App-Recherche

Basierend auf den festgelegten Anforderungen wurden existierende Apps recherchiert und auf ihre Verwendbarkeit hin überprüft. Die App-Auswahl wurde nicht systematisch für alle verfügbaren Apps durchgeführt, sodass nur eine enge Auswahl relevanter Apps getestet wurde. Apps, die schlechte Bewertungen hatten oder unnötige Rechte bei der Installation verlangten (eine App zum Auslesen des Beschleunigungssensors sollte zur Funktionserfüllung nicht die Geräte-ID oder das Adressbuch auslesen müssen) wurden von Vornherein ausgeschlossen. Schließlich wurden drei Apps auf ihre Verwendbarkeit für dieses Experiment hin überprüft.

Die erste getestete App war das „Science Journal“ [35] in der Version 2.0.312 (Dresselhaus release) vom 12.10.2017 von Google LLC. Mit dieser lassen sich die Beschleunigungen in allen drei Achsen messen, jedoch müsste die Geschwindigkeit parallel dazu mit einer weiteren App gemessen werden. Der eigentliche Ausschluss ist jedoch darin begründet, dass in diversen aufgezeichneten Messreihen immer wieder Werte fehlten. Die Ursache dafür konnte leider nicht weiter geklärt werden. Jedoch hätte man die Werte interpolieren oder die Messreihe verwerfen müssen. Beides kam für die Datenerhebung nicht in Frage. Diese sollte ohne Manipulation der Messreihen

geschehen. Ebenfalls war die Menge nicht verwertbarer Messreihen nicht akzeptabel, da dies einen erheblichen Mehraufwand zur Datenerhebung bedeutet hätte. Da zur Durchführung nur ein Smartphone zur Verfügung stand konnte auch nicht geklärt werden, ob die App mit einem anderen Gerät auch unvollständige Messreihen produziert hätte.

Ebenfalls getestet wurde die App „Physics Toolbox Sensor Suite“ [36] in der Version 1.7.6 vom 18.10.2017 von Vieyra Software. Neben vielen weiteren Sensoren kann diese App ebenfalls die Beschleunigungen aufzeichnen. Zusätzlich steht auch das Aufzeichnen der GPS-Signale und damit der aktuellen Geschwindigkeit zur Verfügung. Obwohl diese App sich sehr gut für Experimente eignet, musste auf diese App verzichtet werden, weil die Zeitstempel, die während der Aufzeichnung der Beschleunigungsdaten mitgeschrieben werden, sich relativ auf den Startzeitpunkt des Experiments beziehen. Jede Aufzeichnung fängt daher bei Null an zu zählen und zeichnet nicht die aktuelle Uhrzeit auf. Dies ist allerdings notwendig, will man die Beschleunigungs-Messreihe mit GPS-Daten zusammenführen. So besteht allerdings die Gefahr, dass die Messreihen zueinander zeitlich verschoben zusammengeführt werden. Das synchrone Aufzeichnen von Beschleunigungsdaten und GPS-Daten ist nicht möglich.

Als dritte App wurde die „phyphox“ [37] in der Version 1.0.8 vom 12.07.2017 der RWTH Aachen getestet. Auch mit dieser App lassen sich umfangreiche Sensordaten erheben. Darunter auch Beschleunigungs- und GPS-Daten. Bei dieser App ist eine gleichzeitige Aufzeichnung beider Sensoren möglich. Es konnten auch keine fehlenden Messwerte festgestellt werden und die Messreihen lassen sich problemlos im gewünschten Format exportieren. Jedoch sind die Bedienelemente zu klein, um sie während der Fahrt komfortabel bedienen zu können.

3.3.3 Entwicklung einer Smartphone-App

Die App-Recherche lieferte im Ergebnis keine passende App. Zwar waren die getesteten Apps sehr gut und boten einen sehr guten Funktionsumfang, mit denen sogar Analysen von Messreihen möglich sind, jedoch waren sie mindestens in der Bedienung für die speziellen Erfordernisse nicht geeignet. Da eine reine Messwerterfassung mit Hilfe der Android Plattform Architektur und umfangreichen Bibliotheken ohne Weiteres möglich ist, fiel die Entscheidung auf die Entwicklung einer entsprechenden App.

Die „Sensor Logging App“, dargestellt in Abbildung 3.2, fasst alle Anzeigen und Bedienelemente auf einem Bildschirm zusammen. Da für das Experiment ein Samsung Galaxy S5 Neo zur Verfügung stand, welches über ein relativ großes Display (Displaydiagonale: 5,1 Zoll) verfügt, wurde die Anzeige nur für dieses Gerät optimiert.

In einem oberen Abschnitt werden auf der linken Seite die Beschleunigungs-Messwerte zu allen drei Achsen $x, y,$ und z in der Einheit $\frac{\mathrm{m}}{\mathrm{s}^2}$ angezeigt. Auf der rechten Seite daneben werden die über den GPS-Sensor ermittelte Geschwindigkeit, die Genauigkeit der Positionsangabe und der Breiten- (Latitude) sowie Längengrad (Longitude) angezeigt. Unterhalb dieser Anzeige befindet sich eine Schaltfläche mit der das Auslesen der Sensoren gestartet und gestoppt werden kann. Damit wird lediglich die

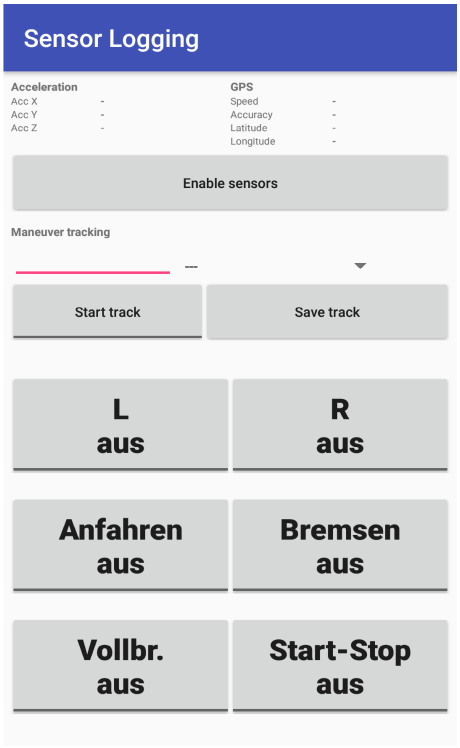


Abbildung 3.2: Sensor Logging App, Quelle: Eigene Darstellung

Interaktion mit den Sensoren gesteuert, eine Aufzeichnung erfolgt nicht automatisch. Es folgt eine Zeile in der zur Messreihenbezeichnung ein Freitext eingegeben werden kann. Alternativ, aber auch in Kombination anwendbar, können vordefinierte Manöverbezeichnungen über eine Liste gewählt werden. Beide Angaben werden für den Dateinamen verwendet. Wird keine der beiden Benennungsmöglichkeiten verwendet, wird automatisch ein Minus als Dateiname verwendet. Dem Dateinamen wird immer automatisch das aktuelle Datum sowie die aktuelle Uhrzeit zum Zeitpunkt des Abspeicherns vorangestellt. Über die Schaltfläche „Start track“ beginnt die Datenaufzeichnung. Diese wird zunächst im Arbeitsspeicher abgelegt und generiert nicht automatisch eine Datei. Über dieselbe Schaltfläche wird die Aufzeichnung gestoppt. Während einer aktiven Datenaufzeichnung ist die Schaltfläche mit „Stop track“ bezeichnet. Eine aufgezeichnete Messreihe verbleibt im Arbeitsspeicher bis die Ressourcen der App wieder freigegeben werden. Sie wird gelöscht wenn eine neue Aufzeichnung gestartet wird. Eine aufgezeichnete Messreihe kann über die Schaltfläche „Save track“ gespeichert werden. Es wird eine CSV-Datei generiert, die alle Messdaten zur letzten Aufzeichnung enthält. Speicherort ist das Verzeichnis „SensorTracker“ im Standardpfad für Apps.

Möchte man im Verlauf einer Fahrt mehrere Manöver aufzeichnen, können diese über die sechs Schaltflächen im unteren Fensterbereich entsprechend markiert werden. Die Markierung erfolgt binär ohne weitere Kodierung. In der generierten CSV-Datei werden sechs Spalten angelegt, beschriftet mit den entsprechenden Manövern der Schaltfläche. Ist ein Manöver nicht aktiv, wird in die entsprechende Spalte eine 0 eingetragen. Ist es aktiv, wird eine 1 eingetragen. Diese Markierungen werden entsprechend für jede erfolgte Messung vorgenommen. Dadurch ist es auch möglich

Manöverkombinationen (diese werden im Rahmen dieser Arbeit nicht analysiert), wie z.B. Anfahren in einer Kurve, zu markieren.

Die CSV-Dateien enthalten in der ersten Zeile immer die Spaltenbezeichnungen entsprechend der Datenfelder in Tabelle 3.2. Alle weiteren Zeilen enthalten die entsprechenden Messwerte und Manövermarkierungen. Dabei entspricht eine Zeile einen vollständigen Datensatz, der im Intervall von 100 ms erhoben wird.

Tabelle 3.2: Datenfelder einer Messreihe in der CSV-Datei

Datenfeld	**Datentyp**	**Beschreibung**
timestamp	Long	Aktuelle Systemzeit des Smartphones (UNIX-Timestamp)
links	Bool	Markierung für Manöver
rechts	Bool	Markierung für Manöver
anfahren	Bool	Markierung für Manöver
bremsen	Bool	Markierung für Manöver
vollbremsung	Bool	Markierung für Manöver
startstop	Bool	Markierung für Manöver
gps_time	Long	Systemzeit der Navigationssatelliten
speed	Double	Geschwindigkeit in $\frac{\mathrm{m}}{\mathrm{s}}$
accuracy	Double	Genauigkeit des Positionsangabe in m
latitude	Double	Geografische Breite in Dezimalgrad
longitude	Double	Geografische Länge in Dezimalgrad
acc_x	Double	Beschleunigung auf der X-Achse in $\frac{\mathrm{m}}{\mathrm{s}^2}$ (Bezug: Smartphone-KOS)
acc_y	Double	Beschleunigung auf der Y-Achse in $\frac{\mathrm{m}}{\mathrm{s}^2}$ (Bezug: Smartphone-KOS)
acc_z	Double	Beschleunigung auf der Z-Achse in $\frac{\mathrm{m}}{\mathrm{s}^2}$ (Bezug: Smartphone-KOS)

Die Datenerhebung muss Rücksicht auf die besonderen Anforderungen der einzelnen Datenquellen und die Architektur der App nehmen. Der Beschleunigungssensor bzw. G-Sensor kann kontinuierlich ausgelesen werden. Hier können u.a. vordefinierte Konstanten, wie z.B. SENSOR_DELAY_GAME (20 ms), die auch in der Sensor Logging App verwendet wurde, oder SENSOR_DELAY_UI (60 ms), ausgewählt werden. Der Sensorwert wird anschließend mit der voreingestellten Verzögerung ausgelesen, sodass in diesem Intervall auch ein aktueller Wert vorliegt. Im Gegensatz dazu kann das GPS-Signal nicht in einem vordefinierten Intervall ausgelesen werden. Hier liefert der Sensor-Manager (internes Konstrukt zur Verwaltung der Sensoren) immer nur dann einen neuen Wert, wenn sich die letzte Position geändert hat. Zwar besteht die Möglichkeit die jeweils letzte Position auszulesen, jedoch würde in den meisten Fällen nur ein bereits bekannter Wert ausgelesen und überschrieben werden. Bei einer hohen Abtastrate, um einen neuen Wert nicht zu verpassen, würden unnötig

viele Kapazitäten gebunden. Auf diese Methode wird daher verzichtet und stattdessen werden die GPS-Daten nur dann ausgelesen, wenn auch eine neue Position zur Verfügung steht. Würde man den Beschleunigungssensor zusammen mit dem GPS-Sensor auslesen, wenn dieser eine neue Position ermittelt hat, würden wichtige Informationen verloren gehen. Die höchste Genauigkeit (maximale Abweichung von der ermittelten Position) lag bei den durchgeführten Positionsbestimmungen bei 6 m. Bei $20\frac{\text{km}}{\text{h}} \approx 5,6\frac{\text{m}}{\text{s}}$ wird diese Strecke beinahe in einer Sekunde durchfahren. Wird die Genauigkeit schlechter und beträgt z.B. nur noch 8 m oder gar 12 m, wird die Strecke entsprechend größer. Ebenfalls das Zeitfenster, in dem keine Sensordaten erfasst werden. Im schlimmsten Fall stehen zum eigentlichen Manöver keine Beschleunigungsdaten zur Verfügung, wie das folgende Beispiel zeigt. Berechnet man den Bremsweg mit der Näherungsformel Anhalteweg in [m] $= (v[\frac{\text{km}}{\text{h}}])^2 100^{-1}$, wie sie in Fahrschulen vermittelt wird, benötigt eine Normalbremsung von $20\frac{\text{km}}{\text{h}}$ auf $0\frac{\text{km}}{\text{h}}$ unter idealen Bedingungen einen Bremsweg von näherungsweise 4 m. Diese Strecke ist kleiner als die geringste maximale Abweichung. Es besteht daher die Möglichkeit, dass nur unmittelbar vor und nach dem Bremsvorgang oder sogar nur zu einem dieser Zeitpunkte eine Positionsbestimmung durchgeführt werden kann. Würden die Beschleunigungsdaten mit dem GPS zusammen erfasst, stünden in solch einem Fall keine Beschleunigungsdaten zur Verfügung. Dies hat zur Folge, dass beide Sensoren über jeweils einen eigenen Thread (Sensorthreads) ausgelesen werden müssen.

Die ausgelesenen Werte werden, wie in Abbildung 3.3 dargestellt, in einem Datensatz abgelegt. Bei dem Datensatz handelt es sich um eine Speicherstruktur, welche die in Tabelle 3.2 aufgelisteten Datenfelder enthält. Der Datensatz ist dem Mainthread zuzuordnen und dient zentral dem Zugriff durch die Sensorthreads um den aktuellen Messwert abzulegen. Gleichzeitig werden die hier gespeicherten Werte für die Anzeige verwendet und die Markierungen, die der Anwender über die Bedienelemente vornehmen kann, gespeichert. Auf diese Weise enthält der Datensatz zu jedem Zeitpunkt aus allen Datenquellen die aktuellen Werte. Sobald ein neuer Wert zur Verfügung steht wird dieser entsprechend im Datensatz abgelegt und überschreibt den alten Wert. Die Anzeige wird automatisch aktualisiert.

Bei einer gestarteten Aufzeichnung wird nun der Datensatz in einem weiteren Thread mit einer Frequenz von 10 Hz abgetastet, sodass alle 100 ms ein Datensatz erfasst wird. Die erfassten Datensätze werden in das Repository kopiert, welches in chronologischer Reihenfolge alle Datensätze einer Aufzeichnung enthält. Auf diese Weise müssen die Sensorzugriffe nicht aufwendig synchronisiert werden und es steht dennoch immer der aktuelle Wert zur Verfügung. Das Repository kann bei Bedarf in eine CSV-Datei exportiert und im Dateisystem gespeichert werden. Das Repository wird gelöscht, wenn eine neue Aufzeichnung gestartet oder die App beendet wird.

3.4 Experimentelle Datenerhebung

Dieser Abschnitt beschreibt den Aufbau und die Durchführung des Experimentes zur Datenerhebung. Es wird bei allen Experimenten die entwickelte App verwendet um die beschriebenen Manöver zu erfassen. Der Aufbau wurde nicht verändert und ist stets der gleiche. Auch das verwendete Fahrzeug sowie das verwendete Smartphone sind bei allen Datenerhebungen dieselben. Alle aufgezeichneten Manöver wurden vom Autor selbst durchgeführt, der kein zertifizierter Testfahrer ist.

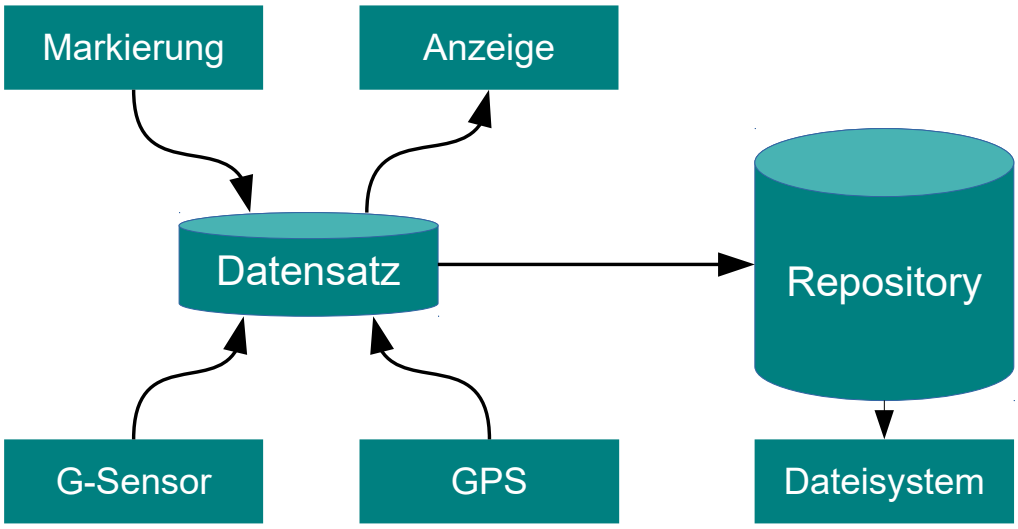


Abbildung 3.3: Sensor Logging App Struktur, Quelle: Eigene Darstellung

3.4.1 Aufbau des Experiments

Bei dem verwendeten Fahrzeug handelt sich um einen Audi A4 ohne Modifikationen mit den in Tabelle 3.3 aufgelisteten Eigenschaften. Während der Testfahrten befand sich ausschließlich der Fahrer im Fahrzeug. Weitere Insassen waren nicht anwesend. Ebenso wurde sicher gestellt, dass das Fahrzeug während der Testfahrten nicht beladen war. Alle Messungen wurden ausschließlich auf asphaltierten Fahrbahnen mit einer ebenen Oberfläche ohne Schlaglöcher oder anderen Unebenheiten durchgeführt. Ebenso wurde darauf geachtet, dass stets trockenes, Niederschlag freies Wetter herrschte und keine sonstigen Fahrbahnverschmutzungen vorhanden waren. Der Fahrer wurde zu keinem Zeitpunkt durch die Sonneneinstrahlung geblendet oder durch andere Quellen abgelenkt abgelenkt.

Tabelle 3.3: Eigenschaften Testfahrzeug

Attribut	**Wert**
Modell	A4 B6
Karosserieform	Kombi
Typenbezeichnung	8E
Baujahr	2002
Kraftstoffart	Benzin
Hubraum	1984 cm^3
Nennleistung	96 kW
Bereifung	Winterreifen, 195/65 R15

Für die Messungen wurde das Samsung Galaxy S5 Neo (Tabelle 3.4) verwendet, welches mit einem Universal-Smartphone-Halter der Firma Hama am Fahrzeug befestigt wurde. Die Halterung selbst wurde an den Lamellen eines Lüftungsschachtes durch einfaches Einstecken, wie in Abbildung A.1 dargestellt, befestigt. Anschließend wurde das Smartphone in der Halterung befestigt. Das Smartphone wurde vor

Beginn einer Messung unter Verwendung des Beschleunigungssensors (G-Sensor) näherungsweise senkrecht ausgerichtet [38, 39]. Überprüft wird dies durch die Messung der Erdanziehungskraft auf der Y-Achse. Da die Messwerte durch die App nicht zusätzlich geglättet werden (z.B. durch einen Tiefpassfilter), wurde die Ausrichtung bei ausgeschaltetem Motor vorgenommen. Die Toleranz lag bei $\pm 0,2 \frac{\mathrm{m}}{\mathrm{s}^2}$. Die Abweichungen vom Idealwert $9,81 \frac{\mathrm{m}}{\mathrm{s}^2}$ vergrößern sich durch die Vibrationen bei eingeschaltetem Motor, selbst im Stand. Während der Messungen lief ausschließlich die in Abschnitt 3.3.3 vorgestellte App auf dem Smartphone, sodass diese exklusiv auf den G-Sensor zugreifen konnte. Das Smartphone war mit keinen weiteren Geräten verbunden. WLAN und Bluetooth waren deaktiviert. Mobile Daten und GPS waren ohne Ausnahme aktiviert.

Tabelle 3.4: Eigenschaften Testsmartphone

Attribut	**Wert**
Hersteller	Samsung
Gerätename	Galaxy S5 Neo
Modellnummer	SM-G903F
Chipsatz	Exynos 7580 Octa
CPU	Octa-core 1.6 GHz Cortex-A53
Sensoren	Beschleunigungssensor Gyroskop Kompass Näherungssensor Herzfrequenzsensor
Positionsbestimmung	GPS-Modul mit A-GPS und Glonass
Betriebssystem	Android 6.0.1

Alle späteren Messungen verwenden diesen Aufbau und diese Konfiguration. Es wurden stets dasselbe Fahrzeug und Smartphone verwendet. Ebenso gab es keinen Fahrerwechsel, sodass alle Manöver vom Autor selbst gefahren wurden.

Auf die Einführung eines separaten Koordinatensystems (KOS) für das Fahrzeug wurde verzichtet, sodass bei allen Daten stets das Smartphone-KOS (Abbildung 3.1) das Bezugskoordinatensystem ist. Ein Grund dafür ist, dass beide KOS zueinander nicht gedreht sind und zur Verbesserung der Lesbarkeit eine entsprechende Achsenbezeichnung, die eine Interpretation der Daten erlaubt, genügt. Zusätzlich hätte dies einen Mehraufwand in der Datenverwaltung bedeutet, um sicherzustellen, dass Daten verschiedener KOS nicht zusammen ausgewertet werden. Die Interpretation der Achsen des Smartphone-KOS ist wie folgt. Auf der X-Achse sind die zum Fahrzeug seitlich angreifenden Kräfte ablesbar. Diese wirken quer zur Fahrtrichtung und ändern ihren Betrag bei Kurvenfahrten. Auf der Z-Achse sind die Kräfte ablesbar, die längs zur Fahrtrichtung wirken. Diese ändern ihren Betrag beim Beschleunigen (Bremsvorgänge werden als verzögernde Beschleunigungen angesehen). Auf der Y-Achse sind Kräfte ablesbar, die senkrecht zum Fahrzeug wirken. In den vorgestellten Manövern ist diese konstant (idealisiert). Betragsänderungen würden

entstehen, würde das Fahrzeug mit einem Lift bewegt werden. Diese Betrachtungen beziehen sich auf ein ideales System. Im realen System sind diese idealen Bewegungen nicht möglich, sodass während der Manöver immer Auswirkungen auf allen Achsen messbar sind. Allein durch die Vibration entstehen messbare Schwankungen. Bei einer Fahrt über eine Steigung, was im Experiment vermieden wurde, würde sich das Smartphone neigen, was zu einer Betragsänderung der gemessenen G-Kraft (Erdanziehung, messbar auf Y-Achse) führt. Die wesentlichen Auswirkungen sind in Tabelle 3.5 zusammengefasst.

Tabelle 3.5: Kraftwirkung (ideal, $a_0 = 0$) im Smartphone-KOS

Manöver	**Achse**	**Vorzeichen**	**Betrag**
Stillstand und Geradeaus	X-Achse		$= 0$
	Y-Achse	+	$\approx 9,8$
	Z-Achse	-	$= 0$
Anfahren in Fahrtrichtung	X-Achse		$= 0$
	Y-Achse	+	$\approx 9,8$
	Z-Achse	-	> 0
Abbremsen in Fahrtrichtung	X-Achse		$= 0$
	Y-Achse	+	$\approx 9,8$
	Z-Achse	+	> 0
Linkskurve	X-Achse	-	> 0
	Y-Achse	+	$\approx 9,8$
	Z-Achse		$= 0$
Rechtskurve	X-Achse	+	> 0
	Y-Achse	+	$\approx 9,8$
	Z-Achse		$= 0$

3.4.2 Durchführung des Experiments

Im Rahmen der Durchführung wurden die Manöver unter den beschriebenen Rahmenbedingungen durchgeführt. Um die Bedienbarkeit der App während einer Fahrt zu evaluieren, wurden zunächst gerade Strecken auf einem separaten, abgesperrten Bereich gefahren. Es stellte sich bei diesen Experimenten heraus, dass eine sichere Bedienung der App trotz großer Schaltflächen nicht gewährleistet werden konnte. Die sichere Bedienung der App, die für zuverlässige Messungen notwendig ist, stellt bereits eine zu große Ablenkung und damit ein nicht vertretbares Risiko dar. Die ursprüngliche Annahme bei der Entwicklung der App, dem Fahrer die Bedienung zumuten zu können, stellte sich als falsch heraus. Neben der Ablenkung traten zudem Probleme durch gemessene Erschütterungen durch die Bedienung der App auf. Unter der Annahme, ähnliche Effekte auch bei der Bedienung durch einen Beifahrer zu erhalten, wurde die Möglichkeit, die Messungen mit einem Beifahrer durchzuführen nicht weiter verfolgt.

Die finale Messmethode wurde schließlich ohne Beifahrer in der Art ausgeführt, dass vor Fahrtantritt ein Manöver in der App gewählt wurde. Anschließend wurde dieses

Manöver gefahren und es wurden die Messdaten erhoben. Dies hatte jedoch zur Folge, dass nun alle Manöver in einer Messreihe gleich markiert waren. Die Messreihen mussten daher anschließend manuell aufbereitet werden, da eine automatische Selektion auf Basis der Manövermarkierungen nicht mehr möglich war. Deutlich wird dies am Beispiel der Messung einer Linkskurve. Das Manöver wird vor Fahrtantritt in der App markiert und die Messung wird gestartet. Die Fahrt beginnt nun allerdings mit dem Anfahren. Es folgt die Linkskurve und abschließend eine Bremsung. Auf diese Weise werden in einer Messreihe drei Manöver aufgenommen und alle als Linkskurve markiert. Der Mehraufwand wurde angesichts des minimierten Risikos bei der Fahrt in Kauf genommen.

Schließlich wurden die Manöver außerhalb des normalen Straßenverkehrs aufgezeichnet. Zum einen lies sich damit das Risiko eines Unfalls weiter reduzieren. Zum anderen gewährleistete dies ähnliche Bedingungen bei den Manövern. Dadurch wurde wiederum sichergestellt, dass sich die Messreihen innerhalb einer Manöverart nicht zu sehr voneinander unterschieden. Identische Messreihen können auf Grund der manuellen Bedienung des Fahrzeugs, grober Manöverparameter und individuellen Fahrweise ausgeschlossen werden.

3.5 Datenselektion

Die Messdaten wurden auf Grund der beschriebenen Problematik bei der Durchführung des Experiments manuell bearbeitet. Dies bedeutet, es wurden aus einer Messreihe die Abschnitte extrahiert, die das tatsächliche Manöver darstellen. Es wurden jeweils auch bis zu zehn Messpunkte extrahiert, die unmittelbar vor und nach dem Manöver liegen und es entsprechend einleiten und beenden. Die Anzahl wurde zufällig festgelegt um eine Erhöhung der Varianz zu erreichen.

Für die Weiterverarbeitung durch die KNN wurden je Messreihe nur die Geschwindigkeit und die Beschleunigungen übernommen. Die Länge eines Manövers ergibt sich indirekt über die Anzahl der Messpunkte, die in einem Abstand von 100 ms aufgenommen wurden. Es wird ebenfalls wieder das CSV-Format verwendet, jetzt jedoch nur mit vier Spalten.

Insgesamt wurden für die Auswertung entsprechend Tabelle 3.6 78 Messreihen ausgewählt. Zwar wurden deutlich mehr Manöver aufgezeichnet, jedoch wurden nicht alle für die Weiterverarbeitung übernommen. Gründe dafür waren u.a. zu ungenaue GPS-Bestimmung (Genauigkeit deutlich schlechter als 12 m) oder überlappende Manöver (starke Beschleunigung während einer Kurve).

Die durchgeführten Messungen sind für eine allgemeine Aussage nicht geeignet. Zum einen ist die Anzahl stark begrenzt, was zu einer ungenügenden Datenlage führt und für eine Verallgemeinerung nicht geeignet ist. Zudem könnten weitere Messungen zusätzliche wichtige Varianten liefern, die ebenfalls ein Manöver repräsentieren, bei der geringen Datenlage aber nicht abgedeckt sind. Weitere wesentliche Kritikpunkte sind die Verwendung nur eines Fahrzeugs, eines Smartphones und eines Testfahrers. Weitere Messungen und Analysen müssten zeigen, inwieweit sich eine Veränderung dieser Parameter auf die Messdaten auswirkt. Auch wenn die Manöver eng begrenzt sind, ist von vornherein nicht auszuschließen, dass z.B. andere Fahrwerke, Motoren

Tabelle 3.6: Anzahl gemessener Manöver

Manöver	**Anzahl**
Stillstand	10
Geradeaus	9
Anfahren	11
Normalbremsung	8
Vollbremsung	8
Linkskurve	15
Rechtskurve	17

oder Smartphones weitere Varianten liefern würden und die Messreihen selbst bei gleichem Manöver unterscheidbar wären. Auch das Einsetzen weiterer Fahrer kann zu anderen Messreihen führen, sodass hier bereits Rückschlüsse, bzw. Einflüsse der Fahrer, messbar wären. Ein Fahrer könnte z.B. Kurven immer nahezu konstant fahren, während andere Fahrer prinzipiell beschleunigen oder bremsen, wenn sie eine Kurve durchfahren. Ebenso zu überprüfen wäre die Auswirkung der ausgewählten Strecke. So könnte ein Fahrer in engen Gassen eine Kurve anders (vorsichtiger) durchfahren als auf einem leeren ausgedehnten Parkplatz. Auf diese Weise wäre vorstellbar, dass eine andere Strecke bei gleichen Manöver ebenfalls weitere Varianten einer Manövermessreihe liefern könnte. Diese Aspekte sind im Rahmen der Datenauswertung und Interpretation kritisch zu betrachten. Die erhobenen Daten sind daher nicht repräsentativ und beschreiben u.U. ein Manöver nicht voll umfänglich, sondern nur im Rahmen des beschriebenen Experiments. Da die vorliegende Arbeit jedoch die prinzipielle Machbarkeit einer Manöverklassifikation untersuchen will, wird in diesem Rahmen mit der vorgestellten Datenlage gearbeitet.

Jeweils eine Messreihe zu ausgewählten Manövern ist in Abbildung 3.4 und Abbildung 3.5 abgebildet. Bei näherer Betrachtung der einzelnen Messreihen ist deutlich zu erkennen, dass die Messwerte näherungsweise die erwarteten Entwicklungen aufzeigen, jedoch auch Abweichungen zu Idealverlauf aufzeigen. Während des Stillstands (3.4a) ist wie erwartet nur die Erdanziehungskraft gemessen worden. Das Rauschen ist größtenteils auf die Vibrationen zurückzuführen, die bei eingeschaltetem Motor auftreten. Die Messreihen zum Anfahren (3.4c) zeigen wie erwartet einen Anstieg der Geschwindigkeit und Kräfte, die vorwiegend auf der Z-Achse wirken und ein negatives Vorzeichen haben. Die, wie erwartet, gegensätzliche Entwicklung zeigen die Messreihe zum Bremsmanöver (3.5a). Die Kraft auf der Z-Achse wirkt mit positiven Vorzeichen und die Geschwindigkeit verringert sich. Der Grund, dass die Geschwindigkeit bei den Messungen nicht zwangsläufig Null ist, obwohl das Fahrzeug nach dem Bremsmanöver steht, ist der Art der Geschwindigkeitsermittelung geschuldet. Die Geschwindigkeit wird aus der Zeit und dem zurückgelegten Weg unter Verwendung des GPS-Signals ermittelt. Ein neues GPS-Signal wird vom System zur Verfügung gestellt, wenn die Position sich geändert hat. Erfolgt der Stillstand innerhalb der Genauigkeitstoleranz des GPS-Signals ist keine neue Position verfügbar. Auf diese Problematik wurde bereits in Abschnitt 3.3.3 hingewiesen. Eine Manipulation der Daten, z.B. durch Anfügen einer Null, wurde nicht durchgeführt. Das Interesse

gilt der Verarbeitung realer Messdaten. Ebenfalls erwartete Messdatenverläufe sind bei den Kurvenfahrten in 3.4e und 3.4f zu erkennen. Je nach Kurvenrichtung ist die auftretende Kraft auf der X-Achse deutlich zu erkennen. Ebenso deutlich erkennbar ist der Umstand, dass es sich hierbei nicht um ideale Manöverfahrten handelt. So ist in 3.4e ebenfalls eine Beschleunigung ablesbar und in 3.4f ein Bremsvorgang (dieser Umstand ist der verringerten Beschleunigung geschuldet und entspricht einem Ausrollen, es wurde nicht aktiv gebremst). Entsprechend erkennbar am Verlauf der Geschwindigkeit und der Beschleunigung auf der Z-Achse. Dennoch ist eine Unterscheidung zum Anfahren möglich. Die Anfangsgeschwindigkeit ist größer Null und die Kräfte auf der X-Achse sind deutlich größer als beim Manöver anfahren. Ebenso besteht die Möglichkeit zur Unterscheidung im Fall des Bremsens. So ist bei der Kurve die Endgeschwindigkeit größer und ein Ausschlag auf der X-Achse zu sehen. Zusätzlich ist beim Bremsmanöver die Kraft auf der Z-Achse deutlich größer als bei der Kurvenfahrt. Auf Grund dieser Merkmale und weiteren prinzipiellen Möglichkeit der Unterscheidung wurden auch diese Messreihen beibehalten und für die Auswertung durch die KNN verwendet.

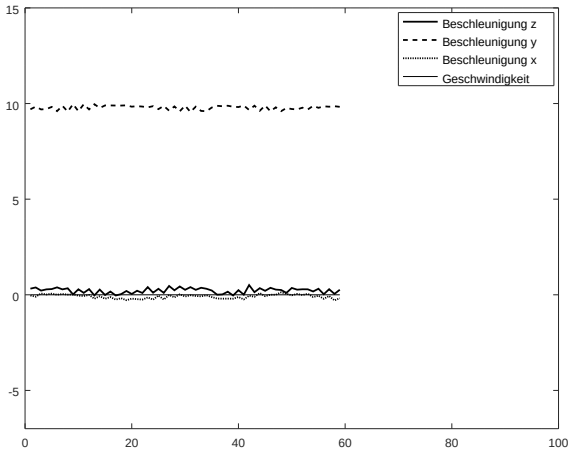
Da nicht nur die Unterscheidung grundlegend verschiedener Manöver von Interesse ist, sondern auch untersucht werden soll ob verschiedene Ausprägungen eines Manövers erkannt werden können, wurden auch Messreihen von Vollbremsungen aufgenommen. Prinzipiell handelt es sich dabei ebenfalls um ein Bremsmanöver, bei dem die Kräfte in dieselbe Richtung wirken wie bei der Normalbremsung. Jedoch zeigen die Messungen zu Vollbremsungen, von der eine in Abbildung 3.5 einer Normalbremsung gegenübergestellt ist, dass deren zeitlicher Verlauf und maximaler Betrag ein anderer ist. Dies entspricht auch der Erwartung, da bei einer Vollbremsung das Fahrzeug deutlich schneller zum stehen kommt. Die wirkenden Kräfte bei einer Vollbremsung müssen daher kürzer, dafür aber stärker auftreten. Bei allen Messreihen zur Vollbremsung griff das Antiblockiersystem aktiv ein, sodass dies in die Messungen mit einfloss.

3.6 Zusammenfassung

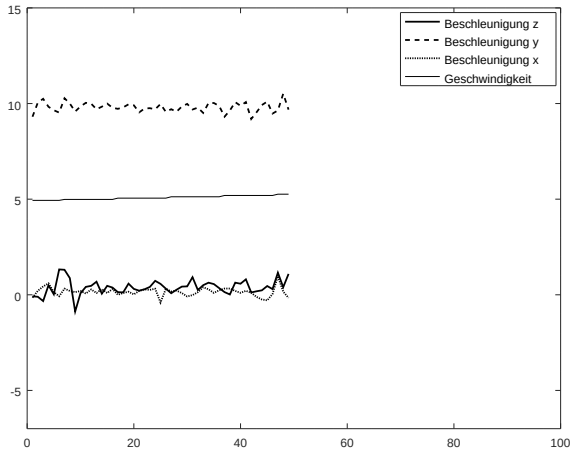
Im aktuellen Kapitel wurde die Datenerhebung vorgestellt, welche Daten relevant sind und wie diese erhoben wurden. Zunächst wurden dafür die einzusetzenden Sensoren betrachtet, mit denen die relevanten Messgrößen entweder direkt oder wie bei der Geschwindigkeit, indirekt erfasst werden können. Ebenfalls betrachtet wurden die Manöver, bei deren Durchfahren die Messdaten erhoben werden sollten. Hierfür wurden Rahmenbedingungen festgelegt, die das Manöver beschreiben. Diese Manöver sollen später durch das neuronale Netz auf Basis der erhobenen Daten erkannt und klassifiziert werden können.

Es wurden Apps als Ergebnis einer App-Recherche vorgestellt und erläutert, in welchem Umfang diese für die Messwerterfassung eingesetzt werden könnten. Da keine App den zuvor definierten Anforderungen entsprach, wurde eine einfache App zur Messwerterfassung entwickelt, deren grundlegende Funktion und Architektur in diesem Kapitel vorgestellt wurden.

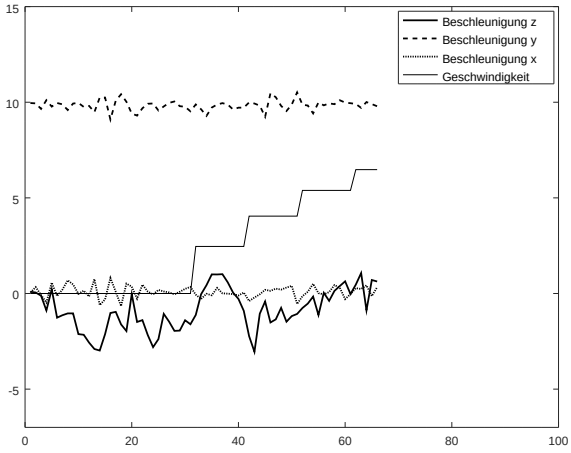
Der Aufbau des Experiments mit den verwendeten Geräten wie Fahrzeug und Smartphone wurden vorgestellt. Ebenso wie die Durchführung des Experiments. Insbeson-



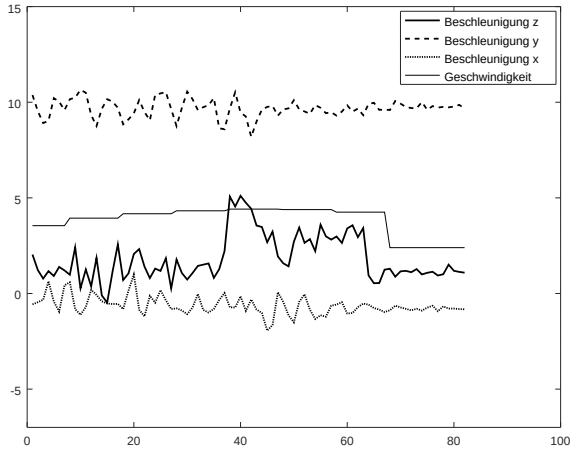
(a) Messreihe 1 Stillstand



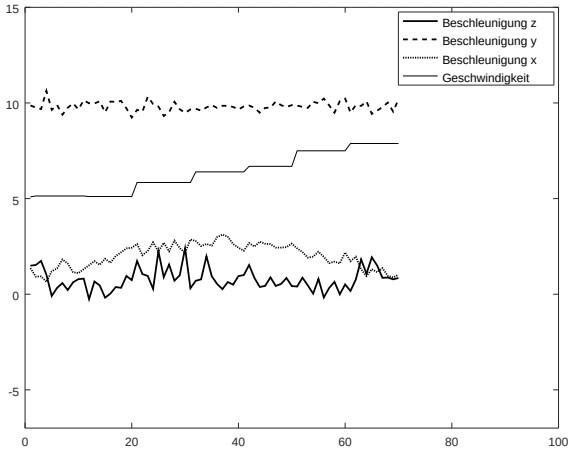
(b) Messreihe 1 Geradeaus



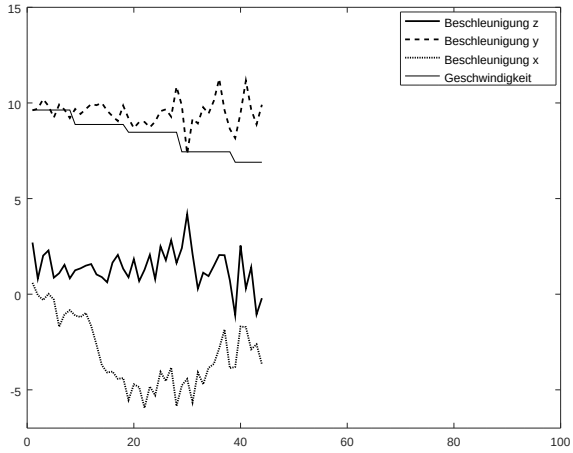
(c) Messreihe 1 Anfahren



(d) Messreihe 1 Normalbremsung

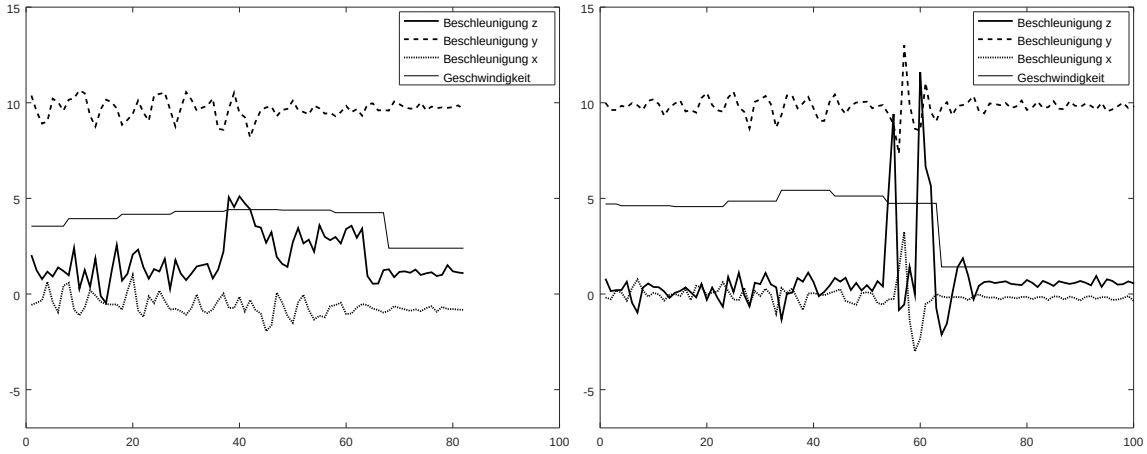


(e) Messreihe 1 Rechtskurve



(f) Messreihe 1 Linkskurve

Abbildung 3.4: Beispiel-Messreihen je Manöver (1) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



(a) Messreihe 1 Normalbremsung (b) Messreihe 1 Vollbremsung

Abbildung 3.5: Beispiel-Messreihen je Manöver (2) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

dere wurden die bei der Durchführung entdeckten Probleme, die es im Zusammenhang mit der App-Bedienung gab, erörtert und Alternativen ausgewertet.

Abschließend wurde dargelegt, wie die Messdaten für die weitere Verarbeitung durch die KNN aufbereitet wurden. Der gesamte experimentelle Aufbau, seine Durchführung und die dabei aufgenommen Daten wurden kritisch betrachtet um Schwachstellen und Grenzen aufzuzeigen, sowohl für die Datenlage selbst auch für die daraus resultierenden Ergebnisse.

4. Klassifikation mit Konzeptoren

Dieses Kapitel widmet sich der Klassifikation der aufgezeichneten Messreihen, die schließlich ein bestimmtes Fahrmanöver widerspiegeln. Unter einer Klasse wird im Folgenden eine Zusammenfassung ähnlicher Messreihen verstanden. Dies bedeutet, dass jedes zuvor definierte Fahrmanöver (Tabelle 3.6) jeweils eine Klasse j repräsentiert. Die entsprechenden Messreihen k bilden die zur Klasse j zugehörigen Elemente (auch Muster genannt) k_j und sind zueinander ähnlich. Dies bedeutet, dass z.B. alle Messreihen, die beim Anfahren aufgenommen wurden, auf Grund bestimmter Eigenschaften (z.B. der Verlauf der Beschleunigungswerte) ähnlich sind und daher der gemeinsamen Klasse (dem Fahrmanöver) Anfahren zugeordnet werden.

Im Rahmen der Klassifikation wird sich mit der Problematik auseinandergesetzt, wie eine unbekannte Messreihe k einer bekannten Klasse j zugeordnet werden kann. Zur Anwendung kommt im Rahmen dieser Arbeit ein Klassifikator, wie er von Jaeger [16] unter Verwendung von Konzeptoren vorgestellt wurde. Folgend werden der allgemeine Aufbau und die Funktionsweise erläutert und anschließend das umgesetzte Projekt zur Klassifikation der Fahrmanöver detailliert vorgestellt.

4.1 Aufbau und Funktionsweise der Klassifikation

Für die Klassifikation der Fahrmanöver wird das von Jaeger [16, S. 74 bis 80] vorgestellte System zur Klassifikation verwendet. Dieses System, basierend auf Konzeptoren, ermöglicht eine leistungsfähige Mustererkennung und Klassifikation auf dem Stand der Technik [16, S. 74]. Ein wesentliches Merkmal dieses Klassifikators besteht darin, dass zu jeder bekannten Klasse ein Konzeptor existiert, der für diese Klasse spezifisch ist und gelernt werden muss. Die Verwendung von Konzeptoren bietet in dieser Architektur eine Besonderheit. Neue Elemente (Messreihen) können gelernt und einer Klasse hinzugefügt werden ohne, dass dieses neue Element im Kontext zu allen anderen Klassen betrachtet werden muss. Es genügt in diesem Fall den Konzeptor der ergänzten Klasse neu zu berechnen (Training). Bereits vorhandene Konzeptoren anderer Klassen bleiben davon unberührt und müssen nicht für das Erlernen des neuen Elements der fremden Klasse herangezogen werden. Ebenso

können vollständig neue, noch unbekannte Klassen dem Klassifikator hinzugefügt werden ohne, dass die neue Klasse und die bereits bekannten Klassen im Kontext zueinander betrachtet werden müssten. Die bekannten Klassen bleiben unberührt. Dennoch können Klassen zu einem späteren Zeitpunkt kombiniert werden [16]. Möglich wird dies durch die für Konzeptoren definierten booleschen Operationen. Somit können Elemente inkrementell einer Klasse hinzugefügt werden und eine neue Klasse kann unabhängig von anderen Klassen definiert und gelernt werden.

Die konkrete Implementierung zur Klassifikation der Fahrmanöver ist eingebettet in ein Experiment, welches vollständig in Octave (Version 4.2.1) [40] implementiert wurde und in Abschnitt 4.2 detailliert beschrieben wird. Es folgt zunächst eine allgemeine Beschreibung zur Funktionsweise der Klassifikation, unabhängig von der konkreten Klassifikationsaufgabe.

4.1.1 Muster und Klassen lernen

Für die Klassifikation wird im Rahmen eines Experimentes zunächst ein Reservoir initialisiert [41]. Es wird für das Lernen und Klassifizieren stets dasselbe Reservoir verwendet. Allerdings wird vor jeder erneuten Verwendung des Reservoirs der initiale Zustand wiederhergestellt, sodass vor jeder Verarbeitung einer Messreihe k der selbe Ausgangszustand existiert [16, S. 75]. Die Gewichtsmatrizen sind auf eins normiert.

Das Reservoir wird wie folgt zur Berechnung eines Konzeptors C_j^+ für die Klasse j verwendet. Das initiale Reservoir erhält als Eingabewert über die Eingabeneuronen den ersten Messpunkt $k_j(1)$ einer Messreihe. Der Messpunkt kann mehrere Werte enthalten, dies ist von der Anzahl der gemessenen Parameter abhängig. Ein Messpunkt einer Messreihe eines Fahrmanövers besteht z.B. aus vier Dimensionen (Formel 4.1). Die Anzahl der Eingabeneuronen entspricht der Anzahl der Dimensionen eines Messpunktes. Mit den anliegenden Messwerten wird mit Hilfe von Formel 4.2 [16, S. 76] der nächste Reservoirzustand $x(n+1)$ berechnet. Ab dem zweiten Messpunkt wird der jeweils zuvor berechnete Reservoirzustand verwendet, nicht mehr das initiale Reservoir. Abhängig von der Anzahl n der Messpunkte, aus der eine Messreihe besteht, werden n Reservoirzustände berechnet. Die Reservoirzustände $x(n)$ und Eingabewerte $k_j(n)$ der Messreihe werden in einem neuen Vektor z zusammengestellt, dessen Aufbau exemplarisch in Formel 4.3 für eine Messreihe dargestellt ist, die aus vier Messpunkten besteht. Der Vektor z enthält untereinander jeweils die Eingabewerte und den Reservoirstatus. Am Beispiel eines Reservoirs mit $N = 10$ Reservoir- und $K = 4$ Eingabeneuronen (jeder Messpunkt besteht aus vier Messwerten), sowie einer Messreihe k mit $n = 4$ Messpunkten, besteht z aus $dim(z) = n * (N + K) = 4 * (10 + 4) = 56$ Dimensionen.

$$k_j(n) = \begin{pmatrix} BeschleunigungZ - Achse(n) \\ BeschleunigungY - Achse(n) \\ BeschleunigungX - Achse(n) \\ Geschwindigkeit(n) \end{pmatrix} \tag{4.1}$$

$$x(n+1) = tanh(W_{Reservoir}x(n) + W_{Eingabe}k(n)) \tag{4.2}$$

$$z_{j,k} = \begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix} \tag{4.3}$$

Für jede Messreihe k einer Klasse j, die zur Trainingsmenge gehört, wird ein entsprechender Vektor $z_{j,k}$ berechnet. Alle Vektoren $z_{j,k}$ einer Klasse j werden zu der Matrix Z_j (Formel 4.4) zusammengefasst. Dadurch ist eine Normalisierung der Messreihen notwendig, sodass $dim(z_{j,k})$ über alle Elemente k gleich groß ist. Auf die Normalisierung wird im Rahmen des Experimentes detaillierter eingegangen (Abschnitt 4.2). Die Matrix Z_j enthält $dim(z_j)$ Zeilen und k Spalten [16, S. 76]. Bezogen auf das vorangegangene Beispiel hätte Z_j bei $k = 5$ Trainings-Messreihen 56 Zeilen und 5 Spalten. Der Aufbau zu diesem Bsp. ist in Formel 4.5 dargestellt. Zusammengefasst enthält Z_j alle gesammelten Netzwerkzustände inkl. der Eingabewerte zu jedem gelernten Element der Klasse j und damit zu jeder gelernten Messreihe eines Fahrmanövers. Z_j wird anschließend für die Berechnung von C_j^+ verwendet, es sollten zur Berechnung von Z_j daher nur die Trainingsdaten verwendet werden.

$$Z_j = [z_{j,1}, \ldots, z_{j,k}] \tag{4.4}$$

$$Z_j = \left(\begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix}_{k=1} \begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix}_{k=2} \begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix}_{k=3} \begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix}_{k=4} \begin{pmatrix} x(1) \\ k_j(1) \\ x(2) \\ k_j(2) \\ x(3) \\ k_j(3) \\ x(4) \\ k_j(4) \end{pmatrix}_{k=5} \right) \tag{4.5}$$

Mit Z_j wird nun die Korrelationsmatrix R_j berechnet. So ergibt sich R_j entsprechend Formel 2.29 zu $R_j = (Z * Z^\top)/k$ [16, S. 76]. Entsprechend des zuvor eingeführten Beispiels hat R_j 56 Zeilen und 56 Spalten. Der vorläufige Konzeptor $\widetilde{C}_j^+$ ergibt sich, da vorläufig $\alpha = 1$ ist, entsprechend Formel 2.31 zu $\widetilde{C}_j^+ = R_j * (R_j + I)^{-1}$ [16, S. 76]. Der Parameter α kann bei Bedarf angepasst werden und dient der Optimierung des Konzeptors, indem mit ihm die Signalstärke des Reservoirs skaliert werden kann [16, S. 43 bis 45]. Abhängig von der Anwendung gibt es diverse Möglichkeiten α für einen Konzeptor zu bestimmen [16, S. 46 bis 50], wobei für die Klassifikation das Gradienten-Kriterium [16, S. 49 f., 77 f.] zur Anwendung kommt. Der endgültige Konzeptor ergibt sich unter Verwendung von Formel 2.33 und Einsetzen von α schließlich zu $C_j^+ = \varphi(\widetilde{C}_j^+, \alpha^+)$ [16, S. 77]. Mit diesem kann der Grad der Klassenzugehörigkeit eines Musters k zu j berechnet werden.

Zur Optimierung der Klassifikation wird ein weiterer Konzeptor $\widetilde{C}_j^-$ zu jeder Klasse j berechnet. Mit diesem wird überprüft, ob ein Muster nicht auch einer anderen Klasse zugeordnet werden kann. Berechnet wird dieser Konzeptor für eine Klasse i, indem alle anderen Konzeptoren $\widetilde{C}_{j\neq i}^+$, mit der booleschen Oder-Operation (Abschnitt 2.5.6) kombiniert werden und das Ergebnis negiert wird. Z.B. würde für $j = 3$ Klassen die Berechnung von $\widetilde{C}^-$ für die zweite Klasse ($i = 2$) wie folgt aussehen: $\widetilde{C}_2^- = \neg \vee \{\widetilde{C}_1^+, \widetilde{C}_3^+\}$ [16, S. 76 f.]. Anschließend wird auch hier eine Optimierung zu $C_j^- = \varphi(\widetilde{C}_j^-, \alpha^-)$ durchgeführt.

Das Lernen einer Klasse j mit k Elementen bedeutet demnach den Konzeptor C_j^+ zu berechnen, der den linearen Unterraum des Reservoirs repräsentiert, in welchem die Hauptaktivität stattfindet, wenn die Klassenelemente (Messreihen) mit dem Reservoir verarbeitet werden. Da ein Konzeptor als Abbildungsmatrix fungiert und ebenso als Identitätsfunktion [33, S. 2] zu einer Reservoiraktivität betrachtet werden kann, kann mit diesen Konzeptoren wie in Abschnitt 4.1.2 erläutert eine Klassifikation vorgenommen werden.

4.1.2 Muster bekannten Klassen zuordnen

Zur Klassifikation wird dasselbe Reservoir wie auch beim Erlernen der Konzeptoren verwendet. Die unbekannten Messreihen (Testdaten) k_{test} müssen, ebenso wie die der Trainingsmenge, normalisiert werden. Die Normalisierung sorgt u.a. dafür, dass schließlich nur Messreihen gleicher Länge vorliegen. Auf die Normalisierung wird in Abschnitt 4.2.2 detaillierter eingegangen. Anschließend werden, wie beim Training, die Reservoirzustände $x(n)$ während der Verarbeitung einer Testmessreihe gespeichert, um schließlich den zugehörigen Vektor z_{test}, wie in Abschnitt 4.1.1 für die Trainingsmessreihen beschrieben, zu generieren.

Dieser Vektor z_{test} wird anschließend, um ein Maß für die Klassenzugehörigkeit zu erhalten, mit allen Konzeptoren C_j^+ in der Form $\widetilde{h}_j^+ = z'_{test}\, C_j^+\, z_{test}$ verrechnet. Die Festlegung zu welcher Klasse j ein Muster k_{test} (repräsentiert durch ein z_{test}) gehört, geschieht über die Identifizierung des dabei größten berechneten Wertes entsprechend Formel 4.6 mit $1 \leq i \leq j_{\max}$ [16, S. 75]). Findet demnach die Reservoiraktivität bei der Verarbeitung der Testdaten von k_{test} zu z_{test} in einem linearen Unterraum des Reservoirs statt, der durch einen Konzeptor C_j^+ beschrieben wird, wird es zu einer größtmöglichen Überlagerung von z_{test} mit diesem, durch C_j^+ beschriebenen, Ellipsoiden (ebenfalls ein linearer Unterraum des Reservoirs) kommen [16, S. 75 f.].

$$j = \text{argmax}_i(z'_{test} C_i^+ z_{test}) = \text{argmax}_i \begin{pmatrix} \widetilde{h}_1^+ \\ \vdots \\ \widetilde{h}_i^+ \\ \vdots \\ \widetilde{h}_{j_{\max}}^+ \end{pmatrix} = \text{argmax}_i(\widetilde{h}^+) \tag{4.6}$$

Der Klassifikator legt schließlich die berechneten Werte für $\widetilde{h}_j^+$ in einem Vektor $\widetilde{h}^+$, wie in Formel 4.7 dargestellt, ab. Dieser Vektor $\widetilde{h}_j^+$ ist als Klassifikationshypothese

zu verstehen [16, S. 77]). Abschließend wird $\widetilde{h}^+$ durch Skalierung zwischen 0 und 1 zu h^+.

Analog zu h^+ wird h^- mit $z'_{test} C_j^- z_{test}$ berechnet. Die Klassenzugehörigkeit ergibt sich dann aus dem Zeilenindex j des Vektors h^+, in dessen Zeile eine 1 steht. Enthält der Vektor h^-, in derselben Zeile wie h^+, den größten Wert, bedeutet dies, dass z_{test} auch nicht zu einer anderen Klasse gehören könnte. Weist h^- dagegen in einer anderen Zeile den größten Wert auf, ist dies ein Indiz für eine mögliche Überschneidung mit einer anderen Klasse, die ebenfalls durch den Zeilenindex bestimmt werden kann [16, S. 77].

$$h = \begin{pmatrix} z'_{test}\, C_{Anfahren}\, z_{test} \\ z'_{test}\, C_{Normalbremsung}\, z_{test} \\ z'_{test}\, C_{Rechtskurve}\, z_{test} \\ z'_{test}\, C_{Linkskurve}\, z_{test} \\ z'_{test}\, C_{Stillstad}\, z_{test} \\ z'_{test}\, C_{Geradeaus}\, z_{test} \\ z'_{test}\, C_{Vollbremsung}\, z_{test} \end{pmatrix} = \begin{pmatrix} z'_{test}\, C_1\, z_{test} \\ z'_{test}\, C_2\, z_{test} \\ z'_{test}\, C_3\, z_{test} \\ z'_{test}\, C_4\, z_{test} \\ z'_{test}\, C_5\, z_{test} \\ z'_{test}\, C_6\, z_{test} \\ z'_{test}\, C_7\, z_{test} \end{pmatrix} \tag{4.7}$$

4.2 Durchführung des Experiments zur Manöver-Klassifikation

Für das Experiment der Klassifikation der Fahrmanöver wurde ein Projekt in Octave (Version 4.2.1) [40] implementiert, welches die in Tabelle 4.1 aufgelisteten Phasen durchläuft. Es werden alle notwendigen Berechnungen, wie sie in Abschnitt 4.1.1 und Abschnitt 4.1.2 beschrieben wurden, durchgeführt. Darüber hinaus werden die Messreihen, sowohl die für Trainings- als auch die für Testzwecke, automatisch normalisiert. Durch den modularen Aufbau und Konfigurationsmöglichkeiten lässt sich das Experiment komfortabel abändern um möglichst variabel arbeiten zu können. So lassen sich Arbeitsschritte einzeln abschalten um z.B. Unittests durchführen zu können. Neue Klassen j sowie Elemente k lassen sich komfortabel hinzufügen und entfernen. Ein Experiment zur Klassifikation lässt sich zudem von der Datenaufbereitung über die Netzwerkerstellung bis hin zur Auswertung der Klassifikation vollautomatisch durchführen. Zu diesem Zweck muss nach erfolgter Konfiguration lediglich das Skript „startExperiment.m“ (Quelltext C.1) ausgeführt werden, welches alle Module nacheinander ausführt. Es werden bei Bedarf ein Protokoll sowie Plots und Reservoirstatus automatisch ausgegeben bzw. abgespeichert. Weitere Informationen zum Octave-Projekt sowie eine detaillierte Beschreibung zu den Unterverzeichnissen und Skripten findet sich in Abschnitt A.2.

4.2.1 Konfiguration des Experiments

Das Experiment ist so implementiert, dass es weitestgehend mit Hilfe der Konfigurationsdatei „cfgExperiment.m“ (Quelltext C.2) gesteuert werden kann. Die Konfigurationsmöglichkeiten und Ausnahmen werden folgend beschrieben.

Alle Klassen j werden in der Variablen „classes“ deklariert. Der Klassenname (z.B. Anfahren) muss mit dem Namen des Unterverzeichnisses im Verzeichnis Samples

Tabelle 4.1: Phasen des Experiments zur Klassifikation (Octave-Projekt)

Nr.	**Name**	**Beschreibung**
1	Datenaufbereitung	Einlesen und Normalisierung aller Messreihen aller Klassen
2	Netzwerkgenerierung	Generieren oder Laden eines Neuronalen Netzes, basierend auf einem Reservoir
3	Reservoirzustände lesen	Auslesen der Reservoirzustände, die bei der Verarbeitung der Messreihen entstehen
4	Konzeptorberechnung	Berechnung der Konzeptoren C_j^+ und C_j^-, die damit verbundenen Matrizen Z_j, R_j, $\widetilde{C}_j^+$ und $\widetilde{C}_j^-$ sowie der Parameter α
5	Test	Test der Klassifizierung durch separate Überprüfung der Trainings- und Testdaten sowie Vergleich mit Erwartungswert und statistischer Auswertung der positiven Klassifizierungen.

übereinstimmen, da die Skripte über diese Bezeichnung den Pfad für das Einlesen der Messreihen automatisch generieren. Weiterhin wird für jede Klasse ein Index vergeben, der später für die Klassenzuordnung verwendet wird. Zu jeder Klasse j wird zudem angegeben, wie viel Elemente k (Messreihen) zu dieser Klasse existieren. Auch diese werden von den Skripten anschließend automatisch eingelesen. Kommen zu einer Klasse weitere Messreihen hinzu, muss in der Konfigurationsdatei lediglich die neue Anzahl hinterlegt werden. Bei zukünftigen Experimenten werden diese dann automatisch mit verwendet. Voraussetzung dafür ist, dass der Dateiname der Messreihe folgende Struktur aufweist: „[Klassenname]_[index,%03d].csv“. Es muss also der Klassenname und eine fortlaufende Nummer mit führenden Nullen, wie z.B. in „Anfahren_001.csv“ vergeben werden. Die Nummerierung muss bei eins beginnen. Die letzte Information die zu jeder Klasse angegeben werden muss, ist die Anzahl der Messreihen, die für das Training verwendet werden sollen. Alle überzähligen Messreihen gehören damit automatisch zur unbekannten Menge der Testdaten. Am konkreten Bsp. von Quelltext C.2 bedeutet dies für die Klasse Linkskurve, dass diese den Index vier hat, es insgesamt 15 Messreihen gibt und nur mit den ersten fünf der Klassifikator trainiert werden soll. Demnach bleiben zehn Messreihen als unbekannte Testdaten über. Diese Art der Datenverwaltung und des Zugriffs darf durchaus kritisch gesehen werden, da ohne weiteren Aufwand immer wieder nur die erste i Messreihen für das Training verwendet werden. Unabhängig von der Gesamtmenge der Messreihen. In Fällen, in denen z.B. lediglich mit den neuesten, also letzten, Messreihen trainiert werden soll, ist die implementierte Lösung ungünstig. Da im Rahmen dieser Arbeit jedoch vielmehr die prinzipielle Anwendbarkeit des Klassifikators, als die optimale Datenverwaltung untersucht werden soll, wird diese einfach handhabbare Methode im Experiment beibehalten. Eine bessere Alternative, z.B. mit einer Datenbank, könnte im Rahmen einer weiteren Testphase oder konkreten Umsetzung berücksichtigt werden.

Sollten weitere Klassen hinzugefügt werden, müssen diese wie zuvor beschrieben in der Konfigurationsdatei hinterlegt werden. Weiterhin müssen im Skript „compute-Conceptors.m“ (Quelltext C.10) die neuen Konzeptoren zu dieser neuen Klasse den booleschen Berechnungen hinzugefügt werden. Diese Stelle ist mit dem Hinweis „in case of new classes add them here“ markiert. Ebenso muss das Skript „testClassification.m“ (Quelltext C.14) angepasst werden. Hier muss hinterlegt werden, welche Klassen getestet werden sollen. Entsprechend müssen hier die Konzeptoren geladen werden. Die Vektoren h^+, h^- und h^{+-} müssen dem Test angepasst werden.

Die Variable „smapleDimensions“ ist für eine variable Anpassung der Messreihen gedacht, wenn diese aus Messpunkten mit mehr als vier Dimensionen bestehen. Grund dafür war die Überlegung, auch Messreihen analysieren zu können, in denen weitere Daten hinterlegt sind (z.B. GPS-Positionen). Allerdings wurde im Rahmen der Auswertung diese Möglichkeit nicht weiter verwendet und verfolgt, sodass diese Parametrierung veraltet ist und der Wert konstant bei vier belassen werden muss.

Die Variable „N“ steuert die Anzahl der Reservoirneuronen. Da die Netzwerke über das Skript automatisch generiert werden können wenn „createNewNetwork“ true ist, können hierüber Experimente mit verschieden großen und verschieden strukturierten (durch zufällige Neugenerierung) Reservoiren komfortabel durchgeführt werden. Die Zustandsvektoren z_j sowie Matrizen Z_j, Korrelationsmatrizen R_j und Konzeptoren C_j^+ sowie C_j^- werden automatisch angepasst. Ein einmal erstelltes Netzwerk wird im Projektverzeichnis abgespeichert. Es kann so (manuell) archiviert und für zukünftige Experimente verwendet werden. Ist „createNewNetwork“ dagegen false, wird kein neues Netzwerk generiert, sondern das im Projektverzeichnis hinterlegte verwendet. Die Anzahl der Neuronen muss dennoch angegeben werden. Auch hier bestehen wieder Möglichkeiten der Optimierung zur Netzwerkverwaltung, die hier nicht weiter erläutert werden sollen, für zukünftige Analysen mit deutlich größeren Datenengen und Szenarien interessant sein könnten.

Über die Variable „makePlots“ wird gesteuert, ob Plots generiert werden sollen. Diese Option wurde implementiert, da nicht zu jedem Durchlauf Plots benötigt werden, dessen Generierung aber durchaus signifikante Rechenzeit in Anspruch nimmt.

Mit „saveWout“ kann festgelegt werden, ob die Ausgabegewichte des Reservoirs als CSV-Datei gespeichert werden sollen.

4.2.2 Datenaufbereitung

Das Experiment beginnt mit dem Skript „samplePreprocessing.m“ (Quelltext C.3) zur Normalisierung der Daten [16, S. 74]. Diese wird über die in Tabelle 4.2 aufgelisteten Schritte erreicht.

Im Skript „samplePreprocessing.m“ wird zunächst das Skript „searchMinAndMax.m“ (Quelltext C.4) ausgeführt, um in allen Messreihen k aller Klassen j nach dem kleinsten und größten Wert zu suchen (Tabelle 4.2, Phase 1). Anschließend werden die Werte identifiziert, die über alle Klassen j den kleinsten und größten Wert darstellen. Dies wird separat für jede Dimension durchgeführt.

Die (jeweils vier) bekannten Minimal- und Maximalwerte werden im Skript „subsampling.m“ (Quelltext C.5) für die Normalisierung (Tabelle 4.2, Phase 2) der Daten

Tabelle 4.2: Phasen des Datennormalisierung

Nr.	Name	Beschreibung
1	Analyse	Suchen nach Minimal- und Maximalwerten innerhalb der Messreihen.
2	Normalisierung	Alle Messreihen werden im Bereich zwischen 0 und 1 skaliert.
3	Polynominterpolation	Annäherung der Messwertkurve durch ein Polynom dritten Grades.
4	Stützpunktberechnung	Berechnung von vier Stützpunkten mit Hilfe des Polynoms.

verwendet. Zunächst wird allen Messwerten der minimale Wert, ist er denn kleiner als Null, als Offset hinzuaddiert. Dadurch sind nun alle Werte größer als Null („shifting“) . Es folgt die Normierung aller Werte im Bereich von 0 und 1 („scaling“) .

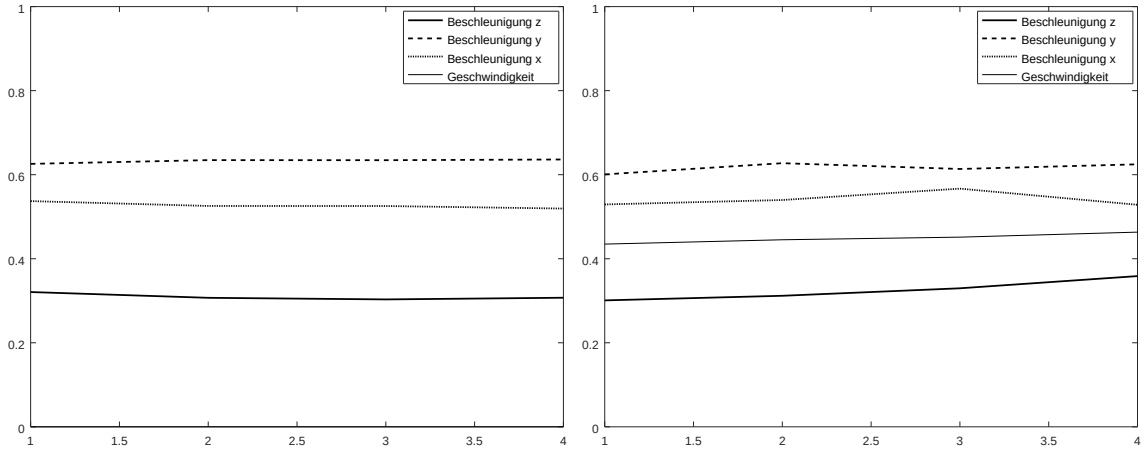
Für die Polynominterpolation (Tabelle 4.2, Phase 3) werden aus jeder Dimension einer jeden Messreihe k vier äquidistante Messpunkte entnommen. Mit diesen wird ein lineares Gleichungssystem erstellt und gelöst, um so die Koeffizienten des Polynoms berechnen zu können. Durch diese Interpolation und Wahl äquidistanter Messpunkte wird nicht nur eine Normalisierung der Messwerte in ihrem Betrag erreicht, sondern auch eine zeitliche Stauchung bzw. Streckung. Unterschiedlich lange Messreihen werden so in ihrer zeitlichen Länge normalisiert.

Abschließend (Tabelle 4.2, Phase 4) werden mit Hilfe des Polynoms vier neue Stützpunkte berechnet. Diese werden mit dem Präfix „pp_“ abgespeichert und für die folgende Klassifizierung verwendet. Die originalen Messreihen werden im folgenden Verlauf nicht weiter verwendet. In Abbildung 4.1 und Abbildung 4.2 sind zu jedem Fahrmanöver je eine normalisierte Messreihe dargestellt, die jeweils nur noch aus vier Messpunkten bestehen. Es handelt sich bei den dargestellten Messreihen um jene, die bereits in Abbildung 3.4 und Abbildung 3.5 in ihrer ursprünglichen Form abgebildet wurden. Eine direkte Gegenüberstellung aller sieben Fahrmanöver aus Abbildung 3.4 und Abbildung 3.5 mit ihren normalisierten Messreihen ist in Abbildung B.1, Abbildung B.2 und Abbildung B.3 abgebildet.

Da die normalisierten Messreihen abgespeichert werden, kann diesen Modul, wenn sich die Datenbasis der Messreihen nicht ändert, für weitere Experimente deaktiviert werden.

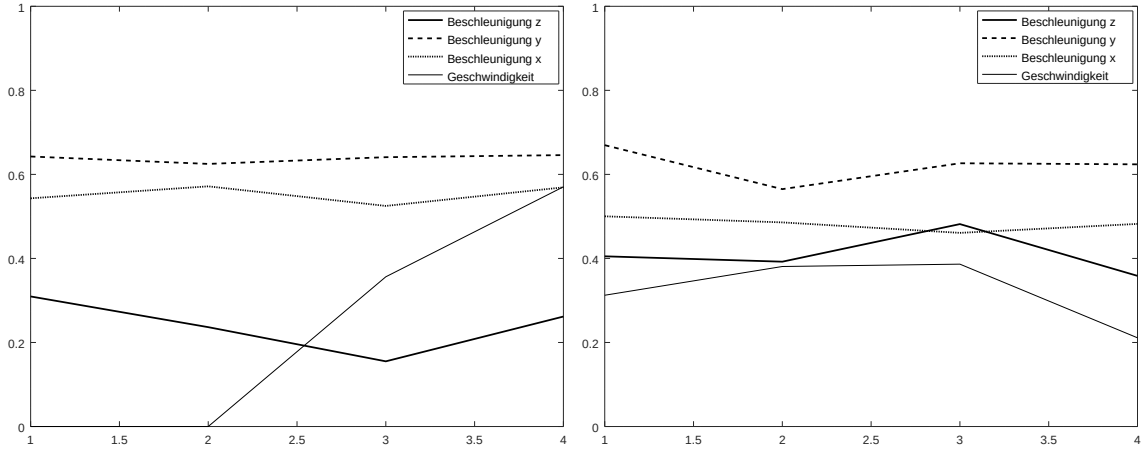
4.2.3 Netzwerk generieren

Phase zwei des Experiments (Tabelle 4.1 (2)) wird über das Skript „createReservoir.m“ (Quelltext C.6) gesteuert. Es wird zunächst von jeder Klasse j eine Messreihe k geladen, mit der eine erste Einschätzung zur Qualität des generierten Reservoirs gemacht werden kann. Zu diesem Zweck werden für jede Messreihe die Ausgabegewichte W_{out} des Reservoirs gelernt (siehe Abschnitt 2.5.4). Anschließend wird mit diesen und dem Reservoir die Messreihe reproduziert und mit der originalen Messreihe verglichen, indem der NRMSE berechnet wird. Dafür wurde das entsprechende



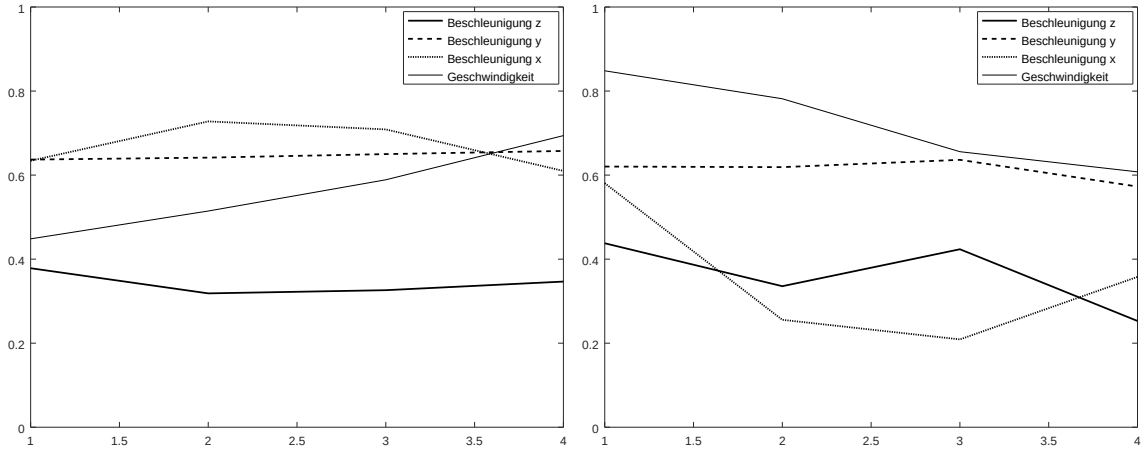
(a) Messreihe 1 Stillstand

(b) Messreihe 1 Geradeaus



(c) Messreihe 1 Anfahren

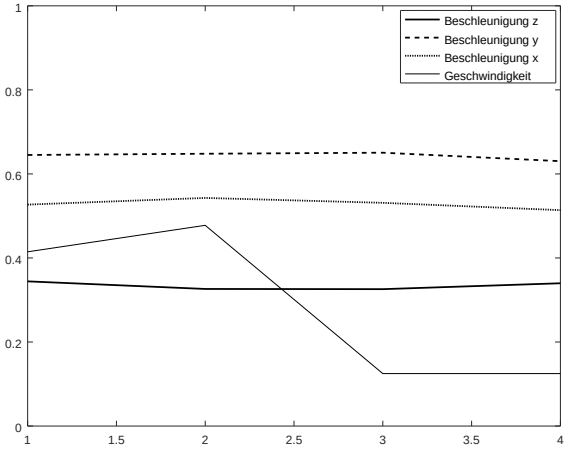
(d) Messreihe 1 Normalbremsung



(e) Messreihe 1 Rechtskurve

(f) Messreihe 1 Linkskurve

Abbildung 4.1: Normalisierte Beispiel-Messreihen je Manöver (1) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



(a) Messreihe 1 Vollbremsung

Abbildung 4.2: Normalisierte Beispiel-Messreihe je Manöver (2) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

Skript (Quelltext C.7) von Prof. Dr. Stolzenburg zur Verfügung gestellt. Entstehen Netzwerke, bei denen der Fehler nicht deutlich kleiner eins ist, wird das Experiment verworfen. Bei Bedarf können entsprechende Plots zum Vergleich der Messreihen gespeichert werden.

Das eigentliche Anlegen des Reservoirs wird über das Script „predict.m“ (Quelltext C.8) realisiert, welches ebenfalls von Prof. Dr. Stolzenburg zu Verfügung gestellt wurde und aus dem Projekt zu den PrNN [1] stammt. In dem Skript wird das Reservoir mit der zuvor festgelegten Größe N (Anzahl Neuronen) generiert, indem die Gewichte über die in Octave implementierte Funktion „randn()“ initialisiert werden. Diese Funktion liefert eine Matrix mit zufälligen, normal verteilten Elementen, deren Mittelwert null und Varianz eins ist. Lediglich die, ebenfalls zufällig generierten, Eingabegewichte werden ausbalanciert. Die Anzahl der generierten Eingabegewichte entspricht der Anzahl der Dimensionen eines Messpunktes, womit das Gesamtnetz über vier Eingabeneuronen verfügt. Da das Experiment es auch erlauben soll, bereits generierte Netzwerke erneut verwenden zu können, wurde das Skript dahingehen modifiziert, dass Netze auch gespeichert und geladen werden können. Darüber hinaus werden mit diesem Skript auch die Reservoirstatus ausgelesen, um die benötigten Konzeptoren berechnen zu können.

4.2.4 Reservoir auslesen

Die dritte Phase des Experiments (Tabelle 4.1 (3)) wird über das Skript „readReservoirStates.m“ (Quelltext C.9) gesteuert. Es werden alle normalisierten Messreihen k aller Klassen j geladen, um diese mit Hilfe des Skripts „predict.m“ (Quelltext C.8) im Reservoir verarbeiten zu können. Es wird so zu jedem der vier interpolierten Messpunkte einer jeden normalisierten Messreihe ein Reservoirstatus $x(n)$ mit $1 \leq n \leq 4$ berechnet. Dies darf nicht verwechselt werden mit den vier Dimensionen, aus denen ein Messpunkt besteht. Pro Messpunkt werden alle vier Dimensionen im selben Reservoir, welches über vier Eingabeneuronen verfügt, gleichzeitig verarbeitet.

Die ausgelesenen Reservoirstatus werden als Matrix der Form $X = [x(1), \ldots, x(4)]$ zusammengefasst und als CSV-Datei gespeichert. Zusätzlich wird zu jeder Messreihe wieder eine Messreihe mit Hilfe des Neuronalen Netzes reproduziert und mit der originalen normalisierten Messreihe verglichen. Es wird der maximale Fehler, der bei der Verarbeitung aller Messreihen auftrat, je Klasse ermittelt. Dies dient wieder einer möglichen Kontrolle, um ein ungeeignetes Reservoir identifizieren zu können.

4.2.5 Berechnung der Konzeptoren

Phase vier des Experiments (Tabelle 4.1 (4)) wird über das Skript „computeConceptors.m“ (Quelltext C.10) gesteuert, wobei im wesentlichen die in Tabelle 4.3 aufgeführten Schritte durchgeführt werden.

Tabelle 4.3: Phasen der Konzeptorberechnung

Nr.	**Name**	**Beschreibung**
1	Konzeptorberechnung	Berechnung der Reservoirzustandsmatrix Z_j, der Korrelationsmatrix R_j sowie des vorläufigen Konzeptors $\widetilde{C}_j^+$ entsprechend Abschnitt 4.1.
2	Konzeptoroptimierung	Berechnung des optimalen Parameter α zur Optimierung des Konzeptors $\widetilde{C}_j^+$ entsprechend Abschnitt 4.1.
3	Konzeptorkombination	Berechnung des Konzeptors C_j^- entsprechend Abschnitt 4.1

Die Berechnung der Konzeptoren entsprechend Tabelle 4.3 Phase 1 wird durch das Skript „computePreliminaryConceptor“ (Quelltext C.11) realisiert. Hier werden über ein weiteres Skript „createCombinedStateVector“ (Quelltext C.12) zunächst für jede Messreihe k die zuvor berechneten Reservoirstatus $x(n)$ und die Messpunkte der normalisierten Messreihe $s(n)$ geladen. Der Vektor $s(n)$ enthält, entsprechend der Dimensionen eines Messpunktes, vier Messwerte der Messreihe k zum Zeitpunkt n. Mit diesen Vektoren wird der Reservoirzustandsvektor $z_{j,k}$ der Form $z_{j,k} = [x(1); s(1); x(n); s(n); \ldots; x(4); s(4)]$ zusammengesetzt. Alle $z_{j,k}$ einer Klasse j werden anschließend spaltenweise zu einer Matrix Z_j (Formel 4.4) zusammengefasst.

Es folgt die Berechnung der Korrelationsmatrix R_j und dessen Normierung, indem durch die Anzahl der in Z_j enthalten Messreihen, ablesbar durch die Spalten von Z_j, dividiert wird (Formel 2.29). Schließlich wird der vorläufige Konzeptor $\widetilde{C}_j^+$ mit Formel 2.31 berechnet $\alpha = 0$ ist.

Zur Optimierung des Konzeptors (Tabelle 4.3 (2)) wird schließlich die Berechnung von α im Skript „computeBestAperture“ (Quelltext C.13) durchgeführt welches auf einen Ausschnitt des Skript „fig24and25and26_JapVowels_Report.m“ [42] ab Zeile 282 basiert. Es implementiert die Optimierung auf Basis des Gradienten-Kriteriums [16, S. 49 f.]. Es werden zunächst zehn Stützpunkte φ nach Formel 4.8 mit $0 \leq i \leq 9$, $i \in N$ berechnet. Mit diesen werden weitere Werte über interpoliert und die Stelle des größten Anstiegs ermittelt. Der finale Konzeptor C_j^+ wird nun mit dem ermittelten Wert und Formel 2.31 berechnet.

$$\varphi_i = \|R(R + \alpha_i^{-2} I)^{-1}\|^2_{FrobeniusNorm} = \|R(R + 2^{i^{-2}} I)^{-1}\|^2_{FrobeniusNorm} \tag{4.8}$$

Abschließend werden zu jeder Klasse j die Konzeptoren C_j^- unter Verwendung der booleschen Operationen (Abschnitt 2.5.6) entsprechend der in Abschnitt 4.1.1 erläuterten Kombination berechnet. Alle Matrizen werden wieder in CSV-Dateien gespeichert.

Ebenfalls berechnet werden die Konzeptoren „CnotAny“ und „CorAll“ die auf Basis der booleschen Operationen (Abschnitt 2.5.6) Klassen zusammenfassen, bzw. ausschließen. Da im Rahmen dieser Arbeit diverse Implementierungen und Versuche durchgeführt wurden, enthält das Skript Quelltext C.10 die notwendigen Berechnungen von „CnotAny“ und „CorAll“ auf Basis von Korrelationsmatrizen und Konzeptoren. Für die späteren Auswertungen und in dieser Arbeit vorgestellten Ergebnisse wurden „CnotAny“ und „CorAll“ ausschließlich auf Basis der Konzeptoren berechnet.

Der Konzeptor „CnotAny“ soll den linearen Unterraum repräsentieren, der keinem Konzeptor einer der bekannten Klasse zugeordnet werden kann. Dafür wurde die Vereinigungsmenge aller Konzeptoren C_j mit $j = 1 \dots 7$ gebildet und negiert (Formel 4.9). Zur einfacheren Implementierung, Wartung sowie Tests wurde Formel 4.9 durch Anwendung der De Morganschen Gesetze in Formel 4.10 umgerechnet, sodass nun nicht mehr Formel 2.36, sondern Formel 2.40 angewendet werden konnte.

$$C_{notAny} = \overline{C_1 \vee C_2 \vee C_3 \vee C_4 \vee C_5 \vee C_6 \vee C_7} \tag{4.9}$$

$$C_{notAny} = \overline{C_1} \wedge \overline{C_2} \wedge \overline{C_3} \wedge \overline{C_4} \wedge \overline{C_5} \wedge \overline{C_6} \wedge \overline{C_7} \tag{4.10}$$

Der Konzeptor „CorAll“ repräsentiert hingegen den linearen Unterraum, der aus der Vereinigungsmenge aller Konzeptoren hervorgeht (Formel 4.11). Mit Hilfe der De Morganschen Gesetze lässt sich zeigen, dass die Negation von Formel 4.10 in Formel 4.11 umgeformt werden kann. Danach ergibt sich „CorAll“ durch die Negation von „CnotAny“ (Formel 4.12).

$$C_{orAll} = C_1 \vee C_2 \vee C_3 \vee C_4 \vee C_5 \vee C_6 \vee C_7 \tag{4.11}$$

$$C_{orAll} = \overline{C_{notAny}} \tag{4.12}$$

Die Konzeptoren „CnotAny“ und „CorAll“ werden später verwendet, um zu prüfen wie der Klassifikator mit unbekannten Klassen umgeht. Wäre z.B. die Klasse „Linkskurve“ unbekannt, würde man deren Konzeptor (C_4) nicht berechnen und bei den Berechnungen von „CnotAny“ und „CorAll“ entsprechend entfernen.

4.2.6 Test der Klassifizierung

Den Abschluss des Experiments bildet das Skript „testClassification.m“ (Quelltext C.14) mit der Überprüfung, ob mit Hilfe der Konzeptoren C_j^+ unter Verwendung von Formel 4.6 eine Klassifizierung für alle Muster k vorgenommen werden kann.

Zunächst werden zu einer Klasse j alle Reservoirzustandsvektoren $z_{j,k}$, die jeweils für je ein Muster k_j berechnet wurden, geladen. Es wird nun der Grad der Übereinstimmung des linearen Unterraums des Reservoirs, repräsentiert durch $z_{j,k}$, mit jedem Konzeptor C_j^+ einer jeden Klasse j berechnet. Dabei repräsentiert jeder C_j^+ ebenfalls einen Unterraum desselben Reservoirs. Genauer den Unterraum des Reservoirs, der durch den Ellipsoiden beschrieben wird (Abschnitt 2.5.6) und einer Klasse j zugeordnet ist. Auf diese Weise erhält man den Grad an Übereinstimmung eines $z_{j,k}$ mit jedem C_j^+ (insgesamt sieben). Diese sieben Werte werden zusammengefasst als Vektor h^+ gespeichert und skaliert, sodass alle Werte im Intervall $[0, 1]$ liegen (Abschnitt 4.1.2). Es wird nun die Zeile i von h^+ ermittelt, in der der größte Wert (eins) steht. Der Konzeptor C_j^+, mit dem der Wert in Zeile i von h^+ berechnet wurde, weist die größte Überschneidung mit $z_{j,k}$ auf. Daher wird $z_{j,k}$ der Klasse zugeordnet, zu der auch der entsprechende Konzeptor C_j^+ gehört.

Im Experiment wird der Vektor h^+ so aufgebaut, dass dessen Zeilennummer mit dem Index der Klasse übereinstimmt. Damit ist bei der Klassifizierung die Bestimmung der Klasse direkt über die Zeilennummer möglich. Befindet sich z.B. der größte Wert von h^+ in Zeile zwei, wird das entsprechende Muster der Klasse „Normalbremsung“ zugeordnet, deren Index zwei ist.

Die Berechnung des Vektors h^- geschieht analog zu der Berechnung von h^+, mit der Ausnahme, dass jeweils C_j^- zur Berechnung verwendet wird. Die Interpretation von h^- ist jedoch eine andere. Hier bedeutet die Zuordnung von $z_{j,k}$ zu einem Konzeptor C_j^- und damit zu einer Klasse j, dass der lineare Unterraum von $z_{j,k}$ auch nicht wesentlicher Bestandteil irgendeines anderen linearen Unterraums (C_j^+) einer anderen Klasse ist. Damit bleibt per Ausschlussverfahren nur der bekannte Unterraum von C_j^- übrig (Abschnitt 4.1.1). So würde, stünde der größte Wert von h^- wieder in Zeile zwei, es für ein Muster k bedeuten, dass es nicht einer anderen, von „Normalbremsung“ verschiedenen, Klasse zugeordnet werden konnte.

Auf diese Weise werden alle Muster einer Klasse zugeordnet. Der Test untersucht dabei die Muster, die für das Training (der Berechnung der Konzeptoren) verwendet wurden, separat. Er weist die Anzahl der erfolgreichen Klassifizierungen entsprechend für jedes Muster der Trainingsdatenmenge und jedes Muster Testdatenmenge einzeln aus. Ebenso wird die Menge erfolgreicher Klassifizierungen zu beiden Mengen separat für h^+ und h^- prozentual einzeln berechnet. Eine Auswertung zu erfolgreichen Klassifikationen für h^+ und h^-, über Trainings- und Testdaten, kumuliert erfolgt zum Ende des Experiments.

4.3 Zusammenfassung

In diesem Kapitel wurde zunächst der auf Konzeptoren und einem Reservoir (Abschnitt 2.5.6 und Abschnitt 2.5.4) basierende Klassifikator vorgestellt, wie er von

Jaeger [16, S. 74 bis 80] entwickelt wurde. Die Idee der Klassifikation beruht schließlich darauf, zwei lineare Unterräume desselben Reservoirs miteinander zu vergleichen. Einmal der Unterraum, der durch einen Konzeptor und dem entsprechenden Ellipsoiden beschrieben wird und jenen, der bei der Verarbeitung eines Musters im Reservoir identifiziert wird. Die Zuordnung geschieht über die Identifizierung der größten Überlagerung mit einem Konzeptor einer Klasse.

Weiterhin wurde in Abschnitt 4.1.1 erläutert, wie das Erlernen einzelner Muster realisiert wird, aus denen dann schließlich ein Konzeptor (linearer Unterraum) der Klasse berechnet werden kann. Ebenso vorgestellt wurde das Verwenden mehrerer Klassen und wie diese über die booleschen Operationen kombiniert werden, um im Klassifikator verwendet werden zu können.

In Kapitel Abschnitt 4.1.2 wurden der Algorithmus und die notwendigen Berechnungsvorschriften erläutert, die für die Berechnung notwendig sind, soll der Grad an Übereinstimmung zweier linearer Unterräume ermittelt werden.

Abschließend wurde das in Octave implementierte Experiment vorgestellt (Abschnitt 4.2). Der Aufbau sowie die einzelnen Module für die Aufbereitung der Daten, Berechnung der Konzeptoren bis hin zum Test der Klassifikation wurden aufgezeigt.

5. Ergebnisse und Analyse der Klassifikation von Fahrmanövern

Mit dem, in Kapitel Kapitel 4 beschriebenem Experiment wurden mehrere Testläufe mit variierenden Parametern durchgeführt. Es wurden die Anzahl der Neuronen als auch die Anzahl der Messreihen verändert, die für das Training verwendet wurden. Ziel war es, die Anwendbarkeit des Klassifikators für die Zuordnung von Messreihen zu bekannten Fahrmanövern, zu untersuchen. Gleichzeitig sollte untersucht werden, welchen Einfluss die Anzahl der Reservoirneuronen als auch die Anzahl der Trainingsmessreihen auf die Klassifikation haben. Weiterhin wurde auf experimenteller Basis untersucht, wie der Klassifikator mit Messreihen umgeht, die zu einer unbekannten Klasse gehören und zuvor nicht gelernt werden konnte.

Die Einzelnen Experimente sowie deren Ergebnisse werden in diesem Kapitel im Detail beschrieben. Der Aufbau und Ablauf folgt dabei stets den Erläuterungen in Abschnitt 4.2. Die Konfigurationen und Modifikationen, sofern diese vom Standard abweichen, werden an entsprechender Stelle in den folgenden Abschnitten erläutert.

5.1 Klassifikation der bekannten Fahrmanöver

In diesem Abschnitt werden die Experimente erläutert, bei denen die Anzahl der Reservoirneuronen und die Größe der Trainingsmenge verändert wurden. Dabei wurden Messreihen aus allen Klassen (Tabelle 3.6) verwendet, sodass jede Klasse gelernt werden konnte und keine unbekannte Klasse zu identifizieren war.

5.1.1 Variation der Anzahl von Reservoirneuronen

Ziel dieses Experiments war es zu klären, inwieweit sich die Anzahl der Neuronen auf die Güte der Klassifikation auswirkt. Als Gütekriterium wird die Anzahl richtiger Zuordnungen von Messreihen k zu ihrer Klasse j in Prozent (%) gewählt. Eine richtige Zuordnung aller Messreihen entspricht einer Erfolgsquote von 100% und damit

der maximalen Güte. Wird keine Messreihe richtig zugeordnet, liegt die Erfolgsquote bei 0% und damit minimaler Güte.

Bei diesem Experiment wurde die Anzahl der Neuronen N der verwendeten Reservoire für verschiedene Testläufe variiert. Entsprechend Tabelle 5.1 wurden Reservoire mit den Größen $N = \{4, 6, 8, 10, 20, 30, 40, 60\}$ generiert und verwendet.

Ein Testlauf entsprach dabei grundlegend dem in Abschnitt 4.2 beschriebenen Ablauf. Jedoch wurden die Messreihen nur einmal aufbereitet (Nr. 1 in Tabelle 4.1) und für alle weiteren Experimente verwendet. Die Netzwerkgenerierung, das Auslesen der Reservoirzustände, die Konzeptorberechnung sowie der Test der Klassifikation (Phasen von Nr. 2 bis Nr. 5 in Tabelle 4.1) wurden dagegen für alle Reservoire durchgeführt. Für jede mögliche Reservoirgröße N wurden jeweils 100 zufällige Reservoire generiert und getestet. Pro Testlauf und Reservoir wurde für jede Klasse j ein Konzeptor berechnet.

Zur Berechnung der Konzeptoren, also dem Training des Netzes, wurden stets die ersten acht Messreihen k einer jeden Klasse j für die Trainingsmenge verwendet. Diese Anzahl entspricht der maximal möglichen Anzahl der zur Verfügung stehenden Trainingsmenge, da für die Klassen Normal- und Vollbremsung insgesamt nur acht Messreihen zur Verfügung stehen (Tabelle 3.6). Um eine bessere Vergleichbarkeit der Klassifikation einzelner Klassen zu erreichen, wurde für jede Klasse die gleiche Anzahl an Messreihen für das Training verwendet. Da bei dieser Konfiguration für die Klassen Normal- und Vollbremsung keine Messreihen für eine Testmenge mehr vorhanden sind, wird zunächst nur die Klassifikation der Trainingsmenge betrachtet.

Die Ergebnisse der Testläufe sind in Tabelle 5.1 aufgelistet. Die Werte sind wie folgt zu lesen. Je Spalte stehen die Ergebnisse zur Güte der Klassifizierung abhängig von der verwendeten Reservoirgröße N. Jeder Wert innerhalb einer Spalte basiert auf einer Auswertung von 100 Netzwerken der jeweiligen Größe N und einer ausgewerteten Klassifizierung von 8 Messreihen der Trainingsmenge. Demnach wurden je Klasse j und Reservoirgröße N 800 Klassifizierungen zur Trainingsmenge durchgeführt. Innerhalb einer Zeile sind die Werte zu jeder Klasse spaltenweise abzulesen, wobei zu jeder Klasse separat die Werte für $h+$ und $h-$ aufgelistet sind. Für die Klasse Normalbremsung bedeutet dies bei einer Reservoirgröße von $N = 4$, dass mit einer Güte von 99,8% bei $h+$ 798 von 800 Klassifizierungen richtig waren. Bei $h-$ wurde dagegen nur eine Güte von 10,8% (86 von 800) erreicht. Bei lediglich 86 Klassifizierungen konnte eine Trainingsmessreihe nicht auch anderen Klassen zugeordnet werden. Dies bedeutet, dass bei 89,2% (714 von 800) der Klassifizierungen die Messreihe tendenziell auch einer anderen Klasse zugeordnet werden könnte.

Die Ergebnisse in Tabelle 5.1 zeigen, dass bereits mit wenigen Neuronen (z.B. $N = 2$) sehr gute Ergebnisse erzielt werden können. So etwa für die Klasse Stillstand, bei der bereits bei $N = 2$ eine Güte von 100% erreicht wird. Ein anderes Ergebnis liefert dagegen die Klassifizierung der Klassen Geradeaus und Vollbremsung. Für das Manöver Geradeaus wird erst ab $N = 8$ Reservoirneuronen eine Güte von mehr als 90% erreicht. Dagegen wird diese Güte für das Manöver Vollbremsung auch mit sehr großen Reservoiren nicht erreicht und beträgt maximal 64,3%. Für die sichere Überprüfung, ob eine Messreihe aus der Trainingsmenge nicht auch einer anderen Klasse zugeordnet werden kann, überprüfbar mit $h-$, sind dagegen mehr Neuronen notwen-

Tabelle 5.1: Klassifikation der Trainingsmenge

		Reservoirgröße N								
Manöver		**2**	**4**	**6**	**8**	**10**	**20**	**30**	**40**	**60**
Anfahren	h+	99,9	100	100	99,9	100	100	99,9	100	100
	h-	2,9	11,1	38,4	50,1	51,0	60,1	69,5	79,6	82,0
Normalbr.	h+	99,9	99,8	99,3	98,8	99,3	99,6	99,8	99,9	100
	h-	1,0	10,8	38,6	39,6	41,3	39,1	41,5	44,1	49,4
Rechtskurve	h+	99,8	99,9	99,3	98,9	98,0	98,4	96,5	96,1	95,5
	h-	78,4	92,6	97,9	98,0	96,4	100	100	100	100
Linkskurve	h+	98,5	98,1	97,5	97,8	96,8	99,3	98,9	99,3	99,8
	h-	36,1	36,9	79,4	92,6	93,9	98,0	100	100	100
Stillstand	h+	100	100	100	100	100	100	100	100	100
	h-	51,6	29,0	79,0	94,0	94,0	98,0	100	100	100
Geradeaus	h+	36,1	62,4	89,3	93,0	99,4	100	100	100	100
	h-	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Vollbr.	h+	31,0	34,3	40,4	45,3	47,1	54,4	59,5	60,8	64,3
	h-	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

dig. Mit lediglich $N = 2$ Neuronen könnten die Messreihen der Klassen Anfahren, Normalbremsung, Geradeaus und Vollbremsung auch anderen Klassen zugeordnet werden. Dies bedeutet, sie enthalten auch Merkmale, die in anderen Klassen zu finden sind. Eine schärfere Abgrenzung wird erst wieder mit größeren Reservoiren erreicht. Ausgenommen hiervon sind wieder die Klassen Geradeaus und Vollbremsung. Eine mögliche Ursache könnte in einer zu geringen Anzahl von Merkmalen liegen, in der sich die Vollbremsung von anderen Klassen unterscheidet. Ebenfalls denkbar ist ein signifikanter Einfluss auf die Klassifikation, der durch die vorgelagerte Datenaufbereitung verursacht wird.

Ausgehend von diesen Ergebnissen bei der Klassifizierung der Vollbremsungen wurde die Zuordnung der einzelnen Messreihen dieser Klasse noch einmal separat betrachtet. Die Ergebnisse sind in Tabelle 5.2 und Tabelle 5.3 zusammengefasst. Eine leere Zelle bedeutet, dass der entsprechende Wert 0 ist. Zur besseren Lesbarkeit wurden jedoch nur Werte aufgenommen die größer als 0 sind. Alle Werte beruhen auf der Auswertung von 100 Testläufen je Reservoirgröße N, in denen jeweils ein neues zufälliges Netzwerk generiert wurde. Für jede der acht Messreihen aus der Trainingsmenge wurden die Klassenzuordnungen bei einer Reservoirgröße von $N = \{4, 10, 60\}$ separat ausgewertet. So ist z.B. für die Messreihe 1 der Vollbremsung zu erkennen, dass bei allen Reservoirgrößen N beinahe alle Klassifizierungen falsch waren und immer der Klasse Normalbremsung ($j = 2$) zugeordnet wurden. Ebenso wird Messreihe 4 beinahe ausschließlich falsch klassifiziert, nämlich als Anfahren ($j = 1$). Messreihe 6 wird ebenfalls vorwiegend falsch klassifiziert. Die Messreihen 5 und 7 werden erst mit sehr großen Netzen ($N = 60$) richtig klassifiziert.

Da die Klassifikation der Vollbremsung grundsätzlich möglich ist, was die Ergebnisse in Tabelle 5.1 gezeigt haben, lediglich einzelne Messreihen fast ausschließlich falsch

Tabelle 5.2: Klassifikationen der Vollbremsungen ($n = 8$, Messreihen 1 bis 4)

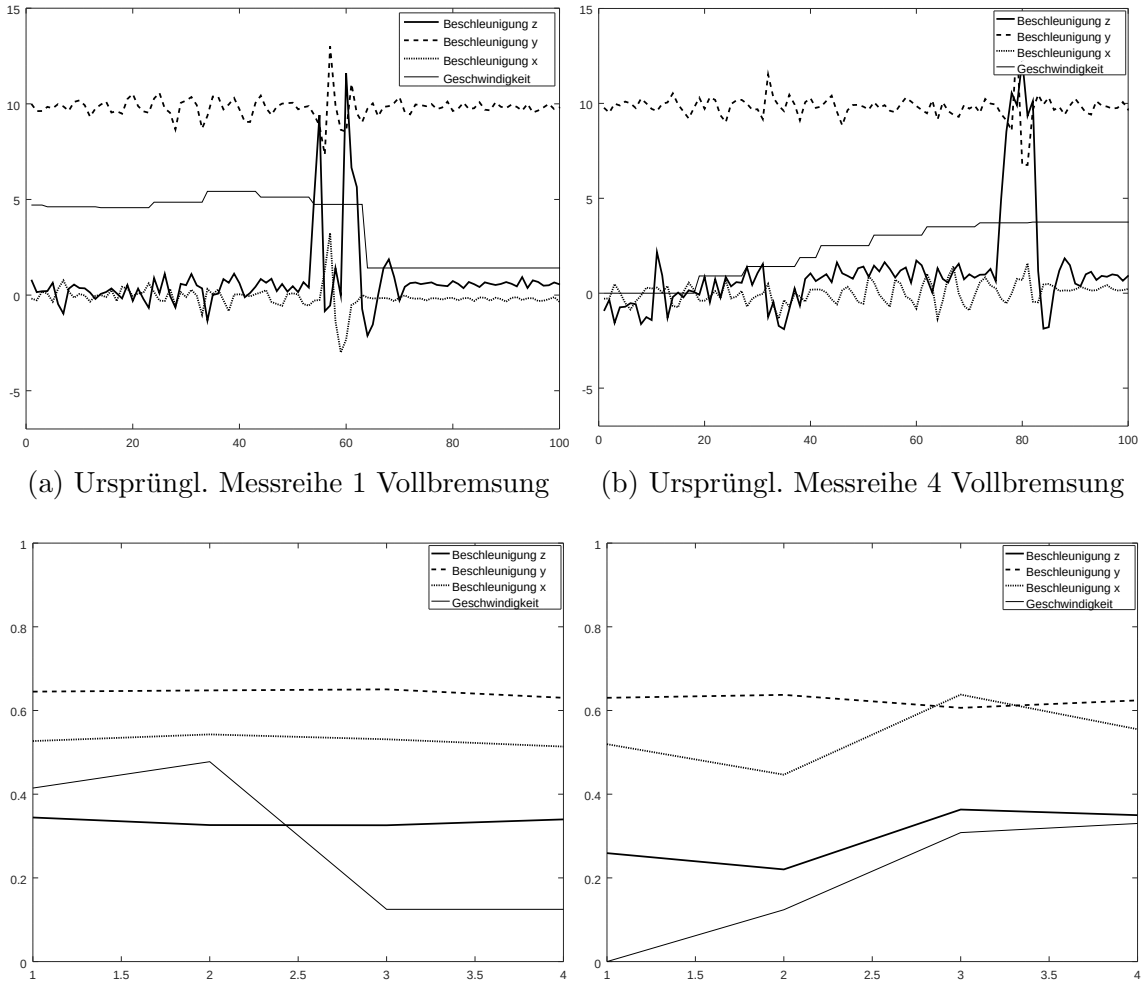
		Messreihe 1			**Messreihe 2**			**Messreihe 3**			**Messreihe 4**		
N		**04**	**10**	**60**	**04**	**10**	**60**	**04**	**10**	**60**	**04**	**10**	**60**
j	1										96	98	100
	2	100	98	99									
	3												
	4												
	5												
	6							97	79				
	7		2	1	100	100	100	3	21	100	4	2	

Tabelle 5.3: Klassifikationen der Vollbremsungen ($n = 8$, Messreihen 5 bis 8)

		Messreihe 5			**Messreihe 6**			**Messreihe 7**			**Messreihe 8**		
N		**04**	**10**	**60**	**04**	**10**	**60**	**04**	**10**	**60**	**04**	**10**	**60**
j	1												
	2	57	28		3	18	74	25	14		4	6	
	3												
	4												
	5												
	6	5	2		96	75	9	5			38	5	
	7	38	70	100	1	7	17	70	86	100	58	89	100

klassifiziert werden (Tabelle 5.2 und Tabelle 5.3), wurde der Grund für diese Ergebnisse zunächst bei der Messreihenaufbereitung (Abschnitt 4.2.2) vermutet. Vor der Datenaufbereitung sind die Klassen Anfahren (3.4c), Normalbremsung (3.5a) und Vollbremsung (3.5b) augenscheinlich noch deutlich voneinander unterscheidbar. Allerdings zeigte sich bei einer weiteren Analyse, bei der die aufbereiteten Messreihen betrachtet wurden, dass die betroffenen Messreihen eins und vier der Vollbremsung, welche am häufigsten falsch klassifiziert wurden, durch die Datenaufbereitung eine andere Charakteristik aufwiesen. Dies wird auch bei der genaueren Betrachtung der Messreihen in Abbildung D.13 und Abbildung D.14 deutlich, vergleicht man die betreffenden Messreihen mit den übrigen Messreihen der Vollbremsung. Die jeweils originale und aufbereitete Messreihe eins und vier ist in Abbildung 5.1 abgebildet.

Vergleicht man bei der aufbereiteten Messreihe eins (5.1c) die Geschwindigkeit und die Beschleunigung auf der Z-Achse mit denen der aufbereiteten Messreihen der anderen Normalbremsungen (Abbildung D.3 und Abbildung D.4), weisen diese einen ähnlichen Verlauf auf. Im Gegensatz dazu zeigt derselbe Vergleich von 5.1c mit den aufbereiteten Messreihen anderer Vollbremsungen (D.13b, D.13c und Abbildung D.14), dass Geschwindigkeit und Z-Achse einen deutlich anderen Verlauf aufweisen.



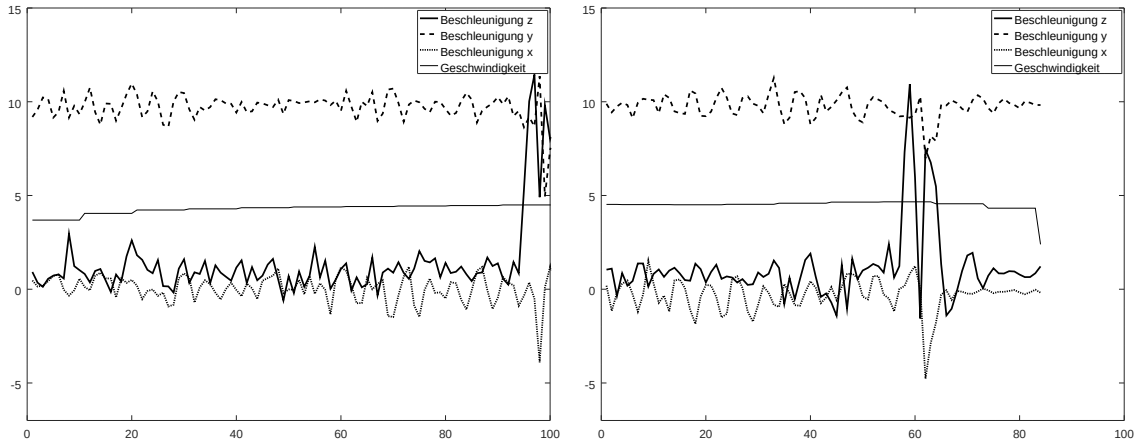
(a) Ursprüngl. Messreihe 1 Vollbremsung (b) Ursprüngl. Messreihe 4 Vollbremsung

(c) Aufber. Messreihe 1 Vollbremsung (d) Aufber. Messreihe 4 Vollbremsung

Abbildung 5.1: Gegenüberstellung: originale und normalisierte Messreihe 1 und 4 der Vollbremsung (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

Die gleiche Problematik trifft auch auf die Messreihe vier der Vollbremsung zu. Ein Vergleich von Geschwindigkeit und Beschleunigung auf der Z-Achse von Messreihe vier (5.1d) mit den aufbereiteten Messreihen zum Anfahren (Abbildung D.1 und Abbildung D.2) zeigt ähnliche Verläufe. Auch hier wieder im Gegensatz dazu derselbe Vergleich von Geschwindigkeit und Z-Achse von Messreihe vier (5.1d), mit den aufbereiteten Messreihen anderer Vollbremsungen (D.13b, D.13c und Abbildung D.14), wo kein ähnlicher Verlauf zu beobachten ist.

Ein weiterer Vergleich der ursprünglichen Messreihen eins (5.1a) und vier (5.1b) mit weiteren ursprünglichen Messreihen der Klasse Vollbremsung in Abbildung 5.2 zeigt, dass die Informationen zur Normalbremsung in Messreihe eins und zum Anfahren in Messreihe vier bereits enthalten waren. Erkennbar ist dies am Verlauf der Geschwindigkeit. In den betreffenden Messreihen ändert sich der Betrag der Geschwindigkeit nachhaltig. In Messreihe eins ist er nach der ersten Hälfte dauerhaft kleiner. In Messreihe vier dagegen dauerhaft größer. Dagegen ändern sind Betrag und Richtung der Beschleunigung auf der Z-Achse (Kraftwirkung längs zur Fahrtrichtung) sprunghaft,



(a) Ursprüngl. Messreihe 5 Vollbremsung (b) Ursprüngl. Messreihe 6 Vollbremsung

Abbildung 5.2: Originale Messreihe 5 und 6 der Vollbremsung (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

nur in einem engen Zeitfenster ohne signifikante nachhaltige Auswirkung. Diese Information scheint bei der Datenaufbereitung jedoch verloren zu gehen.

Die vorangegangen Experimente und Analysen zeigen, dass eine Klassifizierung möglich ist und sogar schon mit sehr wenigen Neuronen sehr gute Resultate liefern kann. Allerdings wurde auch ersichtlich, dass die Klassifizierung stark von der Qualität der vorgelagerten Datenaufbereitung abhängig zu sein scheint. Die Datenaufbereitung, mindestens jedoch die konkrete Implementierung in diesem Experiment, scheint ungenügend. Es wurde daher eine weitere Analyse zu den Auswirkungen der Datenaufbereitung durchgeführt, deren Ergebnisse später in Abschnitt 6.1 vorgestellt werden.

5.1.2 Variation der Anzahl von Trainingsdaten

In diesem Abschnitt werden die Ergebnisse vorgestellt, die bei der Variation der Größe der Trainingsmenge ermittelt wurden. Der experimentelle Aufbau und die Durchführung bleiben im Vergleich zu Abschnitt 5.1.1 weitestgehend identisch. Abweichend davon wurde jedoch stets eine Reservoirgröße von $N = 10$ verwendet und die Anzahl n der für das Training verwendeten Messreihen wurde pro Testdurchlauf um eine Messreihe reduziert. Insgesamt wurden sieben Testdurchläufe durchgeführt, wobei in jedem Testdurchlauf die Klassifizierung mit jeweils 100 unabhängigen Durchläufen (in jedem Durchlauf wurde ein neues Netzwerk erzeugt) vorgenommen wurde. Die Resultate sind in Tabelle 5.4 hinterlegt.

In Tabelle 5.4 ist in der ersten Zeile die Anzahl der Messreihen angegeben, die im jeweiligen Testdurchlauf für das Training verwendet wurden. Alle Ergebnisse in der entsprechenden Spalte beruhen auf der Klassifizierung mit dieser Anzahl n verwendeten Trainingsmenge (Anzahl Messreihen). Es folgen jeweils fünf Zeilen pro Klasse mit den Ergebnissen der Klassifikationen. Für $h+$ und $h-$ ist wieder die Güte in % angegeben. Die jeweils ersten zwei Zeilen geben die Güte für die Klassifikation der Trainingsmenge an (je eine Zeile für $h+$ und $h-$). An diesen ist ablesbar wie gut die Trainingsmenge erfolgreich klassifiziert werden konnte. Die daran anschließenden drei Zeilen beziehen sich jeweils auf die Klassifizierung der Testdaten. Auch hier

ist wiederum die Güte für $h+$ und $h-$ in % angegeben. Zusätzlich ist hier die Anzahl der verwendeten Testdaten (Messreihen) angegeben. Diese Anzahl ergibt sich aus der Gesamtmenge der Messreihen einer Klasse, abzüglich der für das Training verwendeten Messreihen und ist nicht für jede Klasse gleich. Messreihen aus der Testdatenmenge wurden nicht für das Training verwendet.

Insgesamt zeigen die Auswertungen, dass, mit Ausnahme der Vollbremsung, auch mit sehr geringen Trainingsmengen (2 bis 4 Messreihen) sehr gute Ergebnisse erzielt werden können. So ist z.B. bei der Klasse Anfahren bereits mit einer Trainingsmenge von $n = 3$ bei den Testdaten eine Güte von 97,3% ($h+$) erreicht worden. Bei $n = 2$ lag die Güte immer noch bei 83,7% ($h+$). Obwohl auch noch bessere Ergebnisse mit lediglich $n = 2$ erzielt werden konnten, so z.B. bei der Klasse Rechtskurve mit 98,5% ($h+$) und der Klasse Stillstand mit konstant 100% ($h+$), sind auch deutlich schlechtere Ergebnisse ermittelt worden. So wird z.B. für die Klasse Geradeaus bei $n = 2$ nur eine Güte von 52,1% erreicht.

Mit Ausnahme der Klasse Geradeaus zeigen die Daten in Tabelle 5.4, dass eine Vergrößerung der Trainingsmenge von $n = 2$ auf $n = 7$ und $n = 8$ zu einer Verbesserung der Güte bei der Klassifizierung der Testdaten führt. Dies ließe den Schluss zu, dass die Güte mit größer werdender Trainingsmenge kontinuierlich verbessert werden kann. Jedoch ist bei dieser Vermutung die geringe Datenmenge und der Umstand, dass bei größer werdender Trainingsmenge n Messreihen aus der Testmenge automatisch in die Trainingsmenge fallen (siehe Abschnitt 4.2.1), kritisch zu betrachten.

Weiterhin ist aber auch deutlich zu erkennen, dass die Güte bei zunehmender Trainingsmenge auch schlechter werden kann. Deutlich zu erkennen ist dies bei der Normalbremsung und der vergrößerten Trainingsmenge von $n = 3$ auf $n = 6$. Ebenso bei der Vollbremsung und der Vergrößerung von $n = 3$ auf $n = 4$. Eine mögliche Ursache könnte darin liegen, dass mit zunehmender Trainingsmenge u.U. auch weitere Merkmale gelernt werden, die nicht unbedingt repräsentativ für die Klasse sind. Weisen die überwiegenden Testdaten diese Merkmale nicht oder weniger ausgeprägt auf, werden diese u.U. nicht mehr ihrer Klasse zugeordnet. Im Falle der Vollbremsung werden, wie in Abschnitt 5.1.1 gezeigt, zwei Messreihen gelernt, die ebenso Merkmale der Klassen Normalbremsung und Anfahren aufweisen. Hierdurch könnte eine Verschlechterung der Güte begründet sein. Es ist daher von nicht unerheblicher Bedeutung, von welcher Qualität die Trainingsmenge ist und welchen Einfluss vorgelagerte Prozesse, wie z.B. die Datenaufbereitung auf die Klassifizierung haben.

Tabelle 5.4: Klassifikation bei variierender Trainingsmenge mit $N = 10$

			Trainingsdatensätze n						
Manöver			**2**	**3**	**4**	**5**	**6**	**7**	**8**
Anfahren	Training	h+	100	100	100	100	100	99,7	99,9
		h-	93,0	74,0	63,5	47,6	48,5	53,4	52,1
	Test	n	9	8	7	6	5	4	3
		h+	83,7	97,3	98,9	100	100	100	100
		h-	37,7	54,6	50,6	49,3	69,8	72,5	88,0
Normalbremsung	Training	h+	100	100	100	100	100	98,4	99,1
		h-	49,5	56,7	63,0	58,8	45,3	44,9	46,8
	Test	n	6	5	4	3	2	1	0
		h+	88,5	91,0	88,0	88,7	85,5	100	-
		h-	37,7	29,4	11,0	6,7	7,0	27,0	-
Rechtskurve	Training	h+	100	98,7	97,0	97,8	99,5	96,4	98,5
		h-	98,0	99,0	96,0	100	99,3	98,0	96,8
	Test	n	15	14	13	12	11	10	9
		h+	98,5	97,4	95,6	97,7	99,1	97,5	99,2
		h+-	94,1	96,6	95,2	97,5	97,1	97,0	96,0
Linkskurve	Training	h+	100	99,7	100	100	99,8	95,6	96,4
		h-	94,0	95,7	95,0	96,0	95,0	93,0	93,9
	Test	n	13	12	11	10	9	8	7
		h+	70,7	85,8	87,6	83,0	91,6	98,4	99,4
		h-	93,0	94,6	95,0	95,2	95,0	93,0	94,0
Stillstand	Training	h+	100	100	100	100	100	100	100
		h-	94,0	96,0	95,0	95,0	95,0	94,0	95,0
	Test	n	8	7	6	5	4	3	2
		h+	100	100	100	100	100	100	100
		h-	94,0	96,0	95,0	95,0	95,0	94,0	95,0
Geradeaus	Training	h+	93,5	95,7	99,5	95,0	97,7	96,6	95,6
		h-	0	0	0	0	0	0	0
	Test	n	7	6	5	4	3	2	1
		h+	45,4	74,3	89,6	82,3	80,0	74,5	48,0
		h-	0	0	0	0	0	0	0
Vollbremsung	Training	h+	87,0	63,0	30,8	34,2	32,8	40,0	46,3
		h-	22,0	2,3	0	0	0	0	0
	Test	n	6	5	4	3	2	1	0
		h+	11,0	35,6	20,3	36,7	78,5	69,0	-
		h+-	0	0	0	0	0	0	-

5.2 Detektion unbekannter Manöver

In den folgenden Experimenten wurde untersucht, wie der Klassifikator mit Messreihen unbekannter Klassen umgeht. Zu diesem Zweck wurde zunächst die Klasse Linkskurve mit all ihren Messreihen aus der Trainingsmenge entfernt. Entsprechend wurde auch kein Konzeptor für diese Klasse berechnet, mit denen die Messreihen der Linkskurve hätten klassifiziert werden können. Dazu ergänzend wurden drei weitere Experimente durchgeführt, bei denen weitere Klassen aus der Trainingsmenge vollständig entfernt wurden. So wurden im zweiten Experiment die Linkskurve und Rechtekurve entfernt, in einem dritten Experiment die Linkskurve, Rechtskurve und Geradeaus und in einem vierten Experiment die Linkskurve, Rechtskurve, Geradeaus, Normalbremsung und Vollbremsung. Im Rahmen eines fünften Experiments wurde jede der sieben bekannten Klassen j einmal als unbekannte Klasse behandelt, um deren Zuordnung zu ermitteln.

Die Wahl, zunächst nur die Linkskurve wegzulassen, wurde zufällig getroffen. Die Entscheidungen, anschließend die Rechtskurve und später weitere Klassen wegzulassen, beruhen jedoch auf den Ergebnissen des jeweils vorangegangenen Experiments, wie es folgend vorgestellt wird.

Zur Durchführung dieser Experimente waren entsprechende Modifikationen in den Octave-Skripten notwendig. Im Skript Quelltext C.10 wurden die Berechnungen für den positiven und negativen Konzeptor anhand ihres Klassenindexes (vgl. Quelltext C.2) identifiziert und übersprungen. Am Ende desselben Skripts wurden die Berechnungen für die Konzeptoren „CnotAny“ und „CorAll“ entsprechend angepasst (siehe Abschnitt 4.2.5). Weiterhin wurde das Skript testClassification.m (Quelltext C.14) so angepasst, dass bei Klassifikationen nicht mehr der Konzeptor für die Klasse Linkskurve, der auch nicht zur Verfügung gestanden hätte, verwendet wurde. Diese Modifikationen wurden für die weiteren Experimente, bei denen andere Klassen nicht gelernt werden sollten, analog durchgeführt. Sollen mehr als eine Klasse unbekannt sein und sind nicht zu lernen, wurden die entsprechenden Klassen gleichzeitig entfernt und deren Konzeptoren nicht berechnet. Somit wurde sichergestellt, dass für unbekannte Klassen auch keine Konzeptoren verfügbar waren. Eine weitere Modifikation in Quelltext C.14 war die Ergänzung der Konzeptoren „CnotAny“ und „CorAll“, sodass jede Messreihe auch mit diesen Konzeptoren überprüft wurde. Das Ergebnis der Überprüfung mit „CnotAny“ wurde dem Vektor $h+$, und das mit „CorAll“ dem Vektor $h-$, jeweils mit dem Pseudoklassen-Index 8, zugeordnet.

Der Konzeptor „CorAll“ repräsentiert den kombinierten linearen Unterraum aller bekannten Konzeptoren. Eine Messreihe einer bekannten Klasse kann durch Merkmale dieses Raums abgebildet werden werden. Der Konzeptor „CnotAny“ ist die Negation von „CorAll“ und repräsentiert den linearen Unterraum, der nicht durch bekannte Konzeptoren abgebildet werden kann. Die Vermutung vor diesem Experiment war, dass alle Messreihen der jetzt unbekannten Klassen überwiegend der Pseudoklasse „CnotAny“ zugeordnet werden. Also z.B. bei der Überprüfung einer Messreihe der Klasse Linkskurve, die nun unbekannt ist, im Vektor $h+$ der größte Wert überwiegend in Zeile acht zu finden ist.

Die Durchführung der Experimente zu den unbekannten Manövern unterscheidet sich nicht zu denen der vorangegangen Experimenten. Lediglich die zuvor Beschrie-

benen Modifikationen wurden durchgeführt und es wird mit den zwei Konzeptoren „CnotAny“ und „CorAll“ wie beschrieben gearbeitet. Die Datenaufbereitung (Tabelle 4.1 Nr. 1) wurde lediglich einmal durchgeführt um Rechenzeit einzusparen. Zudem hätte eine erneute Durchführung lediglich die selben aufbereiteten Datensätze zur Folge.

Bei den Experimenten zur Untersuchung der unbekannten Manöver, bei dem zunächst lediglich die Linkskurve nicht trainiert wurde, wurde stets ein Reservoir der Größe $N = \{10\}$ verwendet. Je Experiment wurden 100 Testdurchläufe realisiert, wobei pro Durchlauf ein neues zufälliges Netzwerk (inkl. zufällig generiertem Reservoir) verwendet wurde.

Die Ergebnisse der ersten vier Experimente, bei denen schrittweise Klassen aus der bekannten Trainingsmenge entfernt wurden, sind in Tabelle 5.5 aufgeführt. Jedes dieser Experimente ist über die erste Spalte in Tabelle 5.5 auffindbar. In dieser ersten Spalte sind die im jeweiligen Experiment nicht trainierten (unbekannten) Klassen aufgelistet. Es folgt in der zweiten Spalte der Messreihenindex k, sodass zu jeder der acht klassifizierten Messreihen die Ergebnisse einzeln ausgewertet werden können. Es wurden jeweils nur die Messreihen der unbekannten Klasse Linkskurve betrachtet und in Tabelle 5.5 aufgenommen. Die folgenden Spalten enthalten die Ergebnisse zu den Klassifikationen. Zunächst die Ergebnisse für den Ergebnisvektor $h+$, anschließend für $h-$. Jede dieser zwei Ergebnisspalten fasst wiederum acht Spalten zusammen. Diese acht Spalten sind mit dem Klassenindex 1 bis 8 beschriftet. In jeder dieser acht Spalten steht die Anzahl der Zuordnungen von Linkskurven. Zur besseren Lesbarkeit wurden in der Tabelle nur Werte größer 0 eingetragen. Eine Leere Zelle ist demnach gleichbedeutend mit dem Wert 0. Die Klasse, die in diesem Experiment unbekannt war und über keinen Konzeptor verfügte, ist mit einem Minus „-“ gekennzeichnet. Zuordnungen waren hier nicht möglich.

Zu beachten ist, dass in Tabelle 5.5 auch für die Experimente, bei denen mehrere Klassen unbekannt sind (nicht trainiert wurden) nur die Zuordnung der Linkskurven enthalten sind. Die zweite Spalte k adressiert so immer die ersten acht Messreihen der Klasse Linkskurve. Möchte man das Ergebnis der Zuordnung einer Linkskurve in einem bestimmten Experiment wissen, ist die Tabelle wie folgt zu lesen: Es ist zunächst das jeweilige Experiment zu identifizieren. Z.B. das dritte mit unbekannter Linkskurve, Rechtskurve und Geradeaus (in Tabelle 5.5 über die erste Spalte zu identifizieren). Es ist nun der Index der Messreihe festzulegen. Z.B. die achte Messreihe der Klasse Linkskurve (in Tabelle 5.5 über den Index in Spalte k zu identifizieren). Nun können die Ergebnisse in den folgenden Spalten dieser Zeile abgelesen werden. So wurde im entsprechenden Experiment die achte Messreihe aus der Klasse Linkskurve der Klasse Normalbremsung (Klassenindex 2) 74 mal und der Klasse Vollbremsung (Klassenindex 7) 26 mal zugeordnet.

Im ersten Experiment, in dem nur die Linkskurve unbekannt ist, wurden die Messreihen der unbekannten Klasse Linkskurve als Rechtskurve oder Geradeaus erkannt. Entgegen den Erwartungen wurde keine Messreihe dem Konzeptor (Pseudo-Klasse) „CnotAny“ zugeordnet. Dem Konzeptor (Pseudo-Klasse) „CorAll“ wurden dagegen ausnahmslos alle Messreihen zugeordnet.

Tabelle 5.5: Klassenzuordnung unbekannter Linkskurven (1 = Anfahren, 2 = Normalbremsung, 3 = Rechtskurve, 4 = Linkskurve, 5 = Stillstand, 6 = Geradeaus, 7 = Vollbremsung, 8 = CnotAny bei h+ und CorAll bei h-)

		Klassenzuordnung bei N=10 und n=8															
		h+								**h-**							
j unbek.	k	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Linksk.	1			53	-		47						-				100
	2			44	-		56						-				100
	3			24	-		76						-				100
	4			80	-		20						-				100
	5			53	-		47						-				100
	6				-		100						-				100
	7				-		100						-				100
	8				-		100						-				100
Linksk. Rechtsk.	1			-	-		100					-	-				100
	2			-	-		100					-	-				100
	3			-	-		100					-	-				100
	4			-	-		100					-	-				100
	5			-	-		100					-	-				100
	6			-	-		100					-	-				100
	7			-	-		99	1				-	-				100
	8			-	-		99	1				-	-				100
Linksk. Rechtsk. Gerad.	1		98	-	-		-	2				-	-		-		100
	2		95	-	-		-	5				-	-		-		100
	3		84	-	-		-	16				-	-		-		100
	4		96	-	-		-	4				-	-		-		100
	5		100	-	-		-					-	-		-		100
	6		66	-	-		-	34				-	-		-		100
	7		61	-	-		-	39				-	-		-		100
	8		74	-	-		-	26				-	-		-		100
Linksk. Rechtsk. Gerad. Norm.br. Vollbr.	1	29	-	-	-		-	-	71	57	-	-	-		-	-	43
	2	34	-	-	-		-	-	66	52	-	-	-		-	-	48
	3	62	-	-	-		-	-	38	26	-	-	-		-	-	75
	4	47	-	-	-		-	-	53	35	-	-	-		-	-	65
	5	41	-	-	-		-	-	59	43	-	-	-		-	-	57
	6	75	-	-	-		-	-	25	10	-	-	-		-	-	90
	7	81	-	-	-		-	-	19	6	-	-	-		-	-	94
	8	78	-	-	-		-	-	22	8	-	-	-		-	-	92

Von Interesse war nun, welche Klassifikationen vorgenommen werden, wenn auch diese Klassen (Rechtskurve und Geradeaus) unbekannt sind. Zu diesem Zweck wurde im zweiten Experiment gezielt die Rechtskurve ($j = 3$) aus der Trainingsmenge entfernt, die damit auch unbekannt war. Das Ergebnis war, dass die unbekannten Messreihen der Linkskurve nun fast ausschließlich der Klasse Geradeaus zugeordnet wurden ($j = 6$). In Tabelle 5.5 sind diese Daten in den Zeilen hinterlegt, die dem Experiment mit den unbekannten Klassen Links- und Rechtskurve zugeordnet sind. Das Entfernen der Klasse Geradeaus aus der Trainingsmenge, die damit ebenfalls unbekannt war, führt bei der Klassifikation von unbekannten Linkskurven dazu, dass diese den bekannten Klassen Normalbremsung ($j = 2$) und Vollbremsung ($j = 7$) zugeordnet wurden. Erst im vierten Experiment, bei der nur noch die realen Klassen Anfahren ($j = 1$) und Stillstand ($j = 5$) bekannt waren, wurden der Pseudo-Klasse „CnotAny“ Messreihen zugeordnet. Auch im Ergebnisvektor $h-$ wurden erst im vierten Experiment nicht alle Messreihen dem Konzeptor „CorAll“ zugeordnet.

Erst im vierten Experiment mit lediglich zwei bekannten Klassen wurden Messreihe der unbekannten Klasse Linkskurve dem Konzeptor „CnotAny“ zugeordnet. Dies widerspricht den ursprünglichen Erwartungen insofern, dass die unbekannte Klasse zunächst anderen Klassen zugeordnet wurde. Erst als kaum bekannte Klassen zum Vergleich zur Verfügung standen, wurden unbekannte Messreihen dem Konzeptor „CnotAny“ zugeordnet. Die Ergebnisse ließen sich jedoch wie folgt erklären. Jeder Konzeptor stellt einen linearen Unterraum dar, in dem die Reservoiraktivitäten abgebildet werden können. Werden Konzeptoren miteinander kombiniert, wird ein gemeinsamer linearer Unterraum definiert, der nun alle Reservoiraktivitäten abbilden muss, die zuvor durch die einzelnen Konzeptoren abgebildet wurden. Ein konkretes Fallbeispiel könnte wie folgt aussehen. Bezogen auf eine Kombination aus den Fahrmanövern Stillstand und Anfahren bedeutet dies, dass der kombinierte lineare Unterraum aus diesen zwei Konzeptoren nun auch das typische Verhalten abbilden kann, welches in einem Reservoir bei einer Beschleunigung auftritt. Der Konzeptor zum Manöver Stillstand allein enthält diese Informationen (Merkmale) nicht, er kann diesen linearen Unterraum zur Beschleunigung nicht abbilden. Dagegen kann ein, aus mehreren linearen Unterräumen, kombinierter Konzeptor Informationen zu vielen Merkmalen unterschiedlicher Klassen enthalten.

Die Messreihen einzelner Klassen verfügen über viele Merkmale, solche die für eine Klasse spezifisch sind, aber auch solche, die in vielen Klassen auftauchen. Ein Klassen-spezifisches Merkmal der Linkskurve ist z.B. die Beschleunigung quer zur Fahrtrichtung (X-Achse) mit negativen Betragsvorzeichen. Allerdings enthält die Linkskurve auch Informationen zur Erdanziehungskraft, ebenso wie alle anderen Klassen. In Messreihe eins der Linkskurve (3.4f) fällt zudem die Geschwindigkeit kontinuierlich. Ein Merkmal eines Bremsvorgangs. Weisen Messreihen unbekannter Klassen Merkmale auf, repräsentiert durch lineare Unterräume, die in bekannten Klassen ebenfalls auffindbar sind, ist eine Zuordnung von unbekannter zu bekannter Klasse denkbar. Insbesondere dann, wenn diese Übereinstimmung der kombinierten linearen Unterräume größer ist, als deren negierter linearer Unterraum. Es kann damit nicht abschließend geklärt werden, wie mit diesem Klassifikator mit unbekannten Messreihen idealerweise umgegangen werden soll. Weitere Analysen, mit entsprechend geeigneter Datenbasis, sind nötig.

Im Rahmen dieser Arbeit wurde zu der Problematik, Zuordnung unbekannter Messreihen, lediglich noch die Zuordnung der verbleibenden Klassen in einem weiteren Experiment überprüft. Dieses Experiment zu dieser Problematik wurde auf die gleiche Weise durchgeführt, wie die vier bereits vorangegangenen Experimente. Jedoch wurde jeweils immer nur eine Klasse aus der Trainingsmenge entfernt, die damit unbekannt war. Die Ergebnisse dieses Experiments sind in Tabelle 5.6 abgebildet. In der ersten Spalte ist der Name der Klasse zu finden, die aus der Trainingsmenge entfernt wurde. Die zweite Spalte enthält den zur Klasse gehörenden Index j. Es folgen acht Spalten, in denen die Zuordnungen der unbekannten Klasse dargestellt sind. Dargestellt sind wieder nur Werte größer 0. Auch hier zeigt sich wieder, dass die Messreihen einer unbekannten Klasse mindestens einer bekannten Klasse zugeordnet wurden. Keine Messreihe wurde dem Konzeptor „CnotAny“ (Spalte 8 der Klassenzuordnung in Tabelle 5.6) zugeordnet.

Tabelle 5.6: Klassenzuordnung unbekannter Klassen mit $N = 10$

		Klassenzuordnung							
j **unbekannt**	j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Anfahren	1	-				**86,6**		13,4	
Normalbremsung	2		-	0,2			28,9	**70,9**	
Rechtskurve	3			-	13,2		**86,8**		
Linkskurve	4			31,8	-		**68,2**		
Stillstand	5	**100**				-			
Geradeaus	6		1,1	**96,0**	0,3		-	2,6	
Vollbremsung	7	15,2	**58,5**				26,3	-	

5.3 Zusammenfassung

Es wurde in diesem Kapitel zunächst gezeigt, dass mit dem gewählten Ansatz eine Klassifikation bekannter Klassen von Messreihen möglich ist. Es konnte gezeigt werden, dass bereits mit wenigen (4) Reservoirneuronen und einer geringen Trainingsmenge (8 Messreihen) Zuordnungen möglich sind und sehr gute Ergebnisse erzielt werden können. Problematisch scheint dagegen noch die ungenügend geklärte Abhängigkeit der Klassifikationsgüte von der vorgelagerten Datenaufbereitung.

Weiterhin ungelöst, für die Problematik Manövererkennung, ist der Umgang mit unbekannter Manövern, bzw. der sichere Umgang mit zusammengesetzten Manövern. Ein, im Rahmen dieser Arbeit nicht mehr weiter betrachteter Ansatz, wäre eine Dekomposition der Manöver. Denkbar wäre ein Ansatz, bei dem jede Dimensionen einzeln betrachtet wird. Jede Bechleunigungsachse als auch die Geschwindigkeit, könnten einzeln, mit spezialisierten Datenaufbereitungen und neuronalen Netzen, ausgewertet werden. Jedoch sind für konkrete Ergebnisse weiterführende Arbeiten notwendig und konnten im Rahmen dieser Arbeit nicht weiter berücksichtigt werden.

6. Identifikation entscheidender Faktoren für die Klassifikationsaufgabe

In Abschnitt 5.1.1 wurde festgestellt, dass die Qualität der Daten und deren Aufbereitung, wie sie in Abschnitt 4.2.2 erläutert wurde, einen signifikanten Einfluss auf die charakteristischen Eigenschaften einer aufbereiteten Messreihe haben können.

Dieses Kapitel widmet sich daher weiterer Analysen, um zu erörtern, welche Auswirkungen einzelne Faktoren auf die Klassifikationsaufgabe haben. Zunächst wird analysiert, welche Auswirkungen die Auswahl der Stützstellen hat, mit denen die Messreihen aufbereitet (interpoliert) werden. Weiterhin wird analysiert, welche Auswirkungen sich ergeben, wenn eine lineare Aktivierungsfunktion statt des Tangens hyperbolicus angewendet wird und wenn statt der interpolierten Werte die Stützstellen direkt verwendet werden

6.1 Variation der Stützstellen

Im Fall der Vollbremsungen führte die Datenaufbereitung dazu, dass vorwiegend Merkmale in der aufbereiteten Messreihe enthalten waren, die sich über einen längeren Zeitraum (aber noch innerhalb der Messreihe) verändert hatten. Merkmale basierend auf kurzfristigen Werteänderungen schienen dagegen kaum berücksichtigt. Im konkreten Fall führte dies dazu, dass eine Vollbremsung anschließend charakteristische Merkmale einer Normalbremsung aufwies. Eine weitere Vollbremsung wies anschließend charakteristische Merkmale der Klasse Anfahren auf. Folgend wird daher untersucht, welche Auswirkung die Wahl der Stützstellen und die daraus resultierende Abtastung der Messreihen hat.

Die ursprüngliche Abtastung ermittelt die Gesamtlänge einer Messreihe und bestimmt, ausgehend vom ersten Messwert, insgesamt vier äquidistante Messpunkte für die weitere Verwendung. Die Distanz zwischen den Messpunkten wird für jede

Messreihe separat ermittelt und so gewählt, dass diese maximal ist. Dadurch werden vier Messpunkte stets so ausgewählt, dass diese über die gesamte Messreihe gleichmäßig verteilt liegen. Da diese Strategie bei Messreihen der Vollbremsung ungünstig scheint, wurde die Auswahl der Stützstellen variiert, um die Auswirkungen analysieren zu können. Die Festlegung zur Verwendung von nur vier Messpunkten (Stützstellen) beruht auf der Tatsache, dass bei der vorgestellten Klassifikation von Jaeger [16, S. 74] ebenfalls vier Stützstellen verwendet wurden. Diese Festlegung wurde unverändert übernommen. Weiterführende Arbeiten könnten daher untersuchen inwieweit eine größere Anzahl an Stützstellen die Klassifikation beeinflussen. Zu beachten ist dabei, dass sich die verwendete Anzahl an Stützstellen direkt auf die Größe der Zustandsvektoren z auswirkt (siehe Formel 4.3 in Abschnitt 4.1.1). Dies führt wiederum zu einer größeren Korrelationsmatrix R (siehe Abschnitt 2.5.6 und Abschnitt 4.1.1) und einem erhöhten Rechenaufwand.

Bei den alternativen Abtastungen wurden ebenfalls vier äquidistante Messpunkte ermittelt. Jedoch wurden die Abstände zwischen den Stützstellen, wie in Abbildung 6.1 dargestellt, verringert und über die Messreihe zentriert. Damit sollte erreicht werden, dass die Stützstellen und die damit verbundenen abgetasteten Werte, im für eine Vollbremsung charakteristischem Bereich liegen.

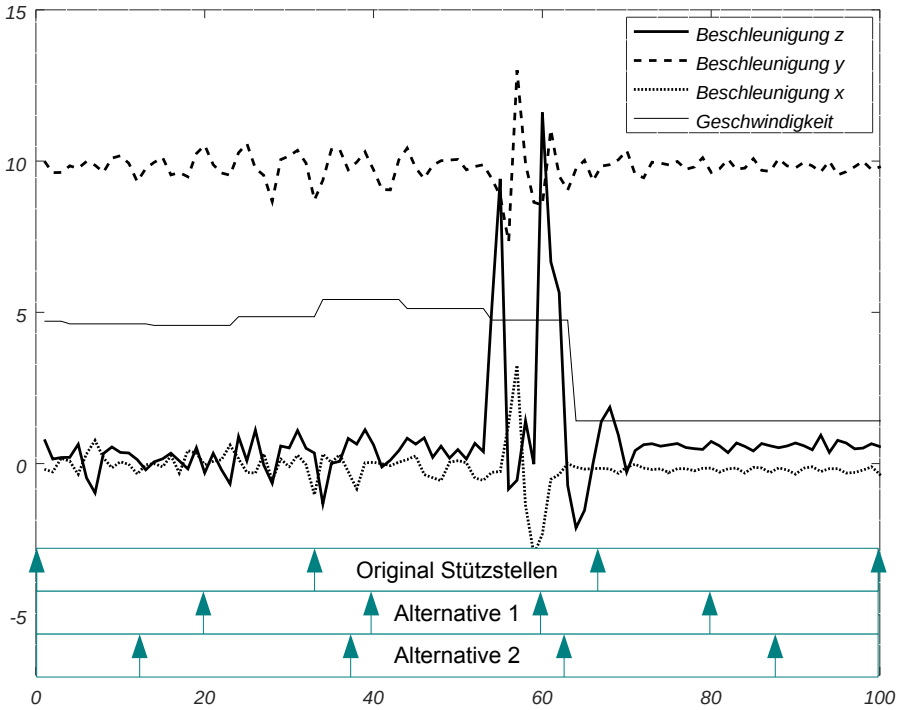


Abbildung 6.1: Alternative Abtastungen der Messreihen

Mit der originalen und zwei alternativen Abtastungen wurden wieder Testläufe zur Klassifizierung, wie in Abschnitt 5.1.1 beschrieben, durchgeführt. Es wurden ebenfalls die ersten acht Messreihen einer jeden Klasse für das Training verwendet und auch nur die Klassifizierung dieser Trainingsmenge betrachtet. In diesem Experiment wurden jedoch nur Reservoire mit einer Größe von $N = 10$ Neuronen betrachtet. Sowohl für die Originalabtastung als auch für die zwei alternativen Abtastungen

wurden jeweils 100 unabhängige (pro Testlauf wurde ein neues Netzwerk inkl. Reservoir erzeugt) Testläufe durchgeführt.

Die Ergebnisse dieser Testläufe sind in Tabelle 6.1 für $h+$ zusammengefasst. Es ist wieder die Güte in % angegeben. Es zeigen sich bei den alternativen Abtastungen deutliche Änderungen, was die Güte der Klassifizierung betrifft. Mit Ausnahme der Vollbremsung und für Alternative 1 auch die Linkskurve werden alle Klassen bei den alternativen Abtastungen schlechter klassifiziert. Im ungünstigsten Fall wird die Güte um bis zu 42,5% (Normalbremsung, Alternative 2) verringert. Die Klasse Stillstand wird bei Alternative 1 weiterhin gut zugeordnet (verringerte Güte um 0,9%). Im Gegensatz dazu ist bei der Klassifizierung der Vollbremsungen eine Verbesserung von 22,2% bei Alternative 1 und von 4,7% bei Alternative 2 zu verzeichnen.

Diese Auswertungen stützen bis auf Weiteres die Annahme, dass die Art, wie die vorgeschaltete Messreihennormalisierung durchgeführt wird, einen erheblichen Einfluss auf die Klassifikation hat. Nicht zuletzt gehen durch sie Merkmale in den Messreihen verloren, wenn diese aufbereitet werden. Dies wird auch deutlich, wenn man die Abtastfrequenzen miteinander vergleicht. Die physikalischen Größen in der App werden mit einer Abtastrate von 10 Hz erfasst. Die Abtastrate bei der Datenaufbereitung variiert abhängig von der Länge der Messreihe. Im Fall der ersten Messreihe der Vollbremsung mit 129 Messpunkten und einer Gesamtlänge von 12,9 Sekunden, beträgt die Abtastrate dann nur noch 0,31 Hz. Bei Messreihe vier der Vollbremsung mit einer Gesamtlänge von 10,1 Sekunden dagegen 0,4 Hz. Durch diese Art der Datenaufbereitung wird zwar eine Reduktion erreicht, sodass Messreihen unterschiedlicher Länge miteinander verglichen werden können. Diese Reduktion ist ebenfalls notwendig, um die Größe der Korrelationsmatrizen R einschränken zu können. Jedoch kann die Nichteinhaltung des Nyquist-Shannon-Abtasttheorems dazu führen, dass das abgetastete Signal fehlerhaft ist. Im Falle der Vollbremsung gingen Informationen und damit die signifikanten Merkmale dieser Klasse bei zwei Messreihen verloren. Die Merkmale, die bei diesen Messreihen erhalten geblieben waren, führten dann wiederum zu falschen Klassifikationen.

Tabelle 6.1: Klassifikation bei alternativer Messreihennormalisierung ($N = 10$, $n = 8$)

	Klassifikation h+		
Manöver	**Original**	**Alternative 1**	**Alternative 2**
Anfahren	99,9	61,3	73,0
Normalbremsung	99,1	72,6	56,6
Rechtskurve	98,5	92,4	84,3
Linkskurve	96,4	96,8	87,4
Stillstand	100	99,1	99,1
Geradeaus	95,6	80,1	84,1
Vollbremsung	46,3	68,5	51,0

Allerdings ist dies zunächst auf dieses Experiment und dem eng gefassten Szenario sowie Messreihen beschränkt. Weitere Analysen müssten zeigen, inwieweit sich eine

entsprechende Datenaufbereitung eignet und optimieren ließe. Ein möglicher Ansatzpunkt könnte eine vorherige Analyse des Frequenzspektrums der Messreihen sein, oder ein optimiertes Verfahren zur Auswahl eines Ausschnitts aus einer Messreihe, welches auf Basis bestimmter Merkmale einen Start- und Endpunkt der zu klassifizierenden Messreihe festlegt. Eine weitere Möglichkeit wäre die Veränderung der Anzahl von Stützstellen oder des Polynoms, mit denen die Messreihen aufbereitet werden. Eine Analyse einer hinreichend großen Datenmenge könnte Auskunft über bessere Strategien zur Abtastung der Messreihe und Interpolation der aufbereiteten Messreihe liefern. Eine Optimierung in diesem Bereich scheint umso wichtiger, da die einzelnen Klassen sich nur in wenigen Merkmalen unterscheiden. Die Anwendung eines solchen Klassifikators kann dann allerdings durchaus kritisch betrachtet werden. So stellt sich z.B. die Frage, inwieweit eine vorab Analyse der Messreihen, das Erkennen bestimmter Merkmale und daraufhin optimierte Aufbereiten der Daten, einen nachgeschalteten Klassifikator noch notwendig machen.

6.2 Verwendung einer linearen Aktivierungsfunktion

Die Aktivierungsfunktion, Tangens hyperbolicus, begrenzt die Werte für die Aktivierung der Neuronen auf das Intervall $[-1, 1]$. Motiviert durch die vorhersagenden neuronalen Netze (Abschnitt 2.5.5), bei denen stets eine lineare Aktivierung der Neuronen erfolgt, wurden weitere Experimente mit einer linearen Aktivierung durchgeführt. Der Verlauf der Experimente entspricht den in Abschnitt 5.1 beschriebenen. Lediglich im Octave-Skript Quelltext C.8 wurde die Aktivierungsfunktion tanh entfernt. Anschließend wurden 100 Testläufe (jeder mit einem zufällig generiertem Reservoir) mit einer Trainingsmenge von $n = 5$ und $n = 8$ durchgeführt. Die Ergebnisse sind in der Spalte „linear“in Tabelle 6.2 zusammengefasst. Angegeben ist jeweils die Differenz der Güte der ursprünglichen Implementierung, mit der Aktivierungsfunktion Tangens hyperbolicus und der modifizierten Implementierung, mit linearer Aktivierungsfunktion in %. Die Berechnungsvorschrift lautet: $\Delta_{\text{Güte}} = \text{Güte}_{\text{original}} - \text{Güte}_{\text{modifikation}}$. Ist $\Delta_{\text{Güte}}$ negativ, ist die Klassifizierung der ursprüngliche Implementierung schlechter. Diese Werte sind in Tabelle 6.2 hervorgehoben.

Die Ergebnisse aus diesem Experiment lassen vorerst keinen endgültigen Schluss zu. Betrachtet man die Klassifizierung der Trainingsmessreihen ist die Güter einer lineare Aktivierung maximal 2,9% (Geradeaus, $h+$, $n = 8$) schlechter. Bei der Vollbremsung ($h+$, $n = 5$) sogar um 1,4% besser. Für $h-$ liegen ebenso gegensätzliche Werte vor. Beim Anfahren erreicht die lineare Aktivierung eine bessere Güte (um 4,0% bzw. 6,8% besser), bei der Normalbremsung dagegen eine deutlich schlechtere (um 23,2% bzw. 29,0% schlechter). Bei der Klassifikation der Trainingsmenge wird mit einer linearen Aktivierung für $h+$ maximal die gleiche Güte erreicht wie mit der originalen Implementierung mit dem Tangens hyperbolicus.

6.3 Entfernung der Polynominterpolation

In diesem Experiment wurde untersucht, welche Auswirkungen das Entfernen der Polynominterpolation hat. Zu diesem Zweck wurde das Octave-Skript Quelltext C.5

Tabelle 6.2: Gegenüberstellung der Modifikationen bei $N = 10$

Modifikation			**linear**		**polynom**		**linear und polynom**	
n			**5**	**8**	**5**	**8**	**5**	**8**
Anfahren	Training	h+	1,0	0,5	0	0	0,8	0,9
		h-	**-4,0**	**-6,8**	0	**-0,4**	**-4,6**	**-5,8**
	Test	n	6	3	6	3	6	3
		h+	0	0	0	0	0	0
		h-	**-6,0**	**-1,3**	**-0,3**	**-1,0**	**-7,7**	1,7
Normalbremsung	Training	h+	0	2,4	0	**-0,6**	0	1,5
		h-	23,2	29,0	3,4	4,9	21,0	23,4
	Test	n	3	0	3	0	3	0
		h+	14,0	-	**-1,7**	-	5,7	-
		h-	1,3	-	2,3	-	1,7	-
Rechtskurve	Training	h+	2,2	0,3	**-0,4**	**-0,5**	1,8	1,6
		h-	1,2	**-1,5**	1,6	**-0,6**	3,0	0,1
	Test	n	12	9	12	9	12	9
		h+	1,8	**-0,3**	0	0,4	1,3	0,1
		h-	1,5	**-3,0**	1,7	**-1,0**	1,4	**-1,0**
Linkskurve	Training	h+	0	**-0,9**	0	**-0,6**	0	0,4
		h-	2,8	**-0,6**	0,2	**-0,6**	3,0	**-0,4**
	Test	n	10	7	10	7	10	7
		h+	1,4	0	0	0	0,9	**-0,1**
		h-	1,9	**-1,0**	1,1	**-1,0**	2,2	**-1,0**
Stillstand	Training	h+	0	0	0	0	0	0
		h-	6,0	**-1,0**	0	**-1,0**	4,0	0
	Test	n	5	2	5	2	5	3
		h+	0	0	0	0	0	0
		h-	6,0	**-1,0**	0	**-1,0**	4,0	0
Geradeaus	Training	h+	0,6	2,9	**-2,6**	**-1,9**	**-2,0**	3,8
		h-	0	0	0	0	0	0
	Test	n	4	1	4	1	4	1
		h+	2,0	7,0	**-4,3**	**-5,0**	**-2,3**	8,0
		h-	0	0	0	0	0	0
Vollbremsung	Training	h+	**-1,4**	1,5	0,8	**-2,1**	**-1,6**	0,6
		h-	**-0,2**	0	0	0	0	0
	Test	n	3	0	3	0	3	0
		h+	0,7	-	3,3	-	4,7	-
		h-	0	-	0	-	0	-

so modifiziert, dass aus der Messreihe lediglich vier Stützstellen verwendet wurden. Eine anschließende Polynominterpolation fand nicht mehr statt. Es wurde so mit den normalisierten Werten der Messreihe gearbeitet. Die Aktivierung erfolgte wieder mittels Tangens hyperbolicus. Es wurden wieder 100 Testläufe (jeder mit einem zufällig generiertem Reservoir) mit einer Trainingsmenge von $n = 5$ und $n = 8$ durchgeführt. Die Ergebnisse sind in der Spalte „polynom“in Tabelle 6.2 zusammengefasst. Angegeben ist jeweils wieder die Differenz der Güte der ursprünglichen Implementierung, mit Polynominterpolation und der modifizierten Implementierung, Polynominterpolation in %. Die Berechnungsvorschrift lautet: $\Delta_{\text{Güte}} = \text{Güte}_{\text{original}} - \text{Güte}_{\text{modifikation}}$. Ist $\Delta_{\text{Güte}}$ negativ, ist die Klassifizierung der ursprüngliche Implementierung schlechter. Diese Werte sind in Tabelle 6.2 hervorgehoben.

Die Polynominterpolation scheint eine geringere Auswirkung auf die Klassifikation zu haben. Zwar gibt es wieder Klassifikationen, die ohne Polynominterpolation eine schlechtere Güte erreichen, allerdings auch Klassifikationen besserer Güte. Insgesamt ist die Güte der Modifikation bis zu 5% besser. So z.B. bei der Klasse Geradeaus bei den Testmessreihen ($h+$, $n8$). Die größte Verschlechterung gegenüber der ursprünglichen Implementierung liegt bei 4,9% bei der Klasse Normalbremsung (Trainingsmessreihen, $h-$, $n = 8$). Häufig wird sogar die gleiche Güte erreicht ($\Delta_{\text{Güte}} = 0$).

6.4 Entfernung der Polynominterpolation und der linearen Aktivierung

In diesem Experiment wurden beide zuvor vorgestellten Modifikationen (Abschnitt 6.3, Abschnitt 6.2) kombiniert. Sowohl die Polynominterpolation wurde entfernt, als auch die lineare Aktivierung angewandt. In diesem Experiment wurden wieder 100 Testläufe (jeder mit einem zufällig generiertem Reservoir) und jeweils mit einer Trainingsmenge von $n = 5$ und $n = 8$ durchgeführt. Die Ergebnisse sind in der Spalte „linear und polynom“in Tabelle 6.2 zusammengefasst. Angegeben ist jeweils die Differenz der Güte der ursprünglichen Implementierung und der modifizierten Implementierung in %. Die Berechnungsvorschrift lautet: $\Delta_{\text{Güte}} = \text{Güte}_{\text{original}} - \text{Güte}_{\text{modifikation}}$. Ist $\Delta_{\text{Güte}}$ negativ, ist die Klassifizierung der ursprüngliche Implementierung schlechter. Diese Werte sind in Tabelle 6.2 wieder hervorgehoben.

Wie bei den voran gegangenen Experimenten in Abschnitt 6.3 und Abschnitt 6.2 sind die Auswirkungen unterschiedlich. Zwar ist die Güte der ursprünglichen Implementierung häufig besser, dennoch gibt es auch Fälle, in denen keine oder nur eine sehr geringe Auswirkung erkennbar ist. So ist z.B. bei der Klasse Geradeaus die ursprüngliche Implementierung bei der Klassifizierung der Trainingsdaten nur geringfügig besser. Bei der Klassifizierung der Testdaten gibt es keinen Unterschied. Lediglich bei der Überprüfung, ob eine Messreihe auch tendenziell zu einer anderen Klasse gehören könnte ($h-$), ist die Modifikation besser. Im Gegensatz dazu ist bei der Normalbremsung die Güte der ursprüngliche Implementierung mit wenigen Ausnahmen besser.

6.5 Zusammenfassung

In diesem Kapitel wurden die Auswirkungen verschiedener Modifikationen untersucht. Es wurde gezeigt, dass die Messreihennormalisierung und Interpolation so-

wie die Abtastung einen erheblichen Einfluss auf die Güte der Klassifikation haben können. Eine Vorab-Analyse der vorliegenden Daten (Messreihen) scheint empfehlenswert, um die Qualität und Grenzen der Klassifikationen sowie Szenarien wie die der unbekannten Klasse, abschätzen zu können. Ebenso ist die Entwicklung einer alternativen Strategie für die Datenaufbereitung empfehlenswert.

Ebenfalls untersucht wurden einzelne Komponenten des Klassifikators. Speziell die Aktivierungsfunktion und die Polynominterpolation. Dabei konnte kein einheitliches Ergebnis erzielt werden, sodass eine generell positive oder negative Auswirkung nicht ausweisbar ist. In einigen Klassifikationen erreichen die ursprüngliche und die modifizierte Implementierung ähnliche und gleiche Güte. In weiteren Klassifikationen führen sie jeweils zu sehr unterschiedlichen Ergebnissen.

Insgesamt müssen jedoch alle Ergebnisse kritisch hinterfragt werden und sind zwangsläufig nicht zu verallgemeinern. So besteht auch hier die Problematik der unterschiedlichen Qualitäten der Messreihen. Hinzu kommt die geringe Anzahl an Messreihen, die für das Training und den Test zur Verfügung stehen. Weitere umfangreichere Analysen sind daher unerlässlich.

7. Schlussbetrachtungen

Dieses Kapitel setzt sich mit der Nachhaltigkeit eines möglichen Einsatzes dieser Technologie auseinander und betrachtet mögliche Szenarien. Notwendige weiterführende Arbeiten werden vorgestellt und die vorliegende Arbeit mit ihrem Ergebnis zusammengefasst.

7.1 Betrachtungen zur Nachhaltigkeit

Die Betrachtungen zur Nachhaltigkeit sollen sich an dieser Stelle auf die sozialen Aspekte beschränken und lediglich skizzieren, welche signifikanten Auswirkungen denkbar sind.

Ein positiv zu bewertendes Szenario wäre der Einsatz eines Fahrerassistenzsystem, welches dazu beiträgt das Fahrverhalten zu verbessern und hilft Unfälle zu vermeiden. Die Privatsphäre der Anwender wird respektiert und seine Daten werden nicht für weitere Zwecke missbräuchlich verwendet. Weniger positiv, aber noch ohne weitere weitreichende negative Folgen, wäre ein Szenario, in dem solch ein Fahrerassistenzsystem als optionales System ausprobiert, dem aber keine ernste Beachtung geschenkt wird.

Dem Gegenüber stehen nachhaltig negative Auswirkungen. Ein mögliches Szenario ist die Individualisierung des Risikos und der Verlust der Solidarität bei der Versicherung eines Fahrzeugs. Dies kann für einige Personengruppen durchaus problematisch werden. Vielfahrer, die häufig unterwegs sind und dadurch mit einer höheren Wahrscheinlichkeit in gefährliche Verkehrssituationen geraten können, könnten einer höheren finanziellen Belastung ausgesetzt werden. Ebenso Personen, die auf Grund regionaler Besonderheiten, wie z.B. regelmäßig hohes Verkehrsaufkommen oder wechselhaftes Wetter, regelmäßig zu Fahrmanövern gezwungen werden, durch die sie dann fälschlicherweise als risikofreudig eingestuft werden. Ein weiterer Aspekt ist der Datenschutz. Da während der Fahrt kontinuierlich eine Vielzahl von Daten erhoben werden, muss sichergestellt werden, dass diese sensiblen Daten nicht missbräuchlich verwendet werden um zusätzliche Informationen ableiten zu können.

Die Einführung und Akzeptanz dieser Technologie muss daher kritisch hinterfragt werden und es bleibt abzuwiegen, ob das technisch mögliche auch nachhaltig gesellschaftlich vertretbar ist. Auch eine Prüfung, ob gewünschte Effekte wie die Reduktion der Unfallzahlen und die Minderung der Unfallfolgen, nicht auch über alternative Maßnahmen erreicht werden können. So haben die Geschwindigkeit, Fahrzeugmasse und Konstruktion einen erheblichen Einfluss auf den Unfallverlauf und könnten Gegenstand einer Prüfung von Alternativen sein.

7.2 Zukünftige Arbeiten und Fazit

Einfache Fahrmanöver lassen sich mit Hilfe ausgewählter physikalischer Größen wie Geschwindigkeiten und Beschleunigung gut beschreiben. Erfasst werden können diese Größen mit Sensoren, wie sie in modernen Smartphones häufig vorkommen. So wurde in Kapitel 3 gezeigt, dass mit einer einfachen App die relevanten Längs- und Querkräfte direkt gemessen werden können. Die Geschwindigkeit konnte mit einfachen Mitteln, indirekt über die Positionsbestimmung mittels GPS, bestimmt werden. Alle gemessenen Messreihen enthalten Merkmale, die zu jedem Manöver spezifisch sind. Allerdings haben weitere Analysen auch gezeigt, dass die im Rahmen dieser Arbeit gewonnenen Messreihen häufig, wenn auch nur im geringen Umfang, Merkmale anderer Fahrmanöver enthalten. Wie z.B. eine Längsbeschleunigung beim Kurvenfahren. So konnten trotz gewissenhafter Vorbereitung und Durchführung, nur bedingt Messreihen erzeugt werden, die ausschließlich Merkmale eines bestimmten Manövers enthielten. Auch die geringe Anzahl von lediglich 78 Messreihen ist kritisch zu sehen. Weitere Analysen mit deutlich umfangreicheren Datenbeständen müssten Bestandteil zukünftiger Arbeiten sein.

Auf Grundlage der bestehenden Datenbasis konnte in Kapitel 5 jedoch gezeigt werden, dass Fahrmanöver mit rekurrenten neuronalen Netzen, welche mit Konzeptoren verknüpft werden, klassifiziert werden konnten. Die Frage ob Fahrmanöver, die mit einem Smartphone sensorisch erfasst wurden, mit Hilfe rekurrenter neuronaler Netze grundsätzlich klassifiziert werden können, kann demnach positiv beantwortet werden. Jedoch muss auch festgehalten werden, dass die Erkennung beim Fahrmanöver Vollbremsung mit weniger als 65% (Tabelle 5.1, bei $N = 60$) deutlich hinter der Erkennung anderer Manöver mit bis zu 100% (Tabelle 5.1, Anfahren bei $N = 4$) zurückbleibt. Zukünftige Arbeiten könnten evaluieren, inwiefern das rekurrente neuronale Netz selbst oder die vorgelagerten Prozesse zur Datenaufbereitung diese Ergebnisse maßgeblich beeinflusst haben. So konnte in Kapitel 5 gezeigt werden, dass Messreihen nach der Aufbereitung eine andere Merkmalsrepräsentation aufwiesen. Die Ergebnisse zur Vollbremsung könnten damit zu erklären sein, dass die entsprechende Trainingsmenge nach der Aufbereitung auch Messreihen enthielt, die für eine Vollbremsung nicht mehr repräsentativ waren. In Kapitel 6 wurde bereits gezeigt, dass die Qualität der Klassifizierung von der vorgeschalteten Datenaufbereitung maßgeblich abhängig ist. Somit ist die interne Validität der Ergebnisse gefährdet, durch die ursprüngliche Entscheidung in den Messreihen einer Klasse eine zu große Varianz zuzulassen, um einen Eindruck für die Qualifizierung realer Messungen zu bekommen (zu sehen in Abbildung 5.2). Auch wenn weitere Analysen unerlässlich sind, zeigen die Ergebnisse doch insgesamt das Potential rekurrenter neuronale Netze, welche sich im vorliegenden Rahmen für die Klassifikation eignen.

A. Anhang

A.1 Experiment zur Datenerhebung

Abbildung A.1: Experimentaufbau - Smartphone im Fahrezug

A.2 Experiment zur Klassifikation

Das Projekt in Octave enthält die in Abbildung A.2 gezeigten Unterverzeichnisse. Eine Beschreibung zu jedem Verzeichnis und den Dateien im Wurzelverzeichnis steht in Tabelle A.1.

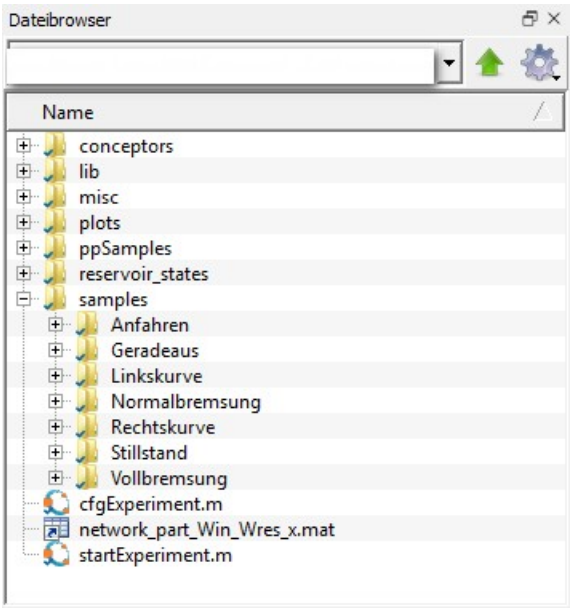


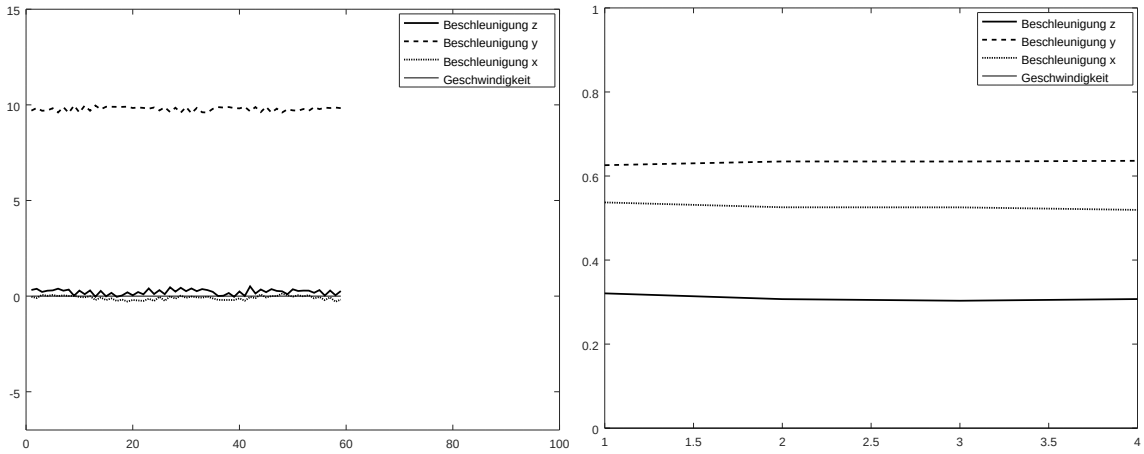
Abbildung A.2: Projektstruktur in Octave - Unterverzeichnisse

Tabelle A.1: Projektstruktur in Octave - Beschreibung Unterverzeichnisse und Dateien

Verzeichnis / Datei	**Beschreibung**
conceptors	Speicherort für Konzeptoren (auch vorläufige) und Korrelationsmatrizen.
lib	Bibliothek mit Skripten für die notwendigen Berechnungen.
misc	Speicherort für individuelle Änderungen, Klassenzuordnungsvektoren und Matrizen für Ausgabegewichte.
plots	Speicherort für Plots (wenn aktiviert).
ppSamples	Speicherort für normalisierte Messreihen. Die Abkürzung pp steht für preprocessed.
reservoir_states	Speicherort für die Reservoirstates, mit denen z berechnet wird.
samples	Speicherort für die Messreihen. Messreihen, die zu einer Klasse gehören, werden gemeinsam in einem weiteren Unterverzeichnis abgelegt.
cfgExperiment.m	Konfigurationsdatei zur Durchführung des Experimentes. Hier wird u.a. festgelegt, ob Plots angefertigt werden sollen. Wie groß das Reservoir sein soll, ob ein neues generiert oder ein vorhandenes verwendet werden soll. Welche Klassen verarbeitet werden sollen und wie viele Messreihen für das Training und für den Klassifikationstest verwendet werden sollen.
network_part_Win_Wres_x.mat	Hier werden notwendige Informationen zum generierten Netzwerk inkl. Reservoir abgelegt. Die ermöglicht die erneute Verwendung, wenn nicht bei jedem neuen Durchlauf ein neues Netz generiert werden soll.
startExperiment.m	Hauptskript, welches die Konfiguration einliest und alle Skripte aus der Bidliothek für die Durchführung des Experimentes einbindet und aufruft.

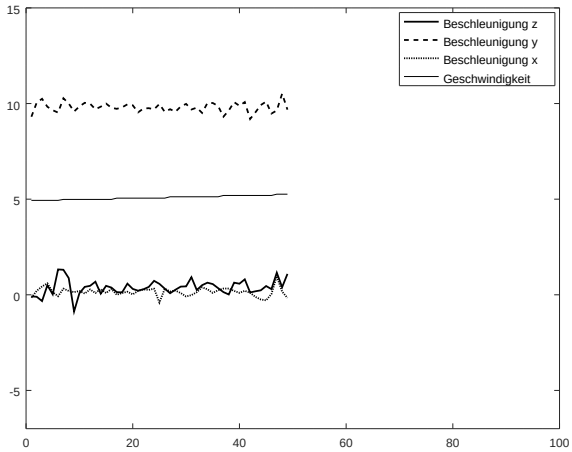
B. Anhang

B.1 Messreihen

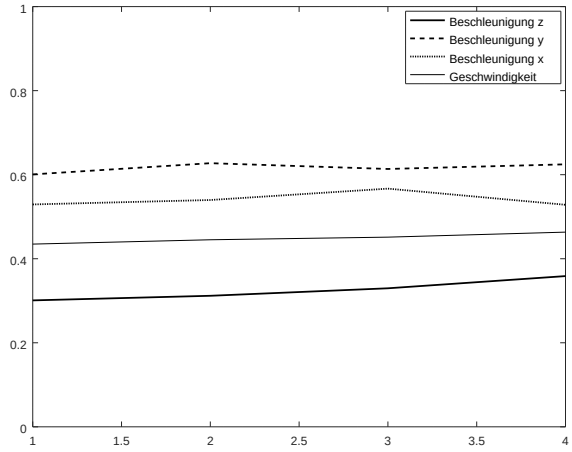


(a) Messreihe 1 Stillstand (b) Messreihe 1 Stillstand (normalisiert)

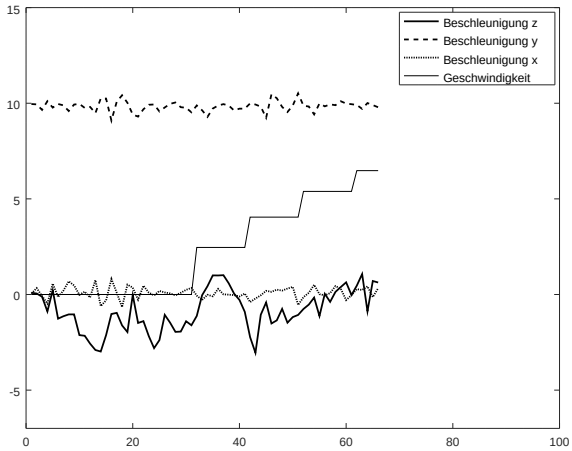
Abbildung B.1: Gegenüberstellung originale und normalisierte Messreihen (1) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



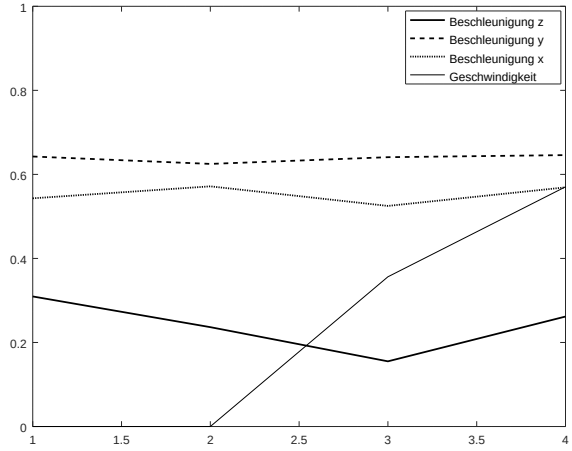
(a) Messreihe 1 Geradeaus



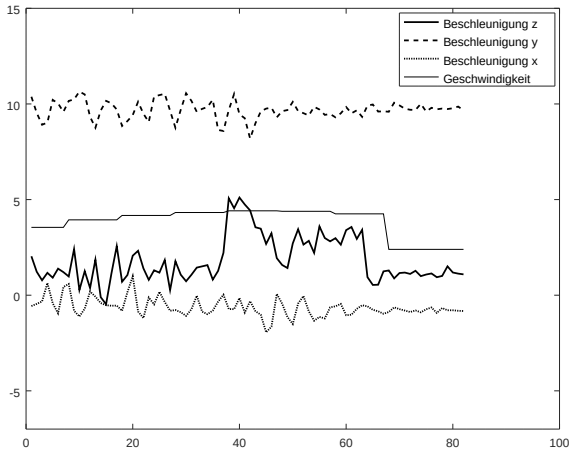
(b) Messreihe 1 Geradeaus (normalisiert)



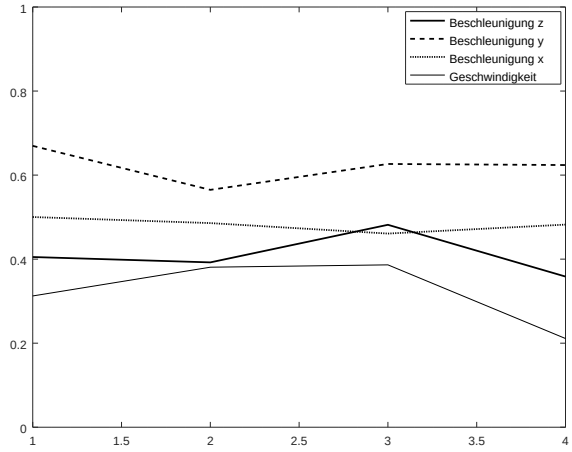
(c) Messreihe 1 Anfahren



(d) Messreihe 1 Anfahren (normalisiert)

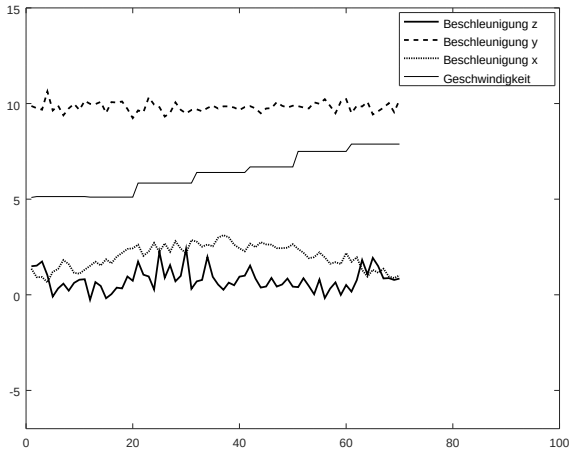


(e) Messreihe 1 Normalbremsung

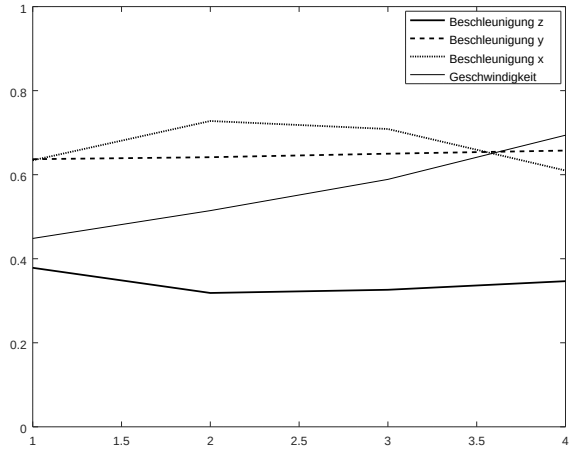


(f) Messreihe 1 Normalbremsung (normal.)

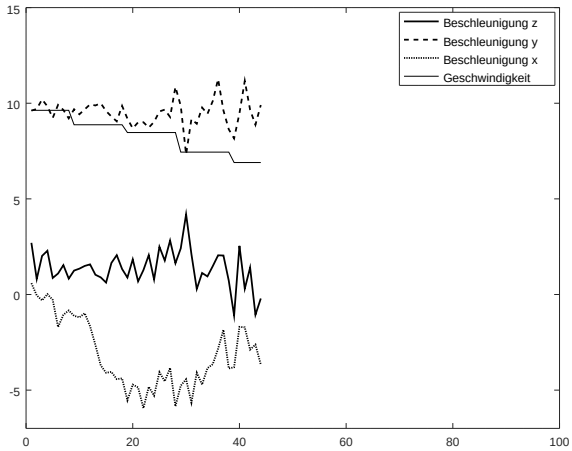
Abbildung B.2: Gegenüberstellung originale und normalisierte Messreihen (2) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



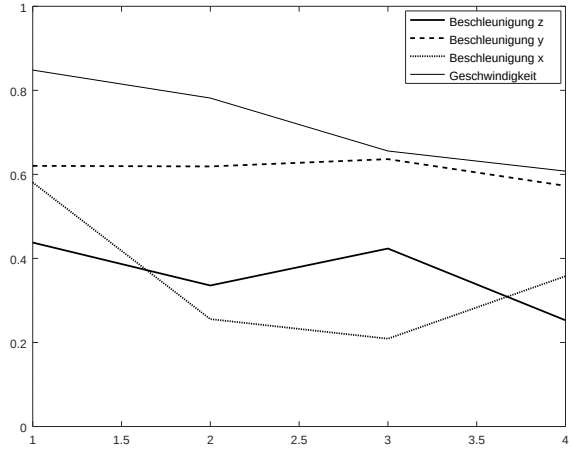
(a) Messreihe 1 Rechtskurve



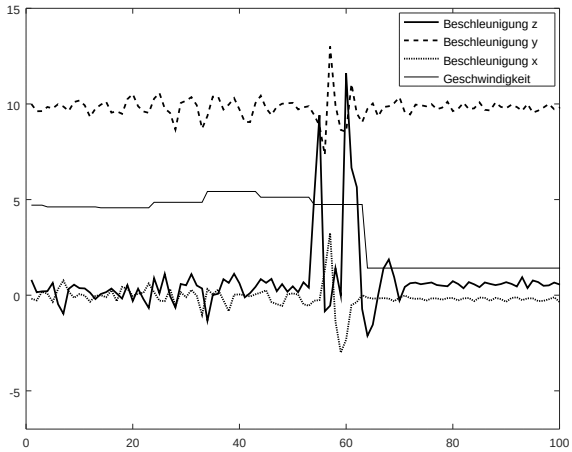
(b) Messreihe 1 Rechtskurve (normalisiert)



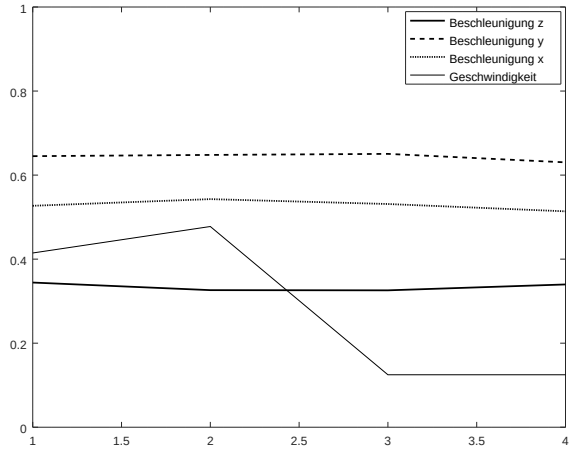
(c) Messreihe 1 Linkskurve



(d) Messreihe 1 Linkskurve (normalisiert)



(e) Messreihe 1 Vollbremsung



(f) Messreihe 1 Vollbremsung (normalisiert)

Abbildung B.3: Gegenüberstellung originale und normalisierte Messreihen (3) (Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

C. Anhang

C.1 Skripte des Oktave-Projekts

```
#configure experiment and script environment
for dokuNo=1:100

  diary on;
  "################################################"
  cfgExperiment;
  #print cfg infos
  "cfg-information:"
  N
  createNewNetwork
  makePlots

  "Starting experiment ..."
  "------------------------------------------------"
  "Preprocessing samples ..."
  #samplePreprocessing();
  "------------------------------------------------"
  "Create reservoir and net ..."
  createReservoir();
  "------------------------------------------------"
  "reading reservoir states ..."
  readReservoirStates();
  "------------------------------------------------"
  "computing conceptors ..."
  computeConceptors();
  "------------------------------------------------"
  "testing classification ..."
  testClassification(dokuNo);

  "all work done ..."
  "################################################"
  diary off;
endfor
```

Quelltext C.1: startExperiment.m

```
addpath("lib");

#classes = {
#             {"classname",
#               classindex,
#               total number of samples,
#               use first n for training
```

```
#              }
#          };

#in case of new classes also mdify computeConceptors.m and testClassification.m

global classes = {
            {"Anfahren",         1, 11, 8},
            {"Normalbremsung",   2,  8, 8},
            {"Rechtskurve",      3, 17, 8},
            {"Linkskurve",       4, 15, 8},
            {"Stillstand",       5, 10, 8},
            {"Geradeaus",        6,  9, 8},
            {"Vollbremsung",     7,  8, 8}
          };

#number of dimensions in samples (acceleration x,y,z and speed
global sampleDimensions = 4;

#neural network configuration
global N = 10;

#global spectralRadiusReservoir = 1;  #not implemented yet
global createNewNetwork = true;

#experiment documentation
global makePlots = false;

global saveWout = false;

#file directories
global pathConceptors = "conceptors/";
global pathPPSamples = "ppSamples/";
global pathReservoirStates ="reservoir_states/";
global pathSamples = "samples/";
global pathPlots = "plots/";
global pathMisc = "misc/";
```

Quelltext C.2: cfgExperiment.m

```
function samplePreprocessing()

  cfgExperiment;

  "Search min and max values within all samples"

  numberOfClasses = numel(classes);
  maxValuesTmp = zeros(sampleDimensions, 1);
  minValuesTmp = zeros(sampleDimensions, 1);

  for i=1:numberOfClasses
    #[maxValuesTmp, minValuesTmp] = searchMinAndMax("../../data/Anfahren/", "
    Anfahren", 11);
    className = classes{i}{1};

    #searching within training samples
    numberOfSamples = classes{i}{3};
    pathSource = [pathSamples className "/"];
    [maxV, minV] = searchMinAndMax(pathSource, className, numberOfSamples);
    maxValuesTmp = [maxValuesTmp maxV];
    minValuesTmp = [minValuesTmp minV];

  endfor

  #min max values for each acceleraition axis and speed
  maxValues = zeros(sampleDimensions,1);
  minValues = zeros(sampleDimensions,1);

  for i=1:sampleDimensions
    maxValues(i,1) = max(maxValuesTmp(i,:));
    minValues(i,1) = min(minValuesTmp(i,:));
  endfor
```

```
32
33    "Subsampling"
34    for i=1:numberOfClasses
35      className = classes{i}{1};
36      numberOfSamples = classes{i}{3};
37      pathSource = [pathSamples className "/"];
38      pathDestination = [pathPPSamples "/"];
39
40      subsampling(pathSource, className, numberOfSamples, pathDestination, maxValues,
         minValues);
41    endfor
42
43  endfunction
```

Quelltext C.3: samplePreprocessing.m

```
1  function [maxValues, minValues] = searchMinAndMax(sampleFilePath, sampleName,
       numOfSamples)
2
3    #search min and max value in each dimension
4
5    ["Search Min and Max in " sampleName]
6    cfgExperiment;
7
8    maxValues = zeros(4,numOfSamples);
9    minValues = zeros(4,numOfSamples);
10
11   #search min and max values
12   for i=1:numOfSamples
13     sampleFileName = [sampleName "_" num2str(i,"%03d") ".csv"];
14     sample = dlmread([sampleFilePath sampleFileName], ",");
15     sample = rot90( sample(2:end, :) ); #skip csv header and rotate matrix
16     for j=1:4
17       maxValues(j, i) = max(sample(j, :));
18       minValues(j, i) = min(sample(j, :));
19     endfor
20
21
22     #make plotes for each sample
23     if (makePlots)
24       newplot;
25       plot( 1:columns(sample), sample(1, :), ";Beschleunigung z;", 'linestyle', '-'
       , 'color', 'k', 'linewidth', 1.5,
26             1:columns(sample), sample(2, :), ";Beschleunigung y;", 'linestyle', '--
       ', 'color', 'k', 'linewidth', 1.5,
27             1:columns(sample), sample(3, :), ";Beschleunigung x;", 'linestyle', ':'
       , 'color', 'k', 'linewidth', 1.5,
28             1:columns(sample), sample(4, :), ";Geschwindigkeit;", 'linestyle', '-',
        'color', 'k');
29       #title([sampleName "\\_"  num2str(i,"%03d")]);
30       #legend ("show", "location", "northeastinside");
31       xlim([0 100]);
32       ylim([-7 15]);
33
34       #graphics_toolkit("gnuplot");
35       print([pathPlots sampleName "_raw_sw_"  num2str(i,"%03d") ".pdf"],  "-
       landscape");
36
37     endif
38
39
40   endfor
41
42
43 endfunction
```

Quelltext C.4: searchMinAndMax.m

```
1  function subsampling(sampleFilePath, sampleName, numOfSamples, destinationPath,
       maxValue, minValue)
```

```
2
3   ["Subsampling " sampleName]
4   cfgExperiment;
5
6   #schift-scaling (1) -> jaeger 2014 p. 74
7   for i=1:numOfSamples
8
9     sampleFileName = [sampleName "_" num2str(i,"%03d") ".csv"];
10    sample = dlmread([sampleFilePath sampleFileName], ",");
11    sample = rot90( sample(2:end, :) ); #skip csv header and rotate matrix
12
13    dimensions = rows(sample);
14
15    for j=1:dimensions
16      #shifting each dimension
17      shiftingValue = 0;
18      if (minValue(j,1) < 0)
19        shiftingValue = abs(minValue(j,1));
20        sample(j,:) = sample(j,:) + shiftingValue;
21      endif
22      #scaling each dimension
23      if ( ( maxValue(j,1) + shiftingValue ) != 0)
24        #speed at stillstand is 0
25        sample(j,:) = sample(j,:) / ( maxValue(j,1) + shiftingValue );
26      endif
27    endfor
28
29    subsample=zeros(4,4);
30
31    for j=1:4
32      #interpolating by cubic polynomial and subsampling (2)+(3) -> jaeger 2014 p.
      74
33        offset = ceil( columns(sample) / 3 ) -1;
34      LES = zeros(4,4);
35      Y = zeros(4,1);
36      supportPoint=1;
37      for x=1:4
38        #extracting point for interpolation with cubic polynomial
39        Y(x, 1) = sample(j, (supportPoint) );
40        LES(x, :) = [ (supportPoint)^3, (supportPoint)^2, (supportPoint), 1];
41        supportPoint += offset;
42      endfor
43      #interpolation by cubic polynomial
44      b = LES \ Y;
45      supportPoint = 1;
46      for x=1:4
47        LES(x, :) = [ (supportPoint)^3, (supportPoint)^2, (supportPoint), 1];
48        supportPoint += offset;
49      endfor
50      Y = LES * b;
51
52      subsample(j,:) = rot90(Y);
53      #plot( (1:columns(sample)), sample(j,:), (1:offset:(3*offset+1)), Y')  #debug
       output_max_field_width
54    endfor
55
56    dlmwrite([destinationPath "pp_" sampleName "_"  num2str(i,"%03d") ".csv"],
      rot90(subsample,-1), ",");
57
58    if (makePlots)
59      #make plotes for each sample
60      plot( 1:columns(subsample), subsample(1, :), ";Beschleunigung z;", 'linestyle
      ', '-', 'color', 'k', 'linewidth', 1.5,
61            1:columns(subsample), subsample(2, :), ";Beschleunigung y;", 'linestyle
      ', '--', 'color', 'k', 'linewidth', 1.5,
62            1:columns(subsample), subsample(3, :), ";Beschleunigung x;", 'linestyle
      ', ':', 'color', 'k', 'linewidth', 1.5,
63            1:columns(subsample), subsample(4, :), ";Geschwindigkeit;", 'linestyle'
      , '-', 'color', 'k');
64      ylim([0 1]);
65      print([pathPlots sampleName "_pp_sw_"  num2str(i,"%03d") ".pdf"], "-landscape
      " );
```

```
66      endif
67
68
69    endfor
70
71    if (makePlots)
72
73      subplot(2,2,1);
74      plot( 1:4,subsample(1,:) );
75      title([sampleName " Beschleunigung z"]);
76      #legend ("show", "location", "northeastoutside");
77      ylim([0 1]);
78      subplot(2,2,2);
79      plot( 1:4,subsample(2,:));
80      title([sampleName " Beschleunigung y"]);
81      #legend ("show", "location", "northeastoutside");
82      ylim([0 1]);
83      subplot(2,2,3);
84      plot( 1:4,subsample(3,:));
85      title([sampleName " Beschleunigung x"]);
86      #legend ("show", "location", "northeastoutside");
87      ylim([0 1]);
88      subplot(2,2,4);
89      plot( 1:4,subsample(4,:));
90      title([sampleName " Geschwindigkeit"]);
91      #legend ("show", "location", "northeastoutside");
92      ylim([0 1]);
93
94      print([pathPlots "pp_" sampleName ".pdf"], "-landscape" );
95
96      close();
97
98    endif
99
100 endfunction
```

Quelltext C.5: subsampling.m

```
1   function createReservoir()
2
3     cfgExperiment;
4     numberOfClasses = numel(classes);
5     create = createNewNetwork; #create new network (if true) only once for the whole
        experiment
6
7     #create reservoir and evaluate with different samples
8
9     for i=1:numberOfClasses
10
11      className = classes{i}{1};
12      #reading pp_Sample (these sample files have no header)
13      sample = dlmread([pathPPSamples "pp_" className "_001.csv"], ",");
14      sample = rot90( sample );
15
16      [Out,M,X] = predict(create,sample,N,4);
17
18      y = compute(M,X(:,1),columns(sample));
19
20      ["NRMSE for " className]
21      nrmse( sample(:,:), y(1:4,:) )
22
23      create = false; #use once created reservoir for further tests
24
25      if (makePlots)
26
27        subplot(2,2,1);
28        nrmseVal = num2str(nrmse(sample(1,:),y(1,:)));
29        plot((1:columns(sample)), sample(1,:), ";teacher values;", (1:columns(y)), y
      (1,:), ";repeated values;")
30        title( [strrep(className, "_", "\\_") " - Z-Achse (NRMSE: " nrmseVal ")"] );
31        xlabel( "t [100 ms] " );
```

```
32        ylabel("a [m/s^2]");
33
34        subplot(2,2,2);
35        nrmseVal = num2str(nrmse(sample(2,:),y(2,:)));
36        plot((1:columns(sample)), sample(2,:), ";teacher values;", (1:columns(y)), y
      (2,:), ";repeated values;")
37        title( [strrep(className, "_", "\\_") " - Y-Achse (NRMSE: " nrmseVal ")"] );
38        xlabel( "t [100 ms] " );
39        ylabel("a [m/s^2]");
40
41        subplot(2,2,3);
42        nrmseVal = num2str(nrmse(sample(3,:),y(3,:)));
43        plot((1:columns(sample)), sample(3,:), ";teacher values;", (1:columns(y)), y
      (3,:), ";repeated values;")
44        title( [strrep(className, "_", "\\_") " - X-Achse (NRMSE: " nrmseVal ")"] );
45        xlabel( "t [100 ms] " );
46        ylabel("a [m/s^2]");
47
48        subplot(2,2,4);
49        nrmseVal = num2str(nrmse(sample(4,:),y(4,:)));
50        plot((1:columns(sample)), sample(4,:), ";teacher values;", (1:columns(y)), y
      (4,:), ";repeated values;")
51        title( [strrep(className, "_", "\\_") " - Geschwindigkeit (NRMSE: " nrmseVal
      ")"] );
52        xlabel( "t [100 ms] " );
53        ylabel("v [m/s]");
54
55        print([pathPlots className "_nrmse_sw_"  num2str(i,"%03d") ".pdf"], "-
      landscape" );
56
57        endif
58
59    endfor
60
61  endfunction
```

Quelltext C.6: createReservoir.m

```
1  function result = nrmse(In,Out,Mode=1,Flag=1)
2  % normalised root mean square error
3
4  % In : given sequence
5  % Out : computed sequence
6  % Mode : kind of normalisation
7  % Flag : 1 averaged value or per row
8
9  result = (Out-In).^2;
10 switch (Mode)
11   case 0  % without normalisation
12     result = sum(result,2);
13   case 1 % wrt. number of elements
14     result = mean(result,2);
15   case 2 % divided by variance
16     result = mean(result,2)./var(In,0,2);
17 endswitch
18
19 if (Flag)
20   result = mean(result);
21 endif
22
23 result = sqrt(result);
24 endfunction
```

Quelltext C.7: nrmse.m (Zur Verfügung gestellt von Prof. Dr. Stolzenburg)

```
1  function [Out,M,X,reservoirStates] = predict(create=1,In,N=5,m=5)
2
3    cfgExperiment;
4
5  % extrapolate input sequence by recurrent neural networks
```

```
6
7  % In : input sequence
8  % Out : output sequence
9  % N : reservoir size (number of neurons)
10 % m : extrapolation for m more steps
11 % V : learned output weights [Wout]
12 % W : input and reservoir weight matrix [Win W]
13 % X : input and reservoir state sequence [X; R]
14
15 % problem dimension
16 n = columns(In); % sequence length
17 d = rows(In); % number of inputs, normally = 1
18
19
20 #otto 27.02.18 changes: save or load W and X
21 X = zeros(d+N,n+m-1); % input and reservoir state sequence
22 if (create==1)
23   % network initialization (randomly)
24   W = zeros(N,d+N); % input and reservoir weight matrix [Win Wres]
25   W(:,1:d) = randn(N,d)/sqrt(d); % balanced input weights [Win]
26
27   [   W(:,d+(1:N)),     X(d+(1:N),1)    ] = reservoir(N); % [Wres,x];
28
29   x = X(d+(1:N),1); #extract initial reservoir state vector
30   save ("-mat7-binary", "network_part_Win_Wres_x.mat", "W", "x");
31 else
32   load ("network_part_Win_Wres_x.mat", "W", "x");
33   X(d+(1:N),1) = x; #restore initial reservoir state vector
34 endif
35 #otto 27.02.18 end changes
36
37 % drive given input through reservoir
38 X(1:d,1:n) = In; % fill in given input
39 for (k=2:n)
40   X(d+(1:N),k) = tanh(W*X(:,k-1));
41 endfor
42
43 % learn output weights
44 Y = In(:,2:n); % predicted sequence
45 V = Y/X(:,1:n-1); % output weights [Wout]
46
47 % extrapolation of sequence
48 for (k=n+(1:m-1))
49   X(:,k) = tanh([V;W]*X(:,k-1));
50 endfor
51
52 % predicted sequence
53 reservoirStates = X(d+1:end,:);
54 Out = [In(:,1),V*X(:,1:end-(m==0))];
55 M = [V;W]; % overall square matrix [Wout; Win Wres]
56
57
58 endfunction
```

Quelltext C.8: predict.m (Zur Verfügung gestellt von Prof. Dr. Stolzenburg [1])

```
1  function readReservoirStates()
2
3    cfgExperiment;
4    numberOfClasses = numel(classes);
5    create = false; #network already created
6
7    for classIndex=1:numberOfClasses
8
9      className = classes{classIndex}{1};
10     numberOfSamples = classes{classIndex}{3};
11
12     allNRMSEs = zeros(4,numberOfSamples);
13     ["reading reservoir states for " className " (" num2str(numberOfSamples) "
       pp_samples)"]
14
```

```
    for i=1:numberOfSamples
      sampleFileName = ["pp_" className "_" num2str(i,"%03d") ".csv"];
      sample = dlmread([pathPPSamples sampleFileName], ",");
      sample = rot90( sample ); #rotate matrix

      [Out,M,X,reservoirStates] = predict(create,sample,N,0);
      y = compute(M,X(:,1),columns(sample));

      if (saveWout)
        V=M(1:sampleDimensions,:);
        dlmwrite([pathMisc "Wout_" className "_"  num2str(i,"%03d") ".csv"], V, ","
    );
      endif

      allNRMSEs(:,i) = nrmse( sample(:,:), y(1:4,:) );
      dlmwrite([pathReservoirStates "reservoir_states_" className "_"  num2str(i,"
    %03d") ".csv"],    reservoirStates , ",");

      if (makePlots)
        plot (sample);
      endif

    endfor

    "max NRMSE per dimension:"
    max(allNRMSEs(1,:))
    max(allNRMSEs(2,:))
    max(allNRMSEs(3,:))
    max(allNRMSEs(4,:))

  endfor

endfunction
```

Quelltext C.9: readReservoirStates.m

```
function computeConceptors()

  cfgExperiment;
  numberOfClasses = numel(classes);

  #we need R_all to compute the negative evidence conceptors
  R_all = zeros( 4*(sampleDimensions+N));
  totalNumberOfTrainingSamples = 0;

  #compute positive conceptors
  "Compute prelimC, R, Cpos and aperture for all sample classes"
  for classIndex=1:numberOfClasses

    className = classes{classIndex}{1};
    numberOfTrainingSamples = classes{classIndex}{4};

    #if ( classIndex!=4 ) #skip linkskurve + +

      #exlude classes with 0 training samples
      if (numberOfTrainingSamples>0)

        [prelimC, Z, Rnorm, R] = computePreliminaryConceptor(className,
    numberOfTrainingSamples);
        bestPosAperture = computeBestAperture(prelimC, true);
        #Cpos = computeFinalConceptor(Rnorm, bestPosAperture, false);

        Cpos = prelimC;

        # output
        [className]
        ["Trainig samples : " num2str(numberOfTrainingSamples)]
        ["Aperture : " num2str(bestPosAperture)]

        totalNumberOfTrainingSamples += numberOfTrainingSamples;
        R_all += R;
```

```
35
36          dlmwrite([pathConceptors "prelimC_" className ".csv"],    prelimC  , ",");
37          dlmwrite([pathConceptors "Rnorm_" className ".csv"],    Rnorm  , ",");
38          dlmwrite([pathConceptors "R_" className ".csv"],    R  , ",");
39          dlmwrite([pathConceptors "Cpos_" className ".csv"],    Cpos  , ",");
40
41        else
42          [className " skipped by cfgExperiment ..."]
43        endif
44
45      #endif
46    endfor
47    ["conceptors were computed with " num2str(totalNumberOfTrainingSamples) " samples
        "]
48
49    #compute negative evidence conceptors
50    "Compute Cneg and aperture for all sample classes"
51    for classIndex=1:numberOfClasses
52
53      className = classes{classIndex}{1};
54      numberOfTrainingSamples = classes{classIndex}{4};
55
56      #if ( classIndex!=4 ) #skip linkskurve + +
57
58        #exlude classes with 0 training samples
59        if (numberOfTrainingSamples>0)
60
61          R = dlmread([pathConceptors "R_" className ".csv"], ",");
62
63          R_otherNorm = (R_all - R) / (totalNumberOfTrainingSamples -
      numberOfTrainingSamples);
64          Cother = R_otherNorm * inv(R_otherNorm + eye(rows(R_otherNorm))); #booelan
      OR operation Jaeger2014 R3 p.51 equation25
65          prelimCneg = eye(rows(Cother)) - Cother; #booelan NOT operation Jaeger2014
      R3 p.52 equation 25
66          bestNegAperture = computeBestAperture(prelimCneg, true);
67          Cneg = computeFinalConceptor(prelimCneg, bestNegAperture, true);
68
69          [className]
70          ["Aperture : " num2str(bestNegAperture)]
71
72          dlmwrite([pathConceptors "Cneg_" className ".csv"],    Cneg  , ",");
73
74        else
75          [className " skipped by cfgExperiment ..."]
76        endif
77
78      #endif
79    endfor
80
81    #further computations of conceptors for investigations ....
82
83    "Compute CAll" #based on correlation matrizes
84    R_otherNorm = (R_all) / (totalNumberOfTrainingSamples);
85    prelimCAll = R_otherNorm * inv(R_otherNorm + eye(rows(R_otherNorm))); #booelan OR
        operation Jaeger2014 R3 p.51 equation25
86    bestPosAperture = computeBestAperture(prelimCAll, true);
87    CAll = computeFinalConceptor(prelimCAll, bestPosAperture, true);
88    ["Aperture : " num2str(bestPosAperture)]
89    dlmwrite([pathConceptors "CAll_old.csv"],    Cneg  , ",");
90
91    "Compute CNotAny"
92    #R_otherNorm = (R_all) / (totalNumberOfTrainingSamples);
93    #prelimCNotAny = R_otherNorm * inv(R_otherNorm + eye(rows(R_otherNorm))); #
        booelan OR operation Jaeger2014 R3 p.51 equation25
94    prelimCNotAny = eye(rows(CAll)) - CAll; #booelan NOT operation Jaeger2014 R3 p.52
        equation 25
95    bestNegAperture = computeBestAperture(prelimCNotAny, true);
96    CNotAny = computeFinalConceptor(prelimCNotAny, bestNegAperture, true);
97    ["Aperture : " num2str(bestNegAperture)]
98    dlmwrite([pathConceptors "CNotAny_old.csv"],    Cneg  , ",");
99
```

```
100   #in case of new classes add them here :
101   #based on conceptors
102
103   CposAnfahren = dlmread([pathConceptors "Cpos_Anfahren.csv"], ",");
104   CposNormalbremsung = dlmread([pathConceptors "Cpos_Normalbremsung.csv"], ",");
105   CposRechtskurve = dlmread([pathConceptors "Cpos_Rechtskurve.csv"], ",");
106   CposLinkskurve = dlmread([pathConceptors "Cpos_Linkskurve.csv"], ",");
107   CposStillstand = dlmread([pathConceptors "Cpos_Stillstand.csv"], ",");
108   CposGeradeaus = dlmread([pathConceptors "Cpos_Geradeaus.csv"], ",");
109   CposVollbremsung = dlmread([pathConceptors "Cpos_Vollbremsung.csv"], ",");
110
111   I = eye(rows(CposAnfahren));
112   CposAnfahrenNot = I - CposAnfahren;
113   CposNormalbremsungNot = I - CposNormalbremsung;
114   CposRechtskurveNot = I - CposRechtskurve;
115   CposLinkskurveNot = I - CposLinkskurve;
116   CposStillstandNot = I - CposStillstand;
117   CposGeradeausNot = I - CposGeradeaus;
118   CposVollbremsungNot = I - CposVollbremsung;
119
120   #test what happens if one class is unknown -> linkskurve
121   CnotAny = inv(  inv(CposAnfahrenNot) + inv(CposNormalbremsungNot) + inv(
        CposRechtskurveNot)  + inv(CposLinkskurveNot) + inv(CposStillstandNot) + inv(
        CposGeradeausNot) + inv(CposVollbremsungNot) - I );
122   bestPosAperture = computeBestAperture(CnotAny, true);
123   CnotAny = computeFinalConceptor(CnotAny, bestPosAperture, true);
124
125   CorAll = I - CnotAny;
126   bestNegAperture = computeBestAperture(CorAll, true);
127   CorAll = computeFinalConceptor(CorAll, bestNegAperture, true);
128
129   dlmwrite([pathConceptors "CNotAny.csv"],   CnotAny , ",");
130   dlmwrite([pathConceptors "CAll.csv"],   CorAll , ",");
131
132 endfunction
```

Quelltext C.10: computeConceptors.m

```
1  function [Cprelim, Z, Rnorm, R] = computePreliminaryConceptor(sampleName,
       numOfSamples)
2
3    cfgExperiment;
4
5    Z = zeros(4*(sampleDimensions+N), numOfSamples);
6
7    #assembling network state matrix
8    for i=1:numOfSamples
9
10     z = createCombinedStateVector(sampleName, i);
11     Z(:,i) = z;
12
13   endfor
14
15   #computing correlation matrix
16   R = Z * Z';
17   Rnorm = R / columns(Z);
18
19   #computing preliminary conceptor
20   Cprelim = Rnorm * inv( Rnorm + eye(rows(Rnorm)) );
21
22 endfunction
```

Quelltext C.11: computePreliminaryConceptor.m

```
1  function [z] = createCombinedStateVector(sampleName, sampleNumber)
2
3    cfgExperiment;
4
5    #reading reservoir states
6    reservoirFileName = ["reservoir_states_" sampleName "_" num2str(sampleNumber,"%03
       d") ".csv"];
```

```
  reservoirStates = dlmread([pathReservoirStates reservoirFileName], ",");
  #reading preprocessed sample
  ppSampleFileName = ["pp_" sampleName "_" num2str(sampleNumber,"%03d") ".csv"];
  ppSample = dlmread([pathPPSamples ppSampleFileName], ",");
  ppSample = rot90( ppSample );

  z = [reservoirStates(:,1); ppSample(:,1)];
  z = [z; reservoirStates(:,2); ppSample(:,2)];
  z = [z; reservoirStates(:,3); ppSample(:,3)];
  z = [z; reservoirStates(:,4); ppSample(:,4)];

endfunction
```

Quelltext C.12: createCombinedStateVector.m

```
function [bestPosAperture] = computeBestAperture(matrix, isConceptor=false)

  cfgExperiment;

  #Jaeger 2014 Matlab Scripts -> fig24and25and26_JapVowels_Report.m from row 283
  #R corelation matrix

  I = eye(rows(matrix));

  maxExponent = 9;
  phiNorm = zeros(1, maxExponent+1);

  R = matrix;
  C = matrix;

  #compute support points
  for i=0:maxExponent

    if (isConceptor == false)
      #compute with correlation matrix R
      phi = R * inv(R + (2^i)^(-2) * I);  #phi
    else
      #compute with conceptor matrix C
      phi = C * inv( C + (2^i)^(-2) * (I-C) );
    endif
    phiNorm(1,i+1) = norm(phi , "fro")^2; #forbenius norm

  endfor

  #interpolate
  interpolationPoints = 0:0.01:maxExponent;
  posIntpl = interp1( 0:1:maxExponent, phiNorm, interpolationPoints, "spline");

  #gradient
  posIntplGrad = ( posIntpl(1,2:end) - posIntpl(1,1:end-1) ) / 0.01;
  posIntplGrad = [posIntplGrad posIntplGrad(end)];

  #determine max value and index
  [maxVal maxIndexPos] = max(abs(posIntplGrad));

  bestPosAperture = 2^interpolationPoints(maxIndexPos);

  if (makePlots)
    plot( interpolationPoints, posIntplGrad);
  endif


endfunction
```

Quelltext C.13: computeBestAperture.m

```
function testClassification(dokuNo)

  cfgExperiment;

```

```
numberOfClasses = numel(classes);

#in case of new classes add them here :
#loading positive and negative evidence conceptors
Cpos_Anfahren = dlmread([pathConceptors "Cpos_Anfahren.csv"], ",");
Cpos_Normalbremsung = dlmread([pathConceptors "Cpos_Normalbremsung.csv"], ",");
Cpos_Rechtskurve = dlmread([pathConceptors "Cpos_Rechtskurve.csv"], ",");
Cpos_Linkskurve = dlmread([pathConceptors "Cpos_Linkskurve.csv"], ",");
Cpos_Stillstand = dlmread([pathConceptors "Cpos_Stillstand.csv"], ",");
Cpos_Geradeaus = dlmread([pathConceptors "Cpos_Geradeaus.csv"], ",");
Cpos_Vollbremsung = dlmread([pathConceptors "Cpos_Vollbremsung.csv"], ",");

Cneg_Anfahren = dlmread([pathConceptors "Cneg_Anfahren.csv"], ",");
Cneg_Normalbremsung = dlmread([pathConceptors "Cneg_Normalbremsung.csv"], ",");
Cneg_Rechtskurve = dlmread([pathConceptors "Cneg_Rechtskurve.csv"], ",");
Cneg_Linkskurve = dlmread([pathConceptors "Cneg_Linkskurve.csv"], ",");
Cneg_Stillstand = dlmread([pathConceptors "Cneg_Stillstand.csv"], ",");
Cneg_Geradeaus = dlmread([pathConceptors "Cneg_Geradeaus.csv"], ",");
Cneg_Vollbremsung = dlmread([pathConceptors "Cneg_Vollbremsung.csv"], ",");

CnotAny = dlmread([pathConceptors "CNotAny.csv"], ",");
CAll = dlmread([pathConceptors "CAll.csv"], ",");

#save for documentation
currentTestPath="15_N10_Vollbremsung/";
destinationPath = ["doku/" currentTestPath];

hStatsAllTestData=zeros(numberOfClasses,3);
hStatsAllTrainingsData=zeros(numberOfClasses,3);
for classIndex=1:numberOfClasses

  className = classes{classIndex}{1};
  numberOfTrainingSamples = classes{classIndex}{4}; #per class
  totalNumberOfSamples = classes{classIndex}{3}; #per class

  "........................................................"
  ["testing sample recognition for : " className]
  ["positive recognition if index is " num2str(classes{classIndex}{2}) ]

  classifications = zeros(totalNumberOfSamples,3);
  for i=1:totalNumberOfSamples

    z = createCombinedStateVector(className, i);

    hplus=zeros(8,1);
    hplus(1,1) = z' * Cpos_Anfahren * z;
    hplus(2,1) = z' * Cpos_Normalbremsung * z;
    hplus(3,1) = z' * Cpos_Rechtskurve * z;
    hplus(4,1) = z' * Cpos_Linkskurve * z;
    hplus(5,1) = z' * Cpos_Stillstand * z;
    hplus(6,1) = z' * Cpos_Geradeaus * z;
    hplus(7,1) = z' * Cpos_Vollbremsung * z;
    hplus(8,1) = z' * CnotAny * z;
    [maxVal maxIndex] = max(hplus);
    [minVal minIndex] = min(hplus);

    #normalize to a range of [0, 1]
    hplus = (hplus - minVal) / (maxVal - minVal);
    [maxVal classifications(i,1)] = max(hplus);

    hminus=zeros(8,1);
    hminus(1,1) = z' * Cneg_Anfahren * z;
    hminus(2,1) = z' * Cneg_Normalbremsung * z;
    hminus(3,1) = z' * Cneg_Rechtskurve * z;
    hminus(4,1) = z' * Cneg_Linkskurve * z;
    hminus(5,1) = z' * Cneg_Stillstand * z;
    hminus(6,1) = z' * Cneg_Geradeaus * z;
    hminus(7,1) = z' * Cneg_Vollbremsung * z;
    hminus(8,1) = z' * CAll * z;
    [maxVal maxIndex] = max(hminus);
    [minVal minIndex] = min(hminus);

```

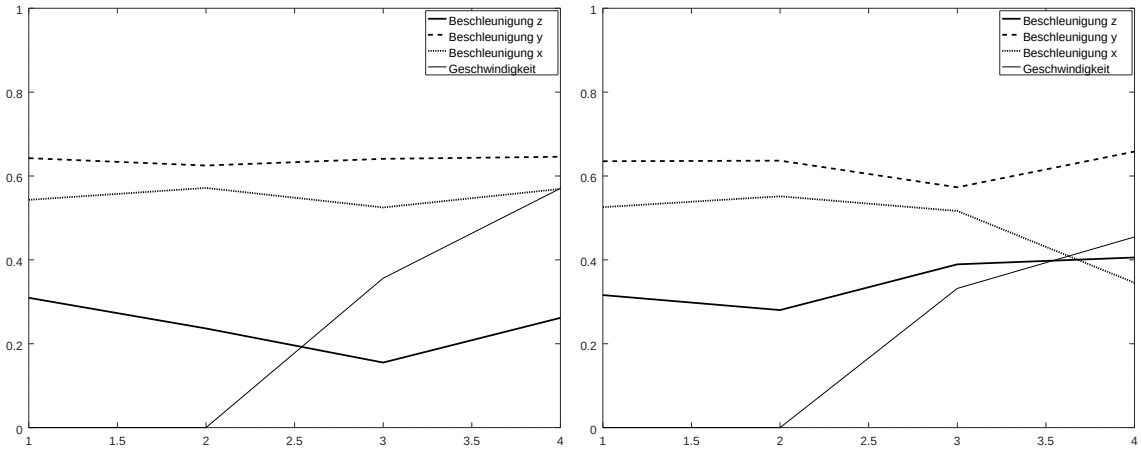
```
77        #normalize to a range of [0, 1]
78        hminus = (hminus - minVal) / (maxVal - minVal);
79        [maxVal classifications(i,2)] = max(hminus);
80
81        hplusminus = (hplus+hminus)/2;
82        [maxVal classifications(i,3)] = max(hplusminus);
83
84      endfor
85
86      #computing classification stats for training data
87      ["recognition of traing data:"]
88      classifications(1:numberOfTrainingSamples,:)
89      hStats=zeros(1,3);
90      for i=1:numberOfTrainingSamples
91        if (classifications(i,1) == classIndex)
92          #h+
93          hStats(1,1) += 1;
94        endif
95        if (classifications(i,2) == classIndex)
96          #h-
97          hStats(1,2) += 1;
98        endif
99        if (classifications(i,3) == classIndex)
100         #h+-
101         hStats(1,3) += 1;
102       endif
103     endfor
104     hStats = hStats / numberOfTrainingSamples *100;
105     hStatsAllTrainingsData(classIndex,:)=hStats;
106     ["h+: "  num2str(hStats(1,1), "%5.3f") "  h-: "   num2str(hStats(1,2), "%5.3f")
        "  h+-: "   num2str(hStats(1,3), "%5.3f")]
107
108     dlmwrite([destinationPath className "_hplus.csv"], classifications(1:
        numberOfTrainingSamples,1)', ",", "-append");
109     dlmwrite([destinationPath className "_hminus.csv"], classifications(1:
        numberOfTrainingSamples,2)', ",", "-append");
110     dlmwrite([destinationPath className "_hplusminus.csv"], classifications(1:
        numberOfTrainingSamples,3)', ",", "-append");
111
112     #computing classification stats for test data
113     ["recognition of test data:"]
114     if (totalNumberOfSamples == numberOfTrainingSamples)
115       ["all existing samples (data) were consumed for training."; "create more
        samples or reduce learning samples"]
116     else
117       classifications(numberOfTrainingSamples+1:end,:)
118
119       hStats=zeros(1,3);
120       for i=numberOfTrainingSamples+1:totalNumberOfSamples
121         if (classifications(i,1) == classIndex)
122           #h+
123           hStats(1,1) += 1;
124         endif
125         if (classifications(i,2) == classIndex)
126           #h-
127           hStats(1,2) += 1;
128         endif
129         if (classifications(i,3) == classIndex)
130           #h+-
131           hStats(1,3) += 1;
132         endif
133       endfor
134       hStats = hStats / (totalNumberOfSamples-numberOfTrainingSamples) *100;
135       hStatsAllTestData(classIndex,:)=hStats;
136       ["h+: "  num2str(hStats(1,1), "%5.3f") "  h-: "   num2str(hStats(1,2), "%5.3f
        ") "  h+-: "   num2str(hStats(1,3), "%5.3f")]
137
138     endif
139
140   endfor
141
142   "Stats for Training Data:"
```

```
143   hStatsAllTrainingsData
144   "Stats for Test Data:"
145   hStatsAllTestData
146
147   #save for documentation
148
149   for i=1:7
150     className = classes{i}{1};
151     dlmwrite([destinationPath className "_Test.csv"], hStatsAllTestData(i,:), ",",
        "-append");
152     dlmwrite([destinationPath className "_Training.csv"], hStatsAllTrainingsData(i
        ,:), ",", "-append");
153   endfor
154   movefile("network_part_Win_Wres_x.mat", [destinationPath num2str(dokuNo,"%03d") "
        _network_part_Win_Wres_x.mat"]);
155   diary off;
156   movefile("diary", [destinationPath num2str(dokuNo,"%03d") "_diary.txt"]);
157
158   clear all
159   clc
160 endfunction
```

Quelltext C.14: testClassification.m

D. Anhang

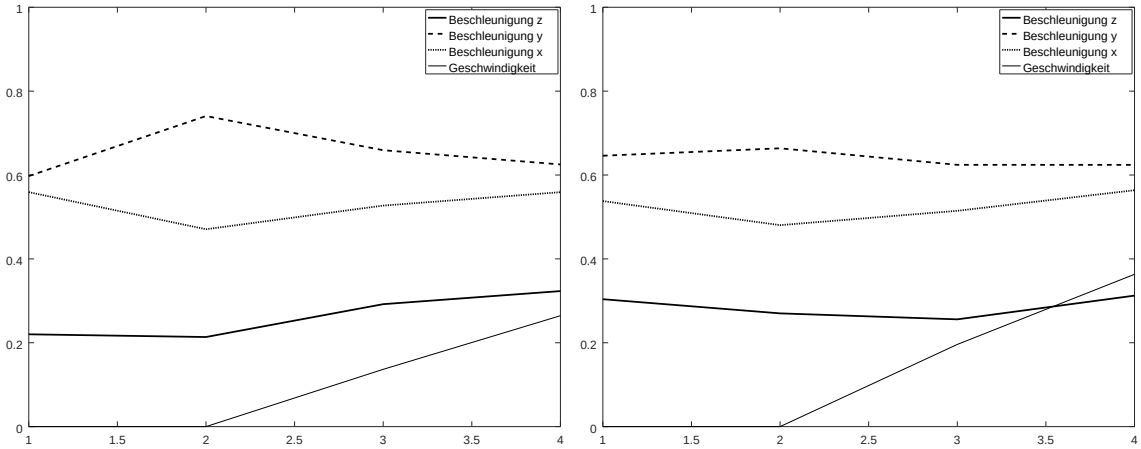
D.1 Aufbereitete Messreihen Anfahren



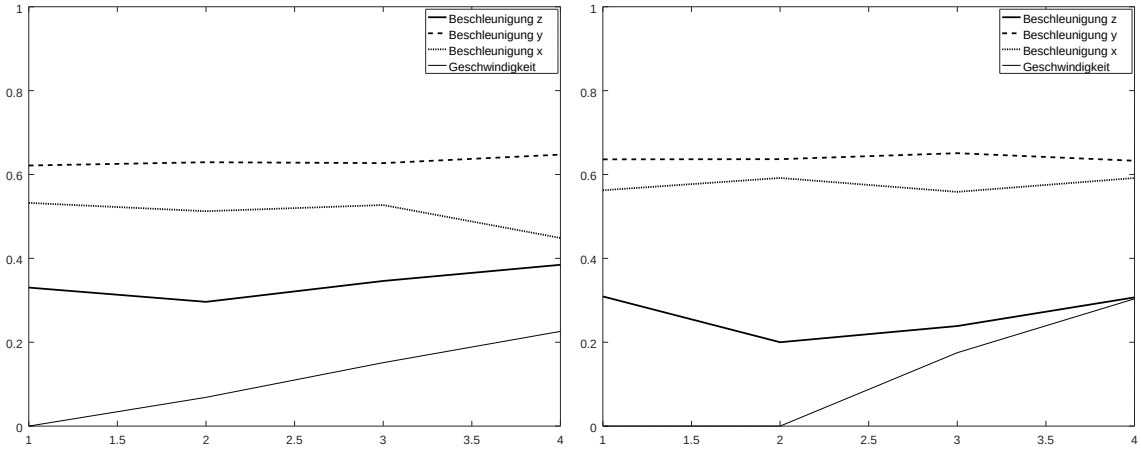
(a) Messreihe 1 Anfahren

(b) Messreihe 2 Anfahren

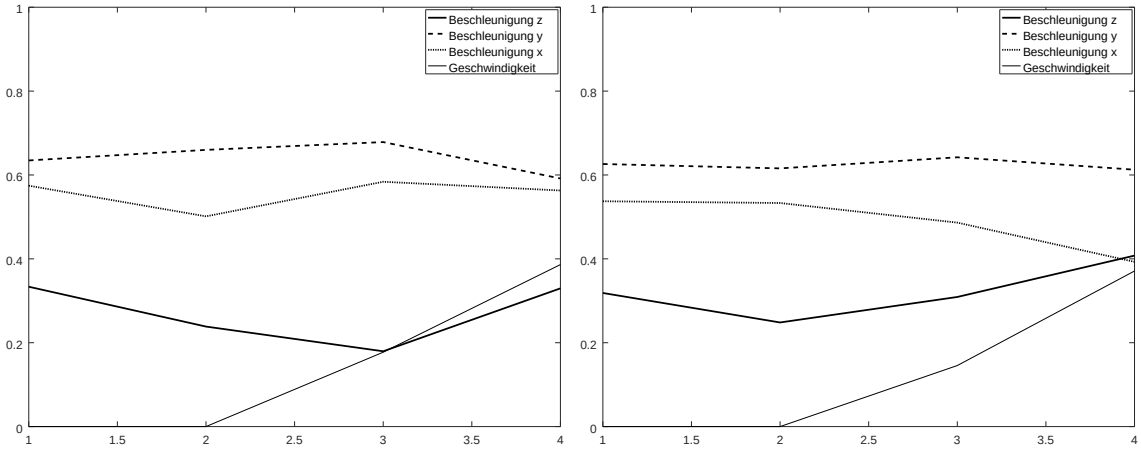
Abbildung D.1: Aufbereitete Messreihen Anfahren (1) (Klassenindex = 1, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



(a) Messreihe 3 Anfahren (b) Messreihe 4 Anfahren



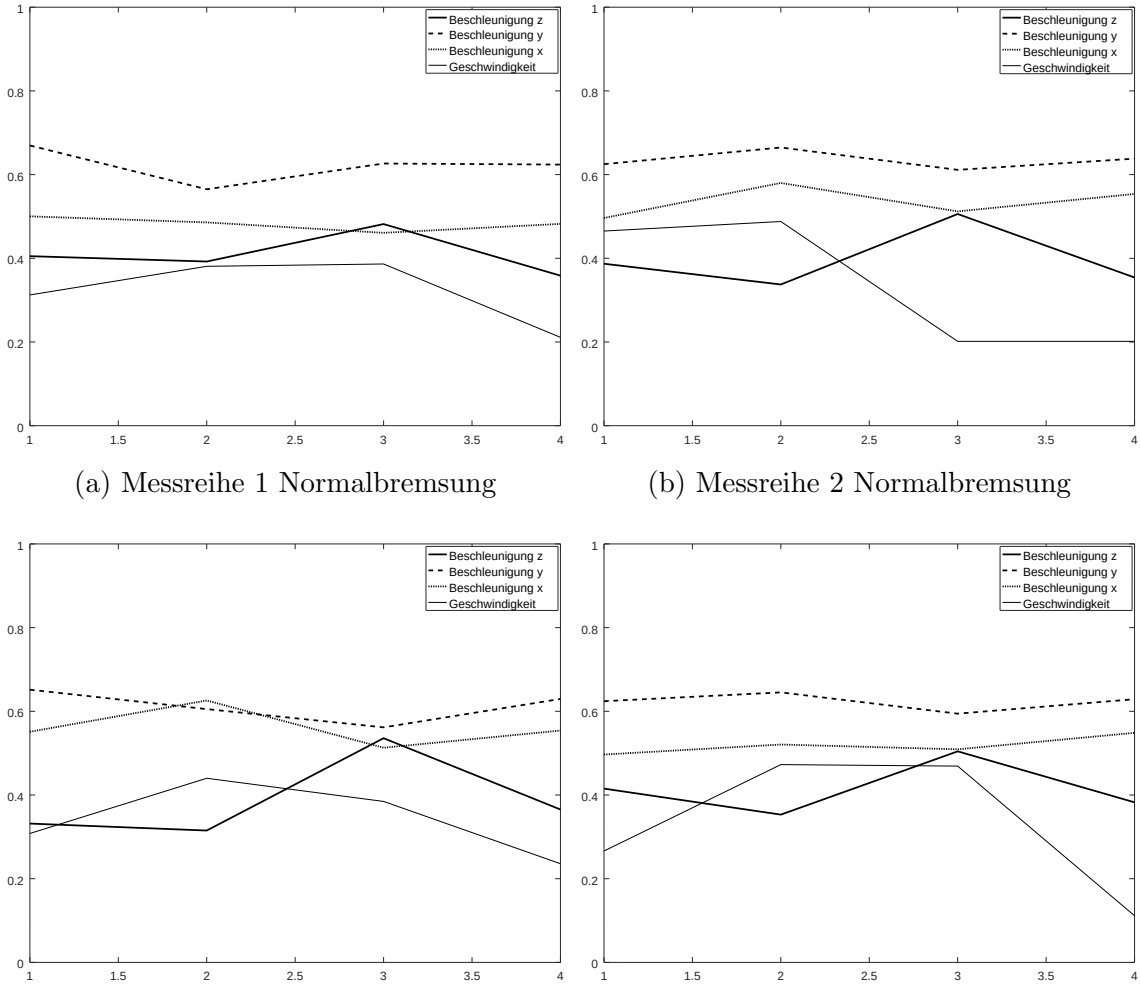
(c) Messreihe 5 Anfahren (d) Messreihe 6 Anfahren



(e) Messreihe 7 Anfahren (f) Messreihe 8 Anfahren

Abbildung D.2: Aufbereitete Messreihen Anfahren (2) (Klassenindex = 1, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

D.2 Aufbereitete Messreihen der Normalbremsung



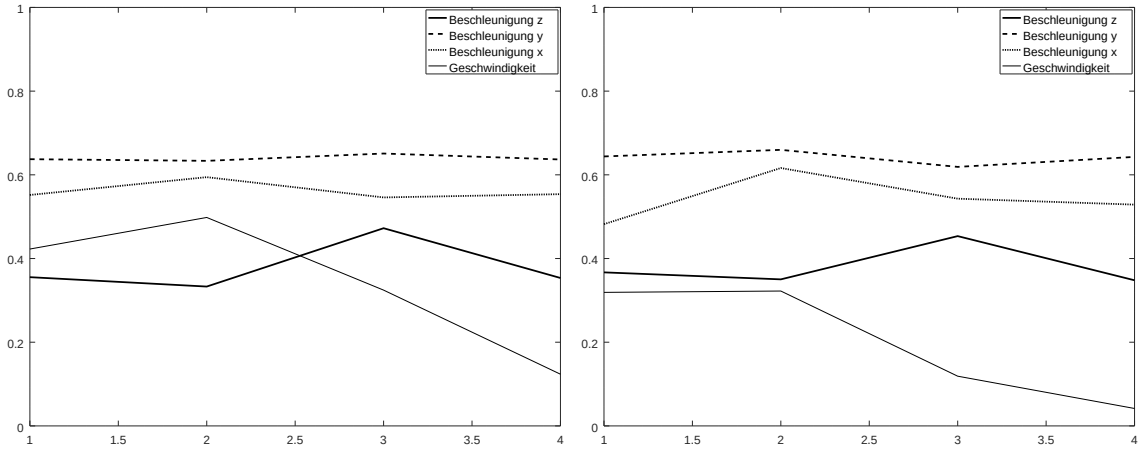
(a) Messreihe 1 Normalbremsung

(b) Messreihe 2 Normalbremsung

(c) Messreihe 3 Normalbremsung

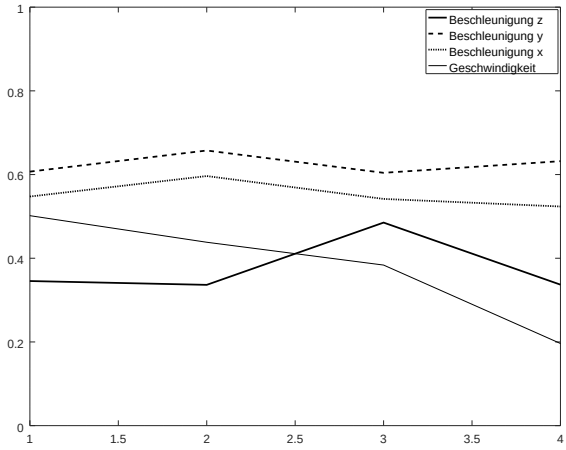
(d) Messreihe 4 Normalbremsung

Abbildung D.3: Aufbereitete Messreihen Normalbremsung (1) (Klassenindex = 2, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

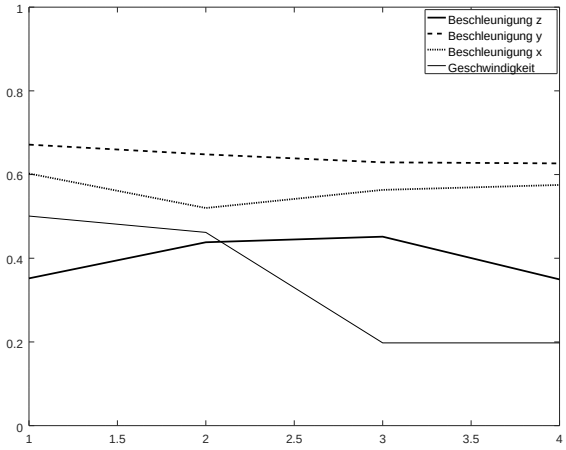


(a) Messreihe 5 Normalbremsung

(b) Messreihe 6 Normalbremsung



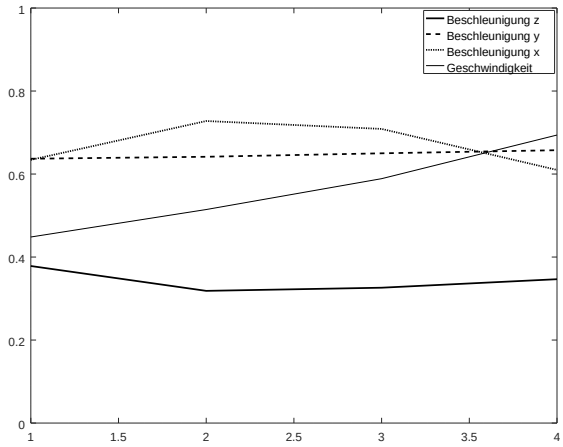
(c) Messreihe 7 Normalbremsung



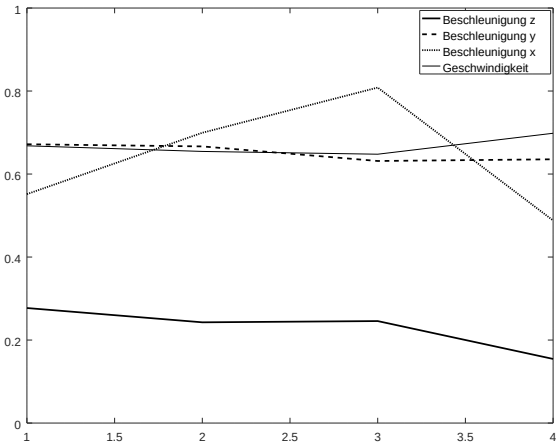
(d) Messreihe 8 Normalbremsung

Abbildung D.4: Aufbereitete Messreihen Normalbremsung (2) (Klassenindex = 2, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

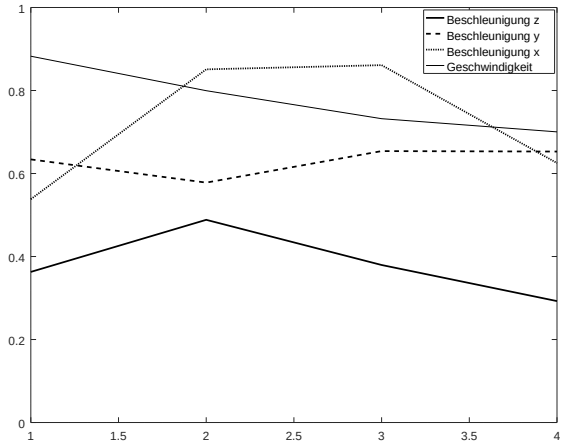
D.3 Aufbereitete Messreihen der Rechtskurve



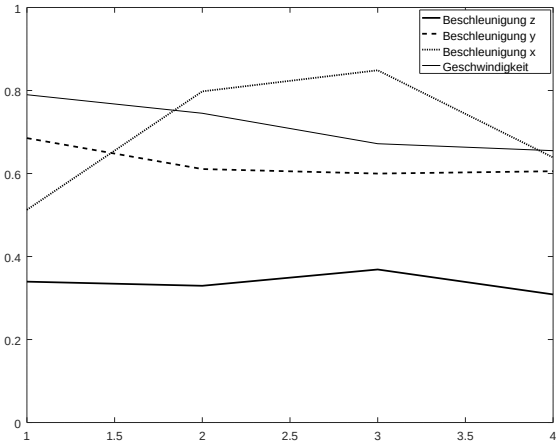
(a) Messreihe 1 Rechtskurve



(b) Messreihe 2 Rechtskurve

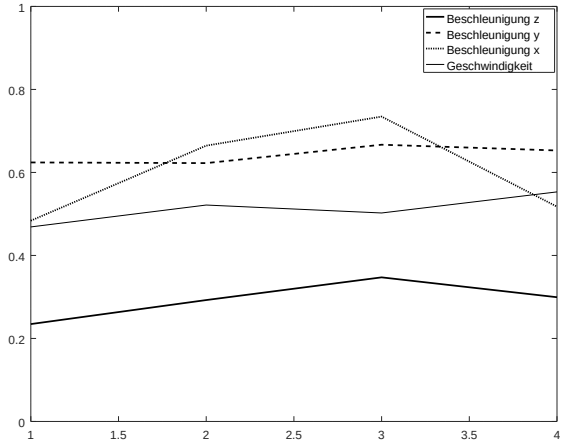


(c) Messreihe 3 Rechtskurve

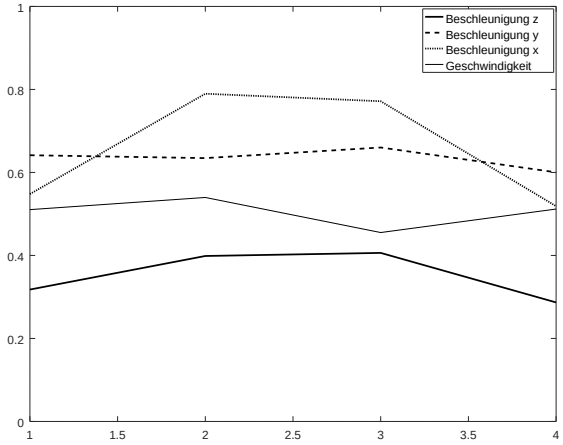


(d) Messreihe 4 Rechtskurve

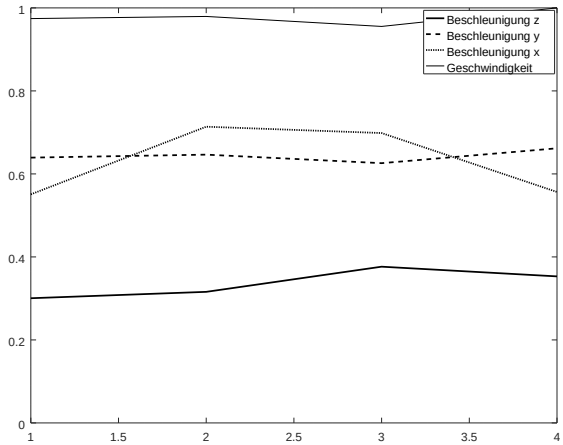
Abbildung D.5: Aufbereitete Messreihen Rechtskurve (1) (Klassenindex = 3, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



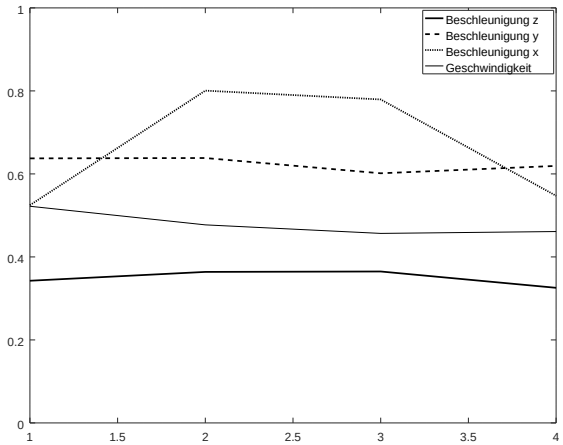
(a) Messreihe 5 Rechtskurve



(b) Messreihe 6 Rechtskurve



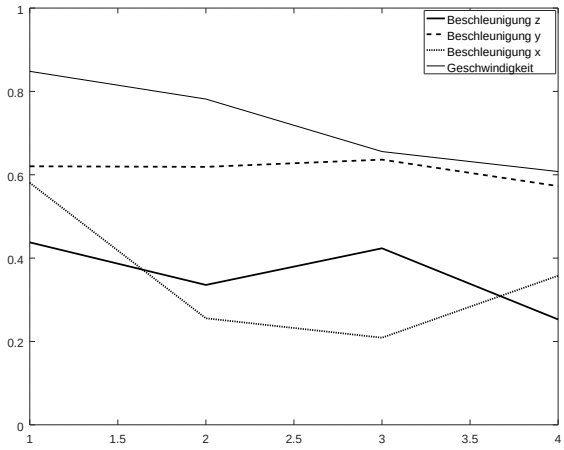
(c) Messreihe 7 Rechtskurve



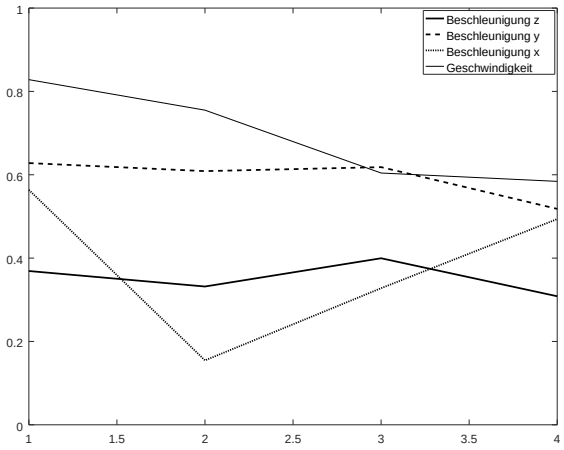
(d) Messreihe 8 Rechtskurve

Abbildung D.6: Aufbereitete Messreihen Rechtskurve (2) (Klassenindex = 3, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

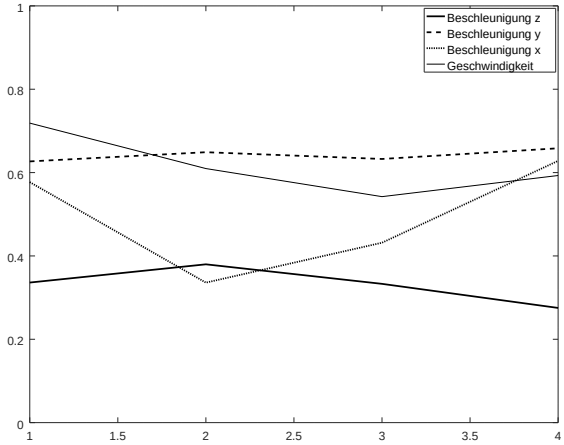
D.4 Aufbereitete Messreihen der Linkskurve



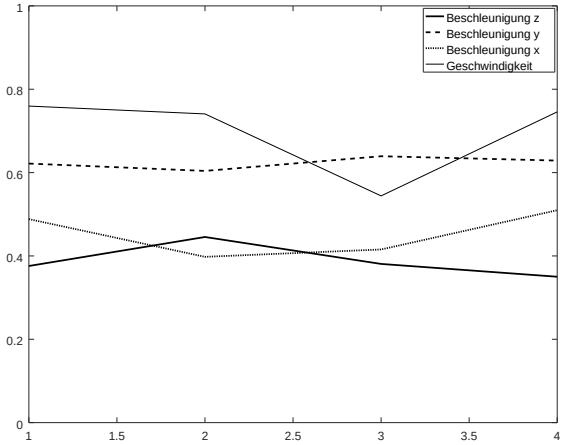
(a) Messreihe 1 Linkskurve



(b) Messreihe 2 Linkskurve

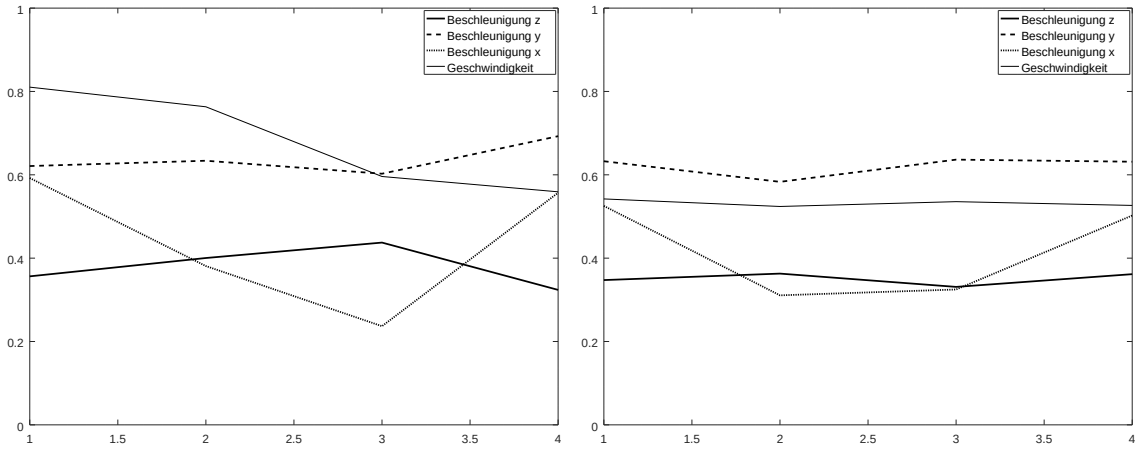


(c) Messreihe 3 Linkskurve



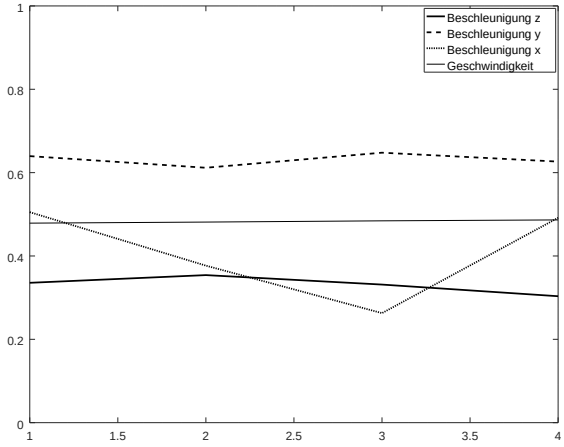
(d) Messreihe 4 Linkskurve

Abbildung D.7: Aufbereitete Messreihen Linkskurve (1) (Klassenindex = 4, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

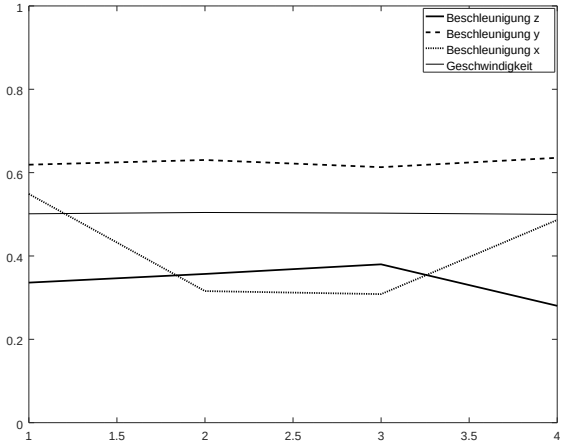


(a) Messreihe 5 Linkskurve

(b) Messreihe 6 Linkskurve



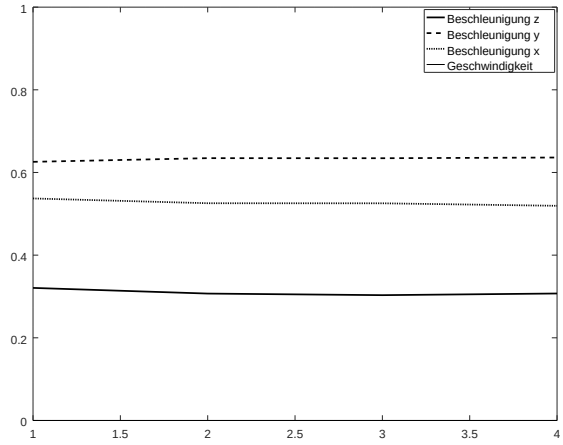
(c) Messreihe 7 Linkskurve



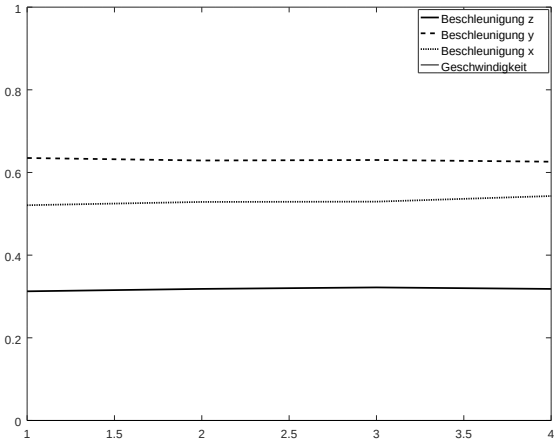
(d) Messreihe 8 Linkskurve

Abbildung D.8: Aufbereitete Messreihen Linkskurve (2) (Klassenindex = 4, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

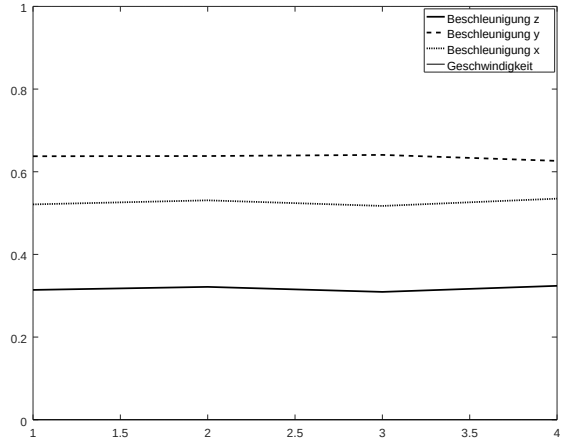
D.5 Aufbereitete Messreihen der Stillstand



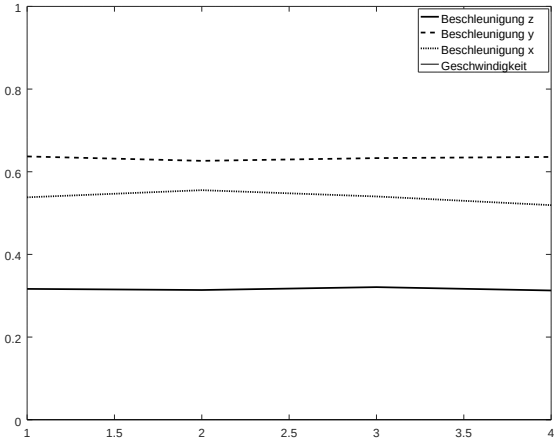
(a) Messreihe 1 Stillstand



(b) Messreihe 2 Stillstand

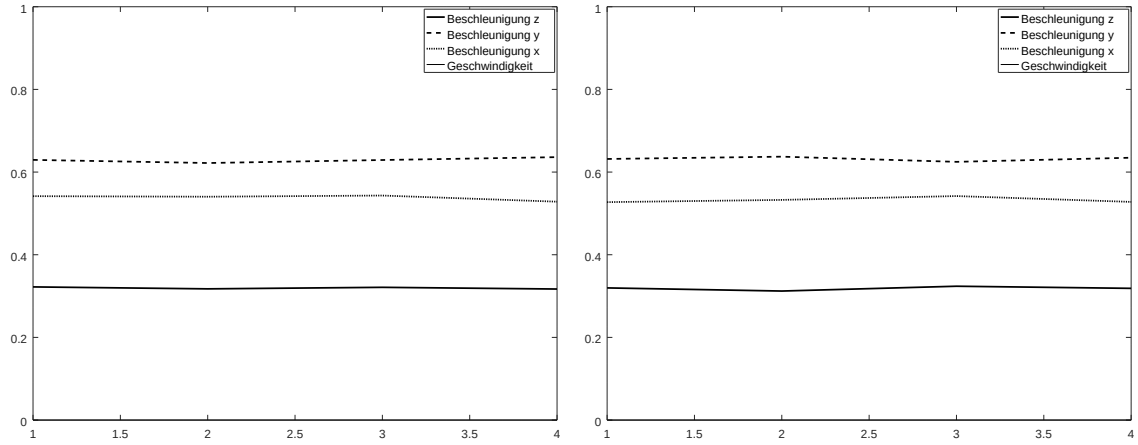


(c) Messreihe 3 Stillstand



(d) Messreihe 4 Stillstand

Abbildung D.9: Aufbereitete Messreihen Stillstand (1) (Klassenindex = 5, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)



(a) Messreihe 5 Stillstand

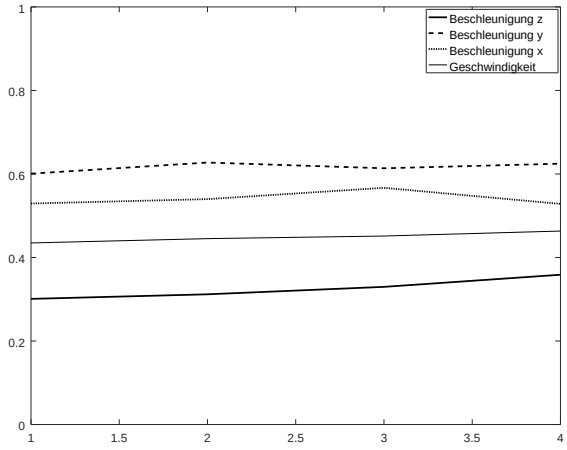
(b) Messreihe 6 Stillstand

(c) Messreihe 7 Stillstand

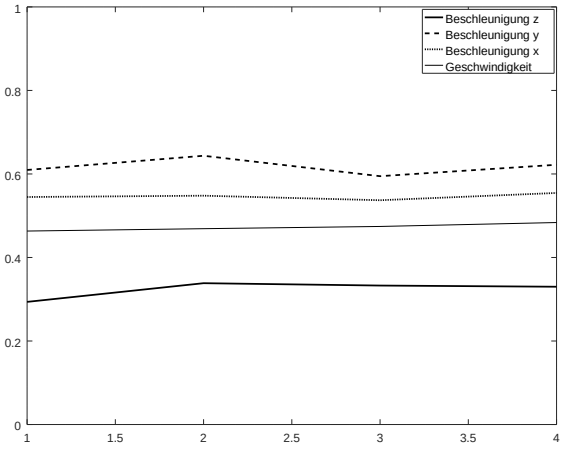
(d) Messreihe 8 Stillstand

Abbildung D.10: Aufbereitete Messreihen Stillstand (2) (Klassenindex = 5, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

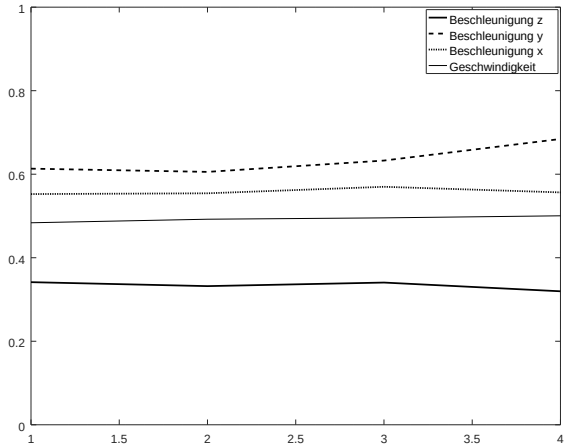
D.6 Aufbereitete Messreihen der Geradeaus



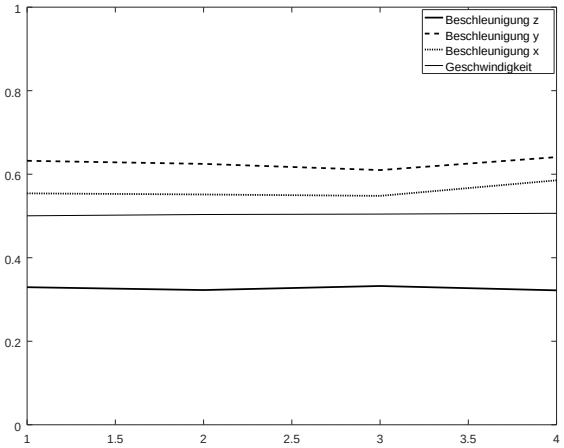
(a) Messreihe 1 Geradeaus



(b) Messreihe 2 Geradeaus

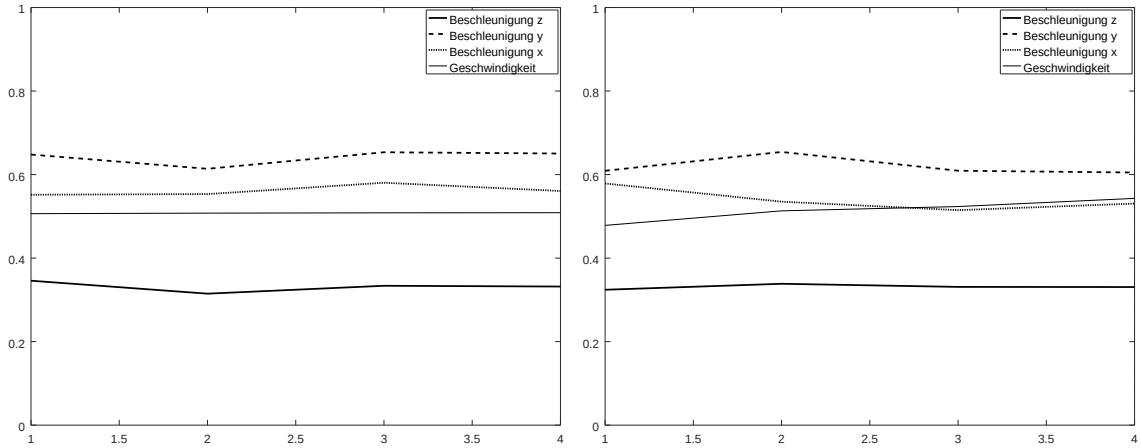


(c) Messreihe 3 Geradeaus



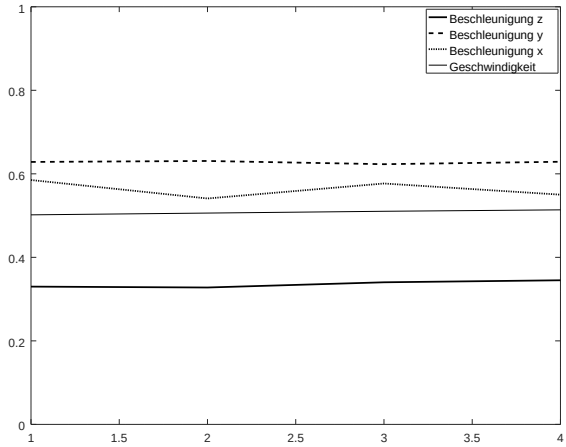
(d) Messreihe 4 Geradeaus

Abbildung D.11: Aufbereitete Messreihen Geradeaus (1) (Klassenindex = 6, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

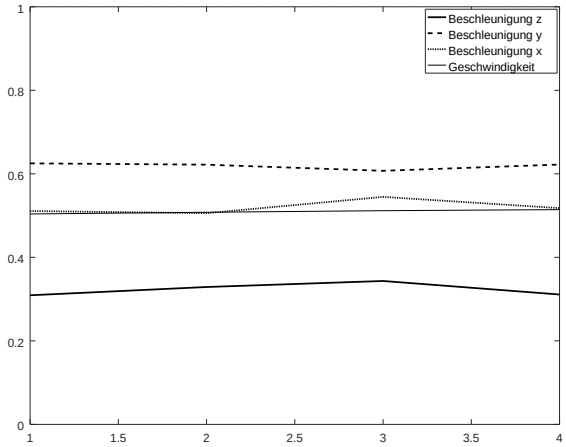


(a) Messreihe 5 Geradeaus

(b) Messreihe 6 Geradeaus



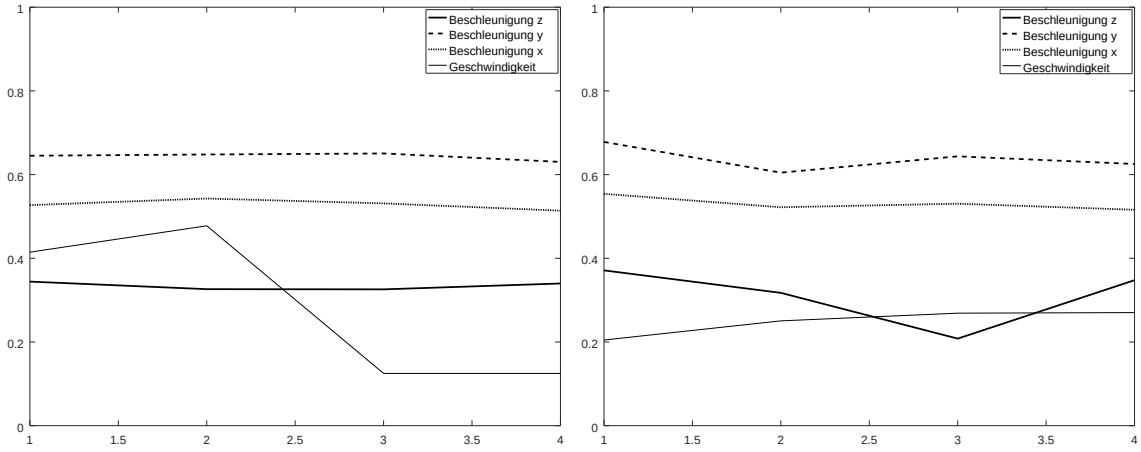
(c) Messreihe 7 Geradeaus



(d) Messreihe 8 Geradeaus

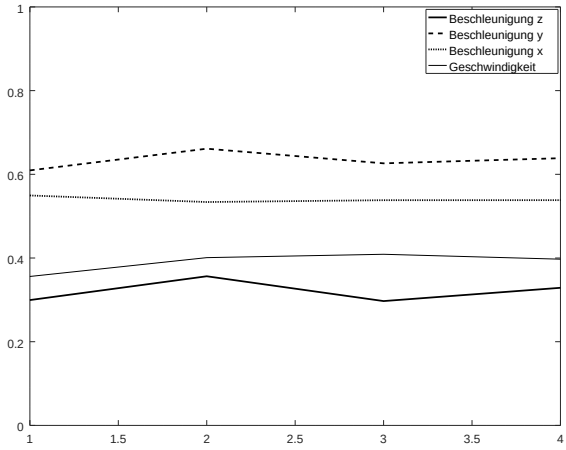
Abbildung D.12: Aufbereitete Messreihen Geradeaus (2) (Klassenindex = 6, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

D.7 Aufbereitete Messreihen der Vollbremsung

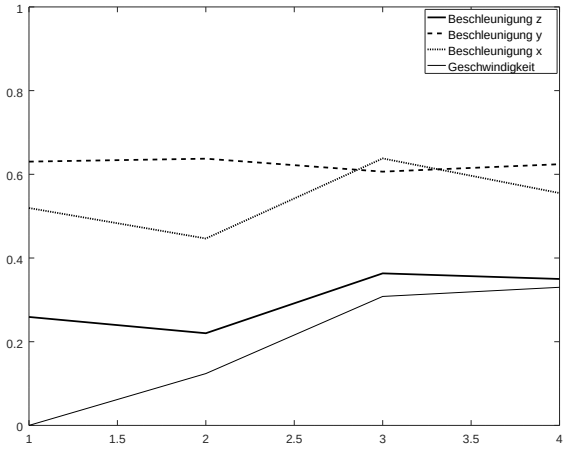


(a) Messreihe 1 Vollbremsung

(b) Messreihe 2 Vollbremsung

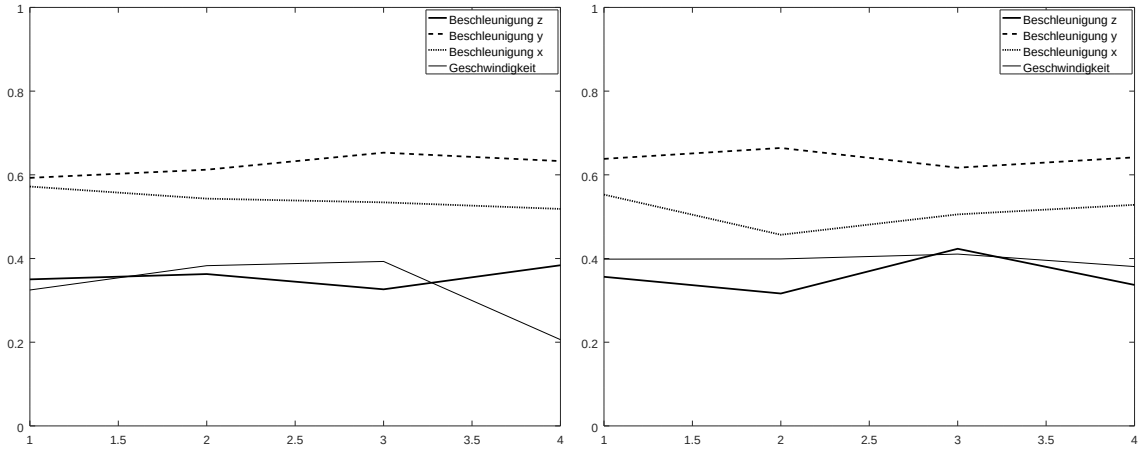


(c) Messreihe 3 Vollbremsung



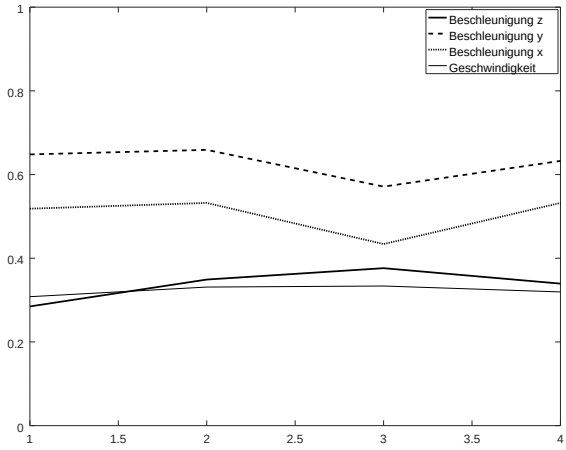
(d) Messreihe 4 Vollbremsung

Abbildung D.13: Aufbereitete Messreihen Vollbremsung (1) (Klassenindex = 7, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

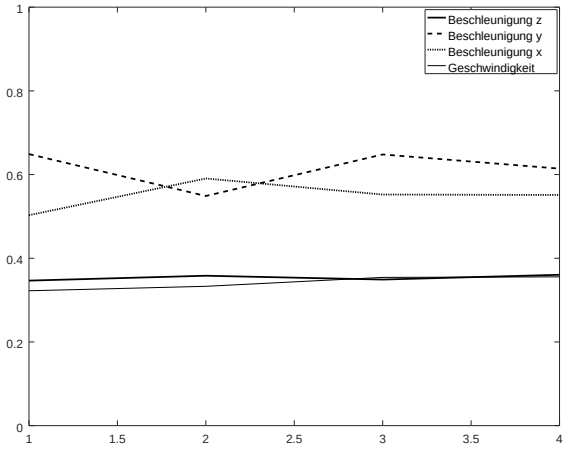


(a) Messreihe 5 Vollbremsung

(b) Messreihe 6 Vollbremsung



(c) Messreihe 7 Vollbremsung



(d) Messreihe 8 Vollbremsung

Abbildung D.14: Aufbereitete Messreihen Vollbremsung (2) (Klassenindex = 7, Beschleunigungen in $\frac{\mathrm{m}}{\mathrm{s}^2}$ und Geschwindigkeiten in $\frac{\mathrm{m}}{\mathrm{s}}$)

Literaturverzeichnis

[1] Frieder Stolzenburg, Olivia Michael, and Oliver Obst. Predictive neural networks. *CoRR*, abs/1802.03308, 2018, url:http://arxiv.org/abs/1802.03308. (zitiert auf Seite xi, 9, 26, 27, 28, 60 und 103)

[2] Statistisches Bundesamt (Destatis). Verkehr aktuell. Technical Report Fachserie 8 Reihe 1.1, 2018. Artikelnummer: 2080110181034. (zitiert auf Seite 1)

[3] Statistisches Bundesamt (Destatis). Verkehrsunfälle. Technical Report Fachserie 8 Reihe 7, 2018. Artikelnummer: 2080700181124. (zitiert auf Seite 1)

[4] Deutscher Verkehrssicherheitsrat. Der Sicherheitsgurt – Lebensretter Nr. 1. Technical Report 15 Schriftenreihe Verkehrssicherheit, 2011. (zitiert auf Seite 1)

[5] Konrad Reif (Hrsg.). *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*. Vieweg+Teubner, 2010, url:https://doi.org/10.1007/978-3-8348-9717-6_1. (zitiert auf Seite 2)

[6] Gary Davis. Relating Severity of Pedestrian Injury to Impact Speed in Vehicle-Pedestrian Crashes: Simple Threshold Model. *Transportation Research Record: Journal of the Transportation Research Board*, 1773:108–113, 2006, doi:10.3141/1773-13, url:https://doi.org/10.3141/1773-13. (zitiert auf Seite 3)

[7] Cosmos Versicherung Aktiengesellschaft. BetterDrive Die Fahranfängerversicherung. Website, 2018. Abgerufen unter https://www.cosmosdirekt.de/betterdrive/?mediacode=dr.03931501_D_Divers_na_Content-Artikel_Vertriebsteaser.Telematik_produktinfos.#produktuebersicht; abgerufen am 24.04.2018. (zitiert auf Seite 3)

[8] Kari Torkkola, Noel Massey, and Chip Wood. Driver inattention detection through intelligent analysis of readily available sensors. In *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pages 326–331. IEEE, Okt 2004, doi:10.1109/ITSC.2004.1398919. (zitiert auf Seite 3, 5, 33 und 36)

[9] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin

Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, 2016. USENIX Association, url:https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi. (zitiert auf Seite 4)

[10] Jin-Hyuk Hong, Ben Margines, and Anind K. Dey. A smartphone-based sensing platform to model aggressive driving behaviors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 4047–4056, New York, NY, USA, 2014. ACM, doi:10.1145/2556288.2557321, url:http://doi.acm.org/10.1145/2556288.2557321. (zitiert auf Seite 5 und 35)

[11] Derick A. Johnson and Mohan M. Trivedi. Driving style recognition using a smartphone as a sensor platform. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1609–1615, Oct 2011, doi:10.1109/ITSC.2011.6083078, issn:2153-0017. (zitiert auf Seite 6 und 35)

[12] Chuang-Wen You, Martha Montes-de Oca, Thomas J. Bao, Nicholas D. Lane, Hong Lu, Giuseppe Cardone, Lorenzo Torresani, and Andrew T. Campbell. Carsafe: A driver safety app that detects dangerous driving behavior using dual-cameras on smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 671–672, New York, NY, USA, 2012. ACM, doi:10.1145/2370216.2370360, url:http://doi.acm.org/10.1145/2370216.2370360. (zitiert auf Seite 6)

[13] Rosolino Vaiana, Teresa Iuele, Vittorio Astarita, Maria Vittoria Caruso, Antonio Tassitani, Claudio Zaffino, and Vincenzo Giofré. Driving behavior and traffic safety: An acceleration-based safety evaluation procedure for smartphones. 8:88–96, 12 2014. (zitiert auf Seite 6 und 35)

[14] Javier E. Meseguer, Carlos Miguel Tavares Calafate, Juan-Carlos Cano, and Pietro Manzoni. Drivingstyles: A smartphone application to assess driver behavior. In *ISCC*, pages 535–540. IEEE Computer Society, 2013, url:http://dblp.uni-trier.de/db/conf/iscc/iscc2013.html#MeseguerCCM13. (zitiert auf Seite 6)

[15] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017, doi:10.1007/s10618-016-0483-9, issn:1573-756X, url:https://doi.org/10.1007/s10618-016-0483-9. (zitiert auf Seite 6)

[16] Herbert Jaeger. Controlling recurrent neural networks by conceptors. *CoRR*, abs/1403.3369, 2014, url:http://arxiv.org/abs/1403.3369. (zitiert auf Seite 6, 29, 30, 31, 51, 52, 53, 54, 55, 57, 61, 64 und 80)

[17] Jiao Bao, Lishen Pei, Mao Ye, and Xuezhuan Zhao. Action recognition based on conceptors of skeleton joint trajectories. 31:11–22, 11 2016. (zitiert auf Seite 6)

[18] Richard Gast, Patrick Faion, Kai Standvoss, Andrea Suckro, Brian Lewis, and Gordon Pipa. Encoding and decoding dynamic sensory signals with recurrent neural networks: An application of conceptors to birdsongs. *bioRxiv*, 2017, doi:10.1101/131052, url:https://www.biorxiv.org/content/early/2017/04/28/131052. (zitiert auf Seite 6)

[19] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3:31, 2009, doi:10.3389/neuro.09.031.2009, issn:1662-5161, url:https://www.frontiersin.org/article/10.3389/neuro.09.031.2009. (zitiert auf Seite 9)

[20] Henning Beck, Sofia Anastasiadou, and Christopher Meyer zu Reckendorf. *Faszinierendes Gehirn*. Springer Spektrum, 2016, doi:10.1007/978-3-662-47092-3. (zitiert auf Seite 9, 10, 11, 12 und 17)

[21] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. Springer Vieweg, 2016, doi:10.1007/978-3-658-13549-2. (zitiert auf Seite 10, 11, 17, 18 und 20)

[22] Rudolf Kruse, Christian Borgelt, Christian Braune, Frank Klawonn, Christian Moewes, and Matthias Steinbrecher. *Computational Intelligence*. Springer Vieweg, 2015, doi:10.1007/978-3-658-10904-2. (zitiert auf Seite 13, 17, 18 und 20)

[23] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. (zitiert auf Seite 13)

[24] Sebastian Dörn. *Programmieren für Ingenieure und Naturwissenschaftler*. Springer Vieweg, 2018, doi:10.1007/978-3-662-54304-7. (zitiert auf Seite 13 und 22)

[25] Peter S. Eriksson, Ekaterina Perfilieva, Thomas Björk-Eriksson, A.M. Alborn, Claes Nordborg, Daniel A. Peterson, and Fred H. Gage. Neurogenesis in the adult human hippocampus. 4:1313–7, 12 1998. (zitiert auf Seite 17)

[26] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982, doi:10.1073/pnas.79.8.2554, issn:0027-8424, url:http://www.pnas.org/content/79/8/2554. (zitiert auf Seite 19 und 20)

[27] Martin Korte. Der Stoff aus dem Erinnerungen sind. *Gehirn & Geist*, (12):40–47, 2017, issn:1618-8519, url:www.spektrum.de/artikel/1508515. (zitiert auf Seite 19)

[28] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990, issn:0364-0213, url:http://crl.ucsd.edu/ elman/Papers/fsit.pdf. (zitiert auf Seite 20, 21 und 23)

[29] Michael I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986. (zitiert auf Seite 21 und 22)

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997, doi:10.1162/neco.1997.9.8.1735, issn:0899-7667, url:http://dx.doi.org/10.1162/neco.1997.9.8.1735. (zitiert auf Seite 23 und 24)

[31] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999, doi:10.1049/cp:19991218, issn:0537-9989. (zitiert auf Seite 24)

[32] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007, doi:10.4249/scholarpedia.2330. revision #186395. (zitiert auf Seite 24, 25 und 26)

[33] Herbert Jaeger. Conceptors: an easy introduction. *CoRR*, abs/1406.2671, 2014, url:http://arxiv.org/abs/1406.2671. (zitiert auf Seite 29, 30 und 54)

[34] Klaus Beuth. *Digitaltechnik*. Vogel, Würzburg, 2003. (zitiert auf Seite 30)

[35] Science Journal. Website, 2018. Abgerufen unter https://makingscience.withgoogle.com; abgerufen am 20.09.2018. (zitiert auf Seite 37)

[36] Physics Toolbox Sensor Suite. Website, 2018. Abgerufen unter https://www.vieyrasoftware.net; abgerufen am 20.09.2018. (zitiert auf Seite 38)

[37] phyphox. Website, 2018. Abgerufen unter https://phyphox.org; abgerufen am 20.09.2018. (zitiert auf Seite 38)

[38] Mark Pedley. Tilt sensing using a three-axis accelerometer. Technical Report AN3461 Rev. 6, Freescale Semiconductor, 2013. (zitiert auf Seite 43)

[39] Tilt measurement using a low-g 3-axis accelerometer. Technical Report AN4509 Rev. 1, STMicroelectronics N.V., 2014. (zitiert auf Seite 43)

[40] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 4.2.0 manual: a high-level interactive language for numerical computations*, 2016, url:http://www.gnu.org/software/octave/doc/interpreter. (zitiert auf Seite 52 und 55)

[41] Herbert Jaeger. A Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the 'Echo State Network' Approach. Technical Report 159, Fraunhofer Institute for Autonomous Intelligent Systems (AIS), October 2002, url:http://minds.jacobs-university.de/pubs. (zitiert auf Seite 52)

[42] Herbert Jaeger. Controlling Recurrent Neural Networks by Conceptors (Matlab code). Website, 2018. Abgerufen unter http://minds.jacobs-university.de/uploads/SW/ConceptorsTechrepMatlab.zip; abgerufen am 27.09.2018. (zitiert auf Seite 61)

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit bisher bei keiner anderen Prüfungsbehörde eingereicht, sie selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Wernigerode, 19. Februar 2020

Trotz des Rückgangs der Verkehrsunfälle in den vergangenen Jahren kommt es regelmäßig zu einer hohen Zahl von Verkehrsopfern. Allein in Deutschland wurden 2017 beinahe 3000 Menschen durch Verkehrsunfälle getötet. Die Ursachen, die zu Verkehrsunfällen führen, sind vielfältig. Einige sind u.a. Selbstüberschätzung, aggressives Fahrverhalten, Müdigkeit oder Ablenkung beim Fahren.

Einen innovativen Ansatz können Fahrerassistenzsysteme bieten, die das Fahrverhaltens analysieren und so Rückschlüsse auf Fahrmanöver ziehen. Sie könnten Ablenkung oder Müdigkeit erkennen und den Fahrer, der evtl. die Lage nicht immer objektiv einschätzen kann, auf ein gestiegenes Risiko hinweisen. Diese Informationen sind auch für Versicherer von Bedeutung, die so eine aktuelle und individuelle Risikobewertung des Fahrers vornehmen können.

Ziel dieser Arbeit ist die Analyse, inwieweit sich Rekurrente Neuronale Netze zur Klassifikation von Fahrmanövern, repräsentiert durch Zeitreihenabschnitte, einsetzen lassen.

