



Masterarbeit

Entwicklung und Evaluation eines auf künstliche Intelligenz gestützten Systems zur Betriebslenkung von Linienbussen im Störfall

31. Januar 2023

Sebastian Knopf
Mat.-Nr.: 68390

Betreuung

M. Sc. Waldemar Titov,
Dirk Weißer (VDV)

Verantwortlicher Hochschullehrer

Prof. Dr.-Ing. Thomas Schlegel

Aufgabenstellung

(DE) Kurzfristige Streckensperrungen oder erhöhtes Verkehrsaufkommen gefährden die Fahrplanstabilität und sorgen dadurch für Frust und Hemmnisse im öffentlichen Personenverkehr. Besonders kleine und mittelständische Verkehrsunternehmen verfügen in den meisten Fällen nicht über eine Betriebsleitstelle, die geeignete Umleitungsfahrwege anweisen und dadurch den Betrieb aufrechterhalten kann. In dieser Masterarbeit soll untersucht werden, inwieweit die Anordnung von Umleitungen durch ein Betriebsleitsystem unterstützt durch künstliche Intelligenz automatisiert werden kann. Nach einer Literaturrecherche zu den Themen Betriebsleitsystem und KI werden passende Verfahren ausgewählt und anhand von Daten aus der Praxis miteinander verglichen. Eine Evaluation prüft anhand ausgewählter Kennzahlen und im Vergleich mit Expertenentscheidungen prüfen Leistungsfähigkeit und Praxistauglichkeit der gewählten Verfahren. Eine kritische Diskussion, eine Zusammenfassung und ein Ausblick runden die Masterarbeit ab.

(EN) Unplanned service interruptions or traffic jams affect the public transport service stability negatively and thus customers feel frustrated and act self-consciously regarded to public transport services. Most smaller and medium-sized traffic companies do not have a control center which may arrange a re-routing and keep the service running. The aim of this work is to find procedures which enable a vehicle control system to arrange re-routing of public transport vehicles automatically based on artificial intelligence (AI). To achieve this, different approaches of AI are selected based on literature research and compared to each other based to datasets from the operation of several bus agencies. An evaluation confirms performance and productivity of the AI procedures found. A critical discussion and a summary show possible commitments in practice and complement the whole work.

Betreuender Hochschullehrer

Prof. Dr.-Ing. Thomas Schlegel

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit mit dem Titel

Entwicklung und Evaluation eines auf künstliche Intelligenz gestützten Systems zur Betriebslenkung von Linienbussen im Störfall

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht zu haben.

Karlsruhe, den 31. Januar 2023

Sebastian Knopf

Inhaltsverzeichnis

Abkürzungsverzeichnis	4
Abbildungsverzeichnis	4
Tabellenverzeichnis	5
1 Einleitung	6
1.1 Motivation.....	7
1.2 Zielsetzung.....	9
1.3 Vorgehensweise.....	9
2 Grundlagen	10
2.1 Definition künstlicher Intelligenz	10
2.2 Überwachtes und unüberwachtes Lernen.....	13
2.3 Bestärkendes Lernen.....	15
2.4 Künstliche Intelligenz und maschinelles Lernen.....	17
2.5 Zusammenfassender Vergleich von ML-Verfahren.....	20
2.6 Algorithmen aus dem Reinforcement Learning	21
2.6.1 Wert- und Strategieapproximation	22
2.6.2 Monte-Carlo- und Temporal-Difference-Methoden.....	23
2.6.3 Einordnung bekannter RL-Algorithmen.....	24
2.7 Grundbegriffe aus dem ÖPNV-Betrieb.....	27
2.7.1 Betriebstag	27
2.7.2 Linie und Linienvariante	28
2.7.3 Umlauf- und Dienstplan.....	28
2.7.4 Betriebsstabilität und dispositive Maßnahme	29
2.7.5 Betriebsstörung	29
2.7.6 Innerbetriebliche und öffentliche Information	30
2.7.7 Rechnergestütztes Betriebsleitsystem	30
2.7.8 Bordrechner	31

3	Theoretische Modellierung und Konzeption	32
3.1	Aufstellung geeigneter Beispielszenarien.....	32
3.2	Auswahl verfügbarer Eingangsdaten.....	34
3.2.1	Betriebliche Daten für Fahrplan und Liniennetz	35
3.2.2	Kartendaten und Routing	36
3.2.3	Störungsmeldungen und Daten zur Verkehrssituation	40
3.3	Auswahl geeigneter RL-Algorithmen	46
3.4	Modellierung der Umwelt zur Simulation	48
3.4.1	Anlehnung an Markov-Entscheidungsprozesse.....	48
3.4.2	Bedeutung von erwartetem und erreichtem Gewinn	53
3.4.3	Approximation der Value-Funktion durch neuronale Netze	54
3.4.4	Trainingsprozess mit neuronalen Netzen im RL.....	59
3.4.5	Ziel- und Strategienetzwerk	61
4	Literaturverzeichnis	62
	Anhang.....	68

Abkürzungsverzeichnis

API	<i>Application Programming Interface</i>
CSV	<i>Comma Separated Values</i>
DQN	<i>Deep Q-Network</i>
GTFS	<i>General Transfer Feed Specification</i>
ITCS	<i>Intermodal Transportation Control System</i>
KI	<i>Künstliche Intelligenz</i>
KNN	<i>Künstliches neuronales Netz</i>
MC	<i>Monte-Carlo-Methode</i>
MEP	<i>Markov-Entscheidungsprozess</i>
ML	<i>Machine Learning, Maschinelles Lernen oder Maschinenlernen</i>
ÖPNV	<i>Öffentlicher Personennahverkehr</i>
OSM	<i>OpenStreetMap</i>
RBL	<i>Rechnergestütztes Betriebsleitsystem</i>
ReLU	<i>Rectified Linear Unit</i>
RL	<i>Reinforcement Learning</i>
TanH	<i>Tangens-Hyperbolicus</i>
TD	<i>Temporal-Difference-Methode</i>
VDV	<i>Verband Deutscher Verkehrsunternehmen</i>
XML	<i>Extensible Markup Language</i>

Abbildungsverzeichnis

Abbildung 1: Graphische Darstellung des k-Nearest-Neighbours Algorithmus	14
Abbildung 2: Graphische Darstellung des Reinforcement Learning.....	16
Abbildung 3: Darstellung einer klassischen und einer unscharfen Menge	18
Abbildung 4: Graphische Darstellung der Überdeckung unscharfer Mengen	19
Abbildung 5: Beispielszenario für Linienbusse im Stadtverkehr	33
Abbildung 6: Beispielszenario für Linienbusse im Regionalverkehr	34
Abbildung 7: Overpass-Abfrage für Straßendaten im Primärnetz	37
Abbildung 8: Overpass-Vorschau für Straßendatendaten im Primärnetz.....	38

Abbildung 9: Ableitung eines für Linienbusse geeigneten Netzes aus OSM-Daten	39
Abbildung 10: Umsetzungsgrad von Smart City Projekten in den 200 größten Städten	41
Abbildung 11: Vordefinierte Meldungen auf einem Bordrechner in einem Linienbus	42
Abbildung 12: Validierung einer systemseitigen Meldung als Flussdiagramm	44
Abbildung 13: Bekannte RL-Algorithmen	46
Abbildung 14: Einordnung der vorgestellten RL-Algorithmen	47
Abbildung 15: Geographische Darstellung von Start- und Terminalzuständen	50
Abbildung 16: Schematische Darstellung der Ein- und Ausgangsdaten im KNN	55
Abbildung 17: Einbindung der Gewichtungsmatrizen in einem KNN	56
Abbildung 18: Schematische Darstellung der Berechnungsschritte innerhalb eines Neurons	57
Abbildung 19: Funktionsgraphen der Sigmoid- und TanH-Funktion	57
Abbildung 20: Funktionsgraphen der ReLU- und Leaky ReLU-Funktion	58
Abbildung 21: Pseudocode des Q-Learning	60
Abbildung 22: Pseudocode des SARSA-Algorithmus	61

Tabellenverzeichnis

Tabelle 1: Straßenkategorien und deren Schlüssel in OSM-Daten	37
--	----

1 Einleitung

Häufig gilt der öffentliche Personenverkehr noch immer als unzuverlässig, wenig flexibel und zu kompliziert. Unvorhergesehene Ereignisse und Streckensperrungen beeinträchtigen die Fahrplanstabilität und bestätigen damit unnötigerweise das Bild des unzuverlässigen, unflexiblen öffentlichen Personenverkehrs. Auswertungen nach dem Ende des im Sommer 2022 von der Deutschen Bundesregierung initiierten 9-Euro-Tickets zeigen, dass neben dem Fahrpreis besonders auch die Angebotsqualität sowohl bezogen auf die Verfügbarkeit als auch auf die vorhandene Fahrgastinformationen erheblichen Einfluss auf die Verkehrsmittelverlagerung haben (vgl. Krämer 2022).

Durch die Marktöffnung im öffentlichen Personenverkehr sind insbesondere in Busnetzen immer häufiger mittelständische Verkehrsunternehmen mit der Betriebsdurchführung betraut. Bedingt durch den hohen und stetig wachsenden Kostendruck leisten sich kleine und mittelständische Verkehrsunternehmen selten eine Betriebsleitstelle, sodass insbesondere bei kurzfristigen Störungen nicht zeitgerecht reagiert und beispielsweise Umleitungen von zentraler Stelle angeordnet werden können. In den meisten Fällen kommt der Betrieb im Störfall kurzzeitig zum Erliegen. Selbst nach Ende der Störung kann es mitunter Stunden dauern, bis alle Fahrzeuge bedingt durch fehlende Wendezeiten in den Wagenumläufen die geplanten Fahrten wieder planmäßig durchführen können. Eine zeitgemäße Information der Fahrgäste unterbleibt in den meisten Fällen komplett, was wiederum zu Frustration und Abneigung auf Seiten der Fahrgäste führt.

Im Rahmen dieser Masterarbeit soll untersucht werden, inwieweit sich Daten aus der Planung als auch aus dem Betrieb als Entscheidungsgrundlage für dispositive Maßnahmen eignen und wie ein Betriebsleitsystem unterstützt durch künstliche Intelligenz damit selbstständig in die Lage versetzt werden kann, den Betrieb auch im Störfall möglichst aufrecht zu erhalten.

1.1 Motivation

Aus technischer Sicht betrachtet sind viele Lösungsansätze für Teilprobleme bereits verfügbar. Besonders in größeren Verkehrsunternehmen, die im Regelfall dann auch über eine über die gesamte Betriebszeit besetzte Betriebsleitstelle verfügen, sind Lösungen zur Unterstützung des Leitstellenpersonals im Störfall vielfach auch implementiert.

Bei kleineren und mittelständischen Verkehrsunternehmen ohne Betriebsleitstelle werden wertvolle Daten aus den Bordrechnern der Fahrzeuge hingegen nicht genutzt, obwohl genau diese eine gute Basis für zukünftige Entscheidungen sein könnten. Diese Entscheidungen könnten sowohl im Umfeld der Betriebslenkung als auch bei der Fahrgastinformation eine Basis zur stabilen Aufrechterhaltung des Betriebes sein. Was dies konkret bedeuten könnte, soll an einem selbst erlebten Beispiel aus dem Alltag verdeutlicht werden:

An einem Samstagvormittag möchte ich öffentliche Verkehrsmittel zu einer großen Veranstaltung nutzen. Erfreulicherweise fährt der kombinierte Stadt- und Regionalverkehr auch am Wochenende im 30-Minuten-Takt. Am Busbahnhof angekommen, sehe ich wie sich eine Busfahrerin etwas aufgeregt mit einem Kollegen unterhält. Beim Einstieg in den Bus erzählt mir die motivierte Busfahrerin, dass meine Zielhaltestelle wegen dem hohen Verkehrsaufkommen nicht mehr angefahren wird und sie mich stattdessen an einer alternativen Haltestelle absetzen wird, die zu Fuß aber dieselbe Entfernung bedeutet. Beim Ausstieg zeigt sie mir noch die Haltestelle, an der ich in den Bus zur Rückfahrt einsteigen solle.

Stunden später finde ich mich pünktlich zur Rückfahrt an der besagten Haltestelle ein, doch auch 20 Minuten nach der planmäßigen Abfahrtszeit taucht kein Bus auf. Einschlägige Informationsmedien und die lokale Fahrplanauskunft helfen mir kein Stück weiter, da sie weder Echtzeitdaten noch eine vernünftige Information über eine eventuelle Umleitung enthalten. Erst 30 Minuten später taucht ein Kleinbus auf, der dann noch einen stark von der Fahrplanauskunft abweichenden Weg zurück zum Busbahnhof fährt und mich schließlich mit einer Verspätung von rund 40 Minuten an den Busbahnhof zurückbringt.

Das hier beschriebene Problem gehört oftmals zum Alltag einer jeden Person, die den öffentlichen Personenverkehr täglich nutzt.

Was ist hier passiert? Die folgende stichpunktartige Analyse soll eine wahrscheinliche Antwort auf diese Frage liefern.

- Ausschlaggebend für die Umleitung war das starke Verkehrsaufkommen an einer Stelle im Verlauf der Buslinie.
- Eine Einhaltung des regulären Linienweges hätte dafür gesorgt, dass sich je Folgefahrt die Verspätung mit einem Delta von +10 Minuten erhöht hätte, da Wendezeiten in diesem Ausmaß im geplanten Wagenumlauf nicht enthalten sind.
- Da es keine Betriebsleitstelle, die korrigierend von zentraler Stelle eingreifen kann und auch kein Konzept für einen solchen Störfall gibt, entschied das Fahrpersonal vor Ort eigenständig, dass die Verspätung durch eigenständige Änderung des Linienweges vermieden wird. Zwar erfolgte eine Absprache und der den beiden aktuell diensthabenden Personen, ob diese aber auch an die Ablösung weitergegeben wurde, bleibt fraglich. Auch andere Fahrzeuge, die an dieser Absprache nicht beteiligt waren, wurden nicht informiert.
- Durch das eigenständige Entscheiden des Fahrpersonals kamen unterschiedliche Umleitungswege zur Anwendung, die jedoch weder einheitlich noch für Fahrgäste kommuniziert wurden. Die Folge davon war, dass die Fahrpersonale jeweils ihre eigenen, nicht einheitlichen Umleitungswege bedienten, die jedoch nicht aufeinander abgestimmt waren.
- Das Fehlen einer zentralen Anweisung sorgte für den willkürlichen Ausfall eines Linienabschnittes oder gar einer ganzen Fahrt. Die ausbleibende Fahrgastinformation machte das Chaos perfekt.

Das Wissen über technische Möglichkeiten und das eigene Interesse, Lösungen zur Behebung dieser längst bekannten Alltagsprobleme bei der Nutzung öffentlicher Verkehrsmittel zu liefern, bilden die Motivation zu dieser Masterarbeit.

1.2 Zielsetzung

Im Rahmen dieser Masterarbeit sollen Möglichkeiten erörtert werden, die ein Betriebsleitsystem in die Lage versetzen, selbstständig Umleitungen im Störfall anzuordnen. Dabei sollen möglichst Eingangsdaten verwendet werden, die ohnehin vorhanden sind oder mit wenig Aufwand erzeugt werden können. Ziel ist ein funktionaler Prototyp, welcher nach Kenntnis über eine Störung selbstständig eine möglichst optimale Umleitung der betroffenen Fahrzeuge anordnet und darüber hinaus auch für die Fahrgastinformation eingesetzt werden kann. Im Kern sollen dabei folgende Forschungsfragen beantwortet werden:

- Welche Daten müssen in welcher Qualität verfügbar sein, um diese zum Training einer künstlichen Intelligenz verwenden zu können?
- Welche Strategien eignen sich für den Einsatz zur automatischen Anordnung einer Umleitung von Linienbussen in einem Betriebsleitsystem?
- Bietet ein solches auf künstliche Intelligenz gestütztes Betriebsleitsystem Potenzial für einen zeitnahen Einsatz in der Praxis?

1.3 Vorgehensweise

Für das grundlegende Verständnis der Arbeit werden in **Kapitel 2** zunächst Grundlagen der künstlichen Intelligenz ermittelt. Darüber hinaus werden Begriffe aus dem täglichen Betrieb eines Verkehrsunternehmens definiert, die für das Verständnis der Arbeit wichtig sind.

Die folgenden **Kapitel 3** und **4** beschäftigen sich jeweils mit der Auswahl zweier zum Vergleich geeigneter ML-Algorithmen und passenden Anwendungsbeispielen dazu und der Implementierung des Funktionsprototypen.

Eine Evaluation in **Kapitel 5** basierend auf qualitativen Daten stellt die beiden ML-Algorithmen auf den Prüfstand und zeigt, ob diese überhaupt geeignet sind und welches Optimierungspotenzial sie bieten. Meinungen von Fachleuten verschiedener Verkehrsunternehmen fließen darüber hinaus als subjektive Einschätzung in die Evaluation ein und schärfen gewonnene Erkenntnisse bezogen auf die Praxistauglichkeit.

In **Kapitel 6** werden die Ergebnisse der Masterarbeit zusammengefasst und zur kritischen Diskussion gestellt.

2 Grundlagen

In diesem Kapitel sollen zunächst theoretische und technische Grundlagen der künstlichen Intelligenz (KI) erörtert und zusammengefasst werden. Neben den wichtigsten Grundlagen zu KI und Machine Learning sind zum Verständnis der Arbeit darüber hinaus auch Begriffe und Abläufe aus dem Betrieb des öffentlichen Personennahverkehrs (ÖPNV) nötig. Im Zweiten Teil des Kapitels werden bestehende Lösungen bei verschiedenen Verkehrsunternehmen beleuchtet. Der Blick geht hierbei bewusst über den Horizont der Linienbusse hinaus, um eventuell gesammelte Erfahrungen aus verwandten Bereichen des Verkehrs und der Logistik transferieren zu können.

2.1 Definition künstlicher Intelligenz

Der Begriff der künstlichen Intelligenz spielt in dieser Masterarbeit eine Schlüsselrolle. Sie soll als Entscheidungsträger in einem Betriebsleitsystem eingesetzt werden. Doch wann genau agiert ein System *intelligent*? Nachfolgend werden einige Definitionsansätze beleuchtet, welche diese Frage beantworten und eine Arbeitsdefinition für künstliche Intelligenz im Kontext der Betriebslenkung im öffentlichen Personenverkehr liefern sollen. Grundsätzlich problematisch hierbei ist, dass der Begriff der Intelligenz selbst nicht definiert ist. So existiert keine einheitliche Definition von Intelligenz, welche als Referenz zur Beurteilung eines Systems dienen kann (vgl. Rimscha 2008, S. 105; Mainzer 2019, S. 2).

Die wohl bekannteste und gleichwohl älteste Definition für den Begriff der künstlichen Intelligenz stammt von Turing aus den 1950er Jahren. Dem sogenannten Turing-Test zur Folge ist ein System dann als intelligent anzusehen, wenn eine beobachtende Person nicht mehr in der Lage ist zu unterscheiden, ob sie mit einer Maschine oder einer natürlichen Person interagiert (vgl. Mainzer 2019, S. 10). Diese Aussage wurde entsprechend oft auch hinterfragt. Bekannt wurde die Kritik von Ada Lovelace, die behauptete, eine Maschine könne ausschließlich dem Zweck dienen, für den sie letztendlich programmiert wurde (vgl. Liggieri und Müller 2019, S. 304). Noch eindrucksvoller wird die Kritik am Beispiel des „Chinesischen Zimmers“ nach Searle (1980). Hierbei handelt es sich um ein gedankliches Experiment mit Ursprung in der Psychologie, bei dem eine Person in einem Raum steht und Fragen auf Chinesisch gestellt bekommt. Als einziges Hilfsmittel verfügt die Person über ein Regelbuch, mit dessen Hilfe sie die Fragen ebenfalls auf Chinesisch beantworten muss. Eine außenstehende Person könnte zu

dem Schluss kommen, dass die Person im Raum die Sprache Chinesisch beherrscht, obwohl sie in Wahrheit nur einen fest definierten Regelsatz abarbeitet und überhaupt nicht über Kenntnisse der chinesischen Sprache verfügt (vgl. Liggieri und Müller 2019, S. 304). Diese beiden Gegenargumente zeigen deutlich, dass die fehlende äußere Unterscheidbarkeit zwischen einem Menschen und einer Maschine noch nicht zwangsläufig eine Interaktion mit einem scheinbar intelligenten Individuum von bedeuten. Im letztgenannten Beispiel wäre dann Intelligenz im Spiel, wenn die Person innerhalb des Raumes sich tatsächlich Kenntnisse der chinesischen Sprache angeeignet hätte.

Einen weiteren, deutlich konkreteren Definitionsansatz liefert Mainzer (2019). Er betrachtet Intelligenz nicht als absolute, binäre Eigenschaft, die durch ein Individuum entweder erfüllt oder nicht erfüllt sein kann, sondern relativiert den Begriff. Statt einem absoluten Vorhandensein oder Nichtvorhandensein der Eigenschaft Intelligenz, wird diese in Graden angegeben und bereits nicht mehr auf beliebige Individuen, sondern schon auf „[Computer]Systeme“ (Mainzer 2019, S. 3) bezogen. Demzufolge hängt der Intelligenzgrad eines Systems „vom Grad der Selbstständigkeit, vom Grad der Komplexität des Problems und dem Grad der Effizienz der Problemlösung“ (Mainzer 2019, S. 3) ab. Diese Definition eröffnet in Bezug auf ein Betriebsleitsystem neue Möglichkeiten. So können Selbstständigkeit, Komplexität des Problems und Effizienz gemessen und in alternativen Systemen miteinander verglichen werden.

Noch kürzer fasst sich Rimscha (2008) bei seiner Definition. Diese besagt sinngemäß, dass ein System dann als intelligent angesehen werden kann, wenn selbstständig Probleme lösen kann, ohne vorher von einem Menschen gesagt zu bekommen, wie genau das Problem gelöst werden soll (vgl. Rimscha 2008, S. 105). Dieser Auffassung ähnelt auch die Definition von Nahrstedt (2012). Ein System im Kontext der künstlichen Intelligenz ist demnach als sogenanntes Expertensystem anzusehen, wenn die gelieferten Lösungen „von der Qualität her von denen eines menschlichen Experten nicht zu unterscheiden sind“ (Nahrstedt 2012, S. 246).

Der Begriff des Expertensystems taucht auch in der historischen Forschung zur künstlichen Intelligenz auf. Während bereits im Barockzeitalter versucht wird, Gedankengut und erlerntes Wissen allein auf Rechnungen zurückzuführen, wächst der Wunsch nach einer Universalsprache. Diese *Lingua Universalis* soll es unabhängig vom Inhalt möglich machen, Wissen in einem mathematischen Kalkül auszudrücken (vgl. Mainzer 2019, S. 8). Die Ergebnisse dieser Forschungen waren jedoch bis Ende der 1960er Jahre kaum praktisch einsetzbar, weshalb sich in den 1970er Jahren der Begriff des Expertensystems etablierte. Hierunter versteht sich die Abgrenzung vom „allgemeinen Problemlöser“ (Mainzer 2019, S. 11) hin zu Expertenwissen in einem in sich geschlossenen und überschaubaren Bereich. Ein Expertensystem verfügt damit über begrenztes Wissen zu einem spezifischen Sachgebiet und kann mit diesem Wissen

entsprechend automatisch Schlussfolgerungen ziehen. Erst diese Eingrenzung ermöglicht erste praxistaugliche Anwendungen basierend auf künstlicher Intelligenz (vgl. Mainzer 2019, S. 12).

Die Kernaussagen dieser Definitionen lassen sich zu einer Arbeitsdefinition für den Begriff des intelligenten Betriebsleitsystems zusammenfassen, die eine Bewertung des Funktionsprototypen aus Kapitel 3 ermöglichen und damit die Überprüfbarkeit sicherstellt.

Ein intelligentes Betriebsleitsystem ist demnach ein System, dass selbstständig in der Lage ist, komplexe Probleme effizient zu lösen, ohne dabei ausschließlich vordefinierte Ergebnisse aus einer endlichen Menge auszuwählen, wobei die gelieferten Ergebnisse hinsichtlich ihrer Qualität nicht von denen eines Menschen unterscheidbar sein dürfen.

Die folgenden Stichpunkte beschreiben beispielhaft, wie ein solches intelligentes Betriebsleitsystem in der Praxis ausgeprägt sein könnte:

- Die Menge der Ergebnisse ist nicht von vorneherein endlich, sondern richtet sich nach messbaren, relevanten Einflussgrößen und bildet dabei auch Schnittmengen von möglichen Kombinationen von Fahrwegen (Eigenständige Lösungsfindung)
- Die Erkennung einer Störungssituation soll vom Betriebsleitsystem automatisch basierend auf bekanntem Wissen erfolgen (Selbstständigkeit)
- Die erwartete Lösung bezieht sich auf die Anordnung von Umleitungen, nicht jedoch auf weitere Dispositive Maßnahmen wie beispielsweise Kurzwenden. Umleitungen sollen hingegen von nahezu beliebigem Umfang sein können (Komplexität)
- Die Lösung soll im Optimalfall schneller als von einem Menschen, zumindest aber in angemessen kurzer Zeit geliefert werden (Effizienz)

2.2 Überwachtes und unüberwachtes Lernen

Insgesamt wird im Machine Learning (ML) zwischen dem *überwachten Lernen* und dem *unüberwachten Lernen* unterschieden. Beim überwachten Lernen werden in *Trainingsdaten* enthaltene Eigenschaften, sogenannte *Features*, extrahiert und einem Ergebnis, dem sogenannten *Label*, zugeordnet. Um zu verhindern, dass das Modell ausschließlich mit den Trainingsdaten funktioniert, wird das Modell nach Abschluss des Trainings auf einen *Testdatensatz* angewandt. Auf diesem Weg kann die Performance ermittelt werden, welche das Modell mit unbekannten Daten erreicht. Die Überanpassung des Modells an die Trainingsdaten wird auch als *Overfitting* bezeichnet (vgl. Niebler 2018, S. 11).

Das unüberwachte Lernen verarbeitet hingegen grundsätzlich unbekannte Daten mit dem Ziel, bislang unbekannte Zusammenhänge innerhalb dieser Daten zu finden oder Inhalte nach verschiedenen Merkmalen zu ordnen. Ein nachgelagerter Test mit Testdaten erfolgt nicht. (vgl. Niebler 2018, S. 11).

Das konstruierte Beispiel der Haltestellensuche kann anhand dieser Definitionen nicht exakt eingeordnet werden. Tendenziell trifft die Beschreibung des überwachten Lernens eher zu. Die Nutzung des Systems kann dabei als fortlaufendes Training betrachtet werden. Die Trigramme der Eingabezeichenfolge werden dem jeweils von den Nutzenden tatsächlich gewählten Suchvorschlag zugeordnet. Die Kombination aus Trigrammen entspricht dabei den Features, der tatsächlich gewählte Suchvorschlag dem Label. Das typischerweise beim überwachten Lernen folgende Testing entfällt an dieser Stelle allerdings. Die verarbeiteten Daten werden direkt gespeichert und beim nächsten Suchvorgang genutzt. Eine Überanpassung ist dann möglich, wenn die Trigramme der Eingabezeichenfolge bereits aus anderen Eingaben enthalten und dadurch einem anderen Suchvorschlag zugeordnet sind als dem eigentlich gewünschten. Entsprechend könnte dem Beispiel auch der k-Nearest-Neighbours-Algorithmus zu Grunde liegen, welcher dem unüberwachten Lernen zuzuordnen ist (vgl. Niebler 2018, S. 11). Der k-Nearest-Neighbours-Algorithmus ordnet Daten dabei anhand ihrer Features in Dimensionen ein und sucht entsprechend die k Nachbarn, deren Features am nächsten am Ausgangsdatensatz liegen. Anhand dieser k Nachbarn wird dann versucht, eine Aussage über den Ausgangsdatensatz zu treffen.

Die folgende Abbildung verdeutlicht die Funktionsweise des k-Nearest-Neighbours-Algorithmus anhand einem Datensatz über den eine Aussage basierend auf den nächsten k Datensätzen aus zwei verschiedenen Klassen getroffen werden soll:

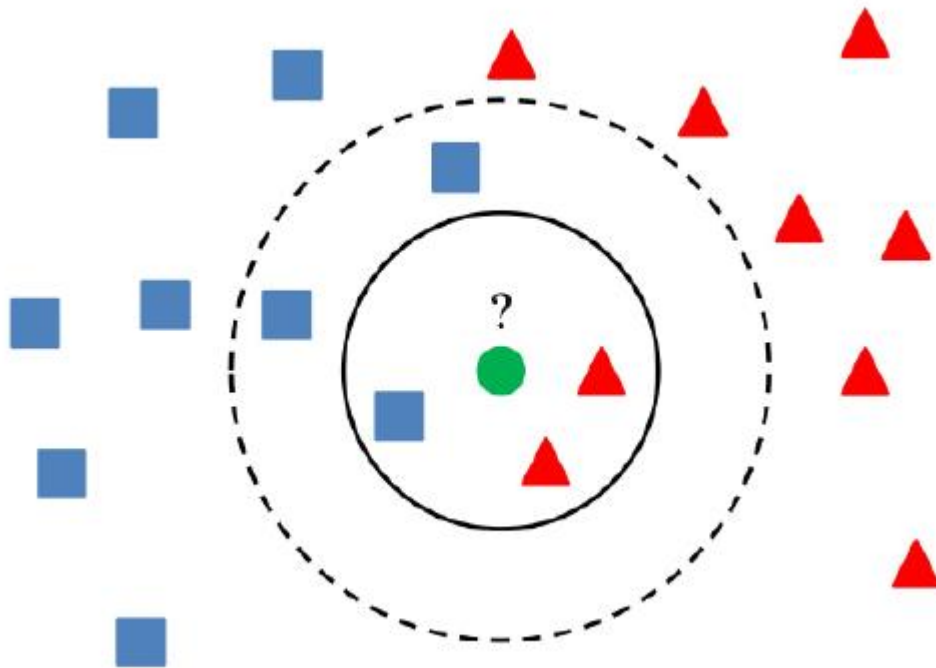


Abbildung 1: Graphische Darstellung des k-Nearest-Neighbours Algorithmus (Quelle: Alaliyat 2022, S. 38)

In grün dargestellt ist der Datensatz, über den eine Aussage getroffen werden soll. In rot und blau respektive als Dreieck und als Quadrat dargestellt, sind die bereits klassifizierten Datensätze. Der k-Nearest-Neighbours-Algorithmus ermittelt nun die k Datensätze, die am nächsten liegen und verwendet diese als Grundlage für eine Aussage über den neuen Datensatz (vgl. Alaliyat 2022, S. 38). Je größer k ist, desto größer ist im Regelfall auch der Suchraum. Am Beispiel der Haltestellensuche käme eine Zählung der Trigramme der Eingabezeichenfolge mit anschließendem Abgleich zwischen bereits einmal gesuchten Trigrammen und anschließender Suche mittels k-Nearest-Neighbours-Algorithmus am nächsten.

2.3 Bestärkendes Lernen

Im vorhergehenden Kapitel wurde Machine Learning bereits in beiden verbreiteten Verfahrensarten des überwachten und unüberwachten Lernens unterteilt. In diesem Kapitel wird entkoppelt vom Beispiel der Haltestellensuche das sogenannte *bestärkende Lernen* betrachtet.

Üblicherweise wird das bestärkende Lernen auch als Reinforcement Learning (RL) bezeichnet. Im Gegensatz zum überwachten und unüberwachten Lernen benötigt RL keine Datenbasis mit entsprechenden Features und Labels (vgl. Schmitz 2017, S. 12). RL liefert als Ergebnis eine Folge von *Aktionen*, die beim Eintreten eines gewissen Falles optimalerweise durchgeführt werden sollen. Das Training eines sogenannten *Agenten* erfolgt nach einem „Trial-and-Error“ Verfahren in einer Simulation des späteren Einsatzgebietes, wobei eine virtuelle *Belohnung* zur Bewertung einer Aktion dient. Ziel des RL ist es, in einer *Umgebung* ausgehend von einem gewissen *Zustand* durch eine bestimmte Strategie bestehend aus geeigneten Aktionen ein Maximum an Belohnung zu erhalten (vgl. Chan et al. 2022, S. 47). Die Verfahrensweise von RL ähnelt dem Lernprozess bestimmter Lebewesen sehr. Ein bekanntes Beispiel dazu ist ein Baby, dass Laufen lernt (vgl. Schmitz 2017, S. 12). Schafft es das Baby, eine bestimmte Strecke zu laufen, wird es gelobt. Es kommt zu einem Erfolgserlebnis, welches in diesem Fall die positive Belohnung repräsentiert. Fällt das Baby hingegen hin, führt dies zum Scheitern und damit zu einer negativen Belohnung. Das Beispiel lässt sich ähnlich auch auf ein autonomes Fahrzeug übertragen, welches ein bestimmtes Ziel erreichen soll. Das Fahrzeug kann abbiegen oder geradeaus fahren. Erreicht das Fahrzeug das geforderte Ziel ohne einen Unfall, folgt eine positive Belohnung. Kommt es hingegen zu einem Zusammenstoß mit einem anderen Fahrzeug, folgt daraus eine negative Belohnung (vgl. Chan et al. 2022, S. 47). Generell werden im RL eine *Policy*- und eine *Valuefunktion* definiert. Während die Policyfunktion eine Wahrscheinlichkeit angibt, mit der eine bestimmte Aktion ausgeführt wird, definiert die Valuefunktion, ob ausgehend von einem bestimmten Zustand beim Ausführen einer Aktion eine negative oder positive Belohnung zu erwarten ist (vgl. Schmitz 2017, 12 ff.).

Die folgende Grafik zeigt das Zusammenspiel zwischen Agenten und Umgebung im Reinforcement Learning:

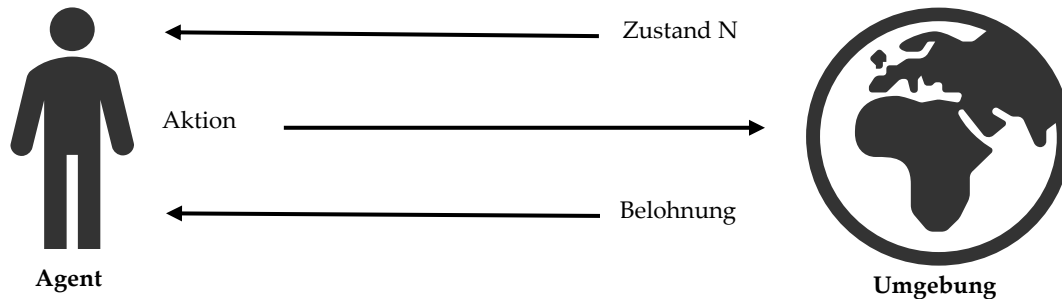


Abbildung 2: Graphische Darstellung des Reinforcement Learning (in Anlehnung an Schmitz 2017, S. 12)

Ausgehend von einem bestimmten Zustand der Umgebung entscheidet sich der Agent für eine verfügbare Aktion. In Abhängigkeit der Valuefunktion erhält der Agent dafür eine positive oder negative Belohnung, welche wiederum Einfluss darauf nehmen wird, welche Aktion beim nächsten Durchgang mit demselben Ausgangszustand gewählt wird. Je nach Dauer und Detaillierungsgrad des Trainings handelt es sich also um eine Kombination aus „Trial and Error“ und bereits erlerntem Wissen, welches der Agent für seine Entscheidungsfindung heranzieht (vgl. Lüth 2019, S. 2). Daraus folgt, dass der Agent umso sicherer in seiner Entscheidung wird, je häufiger er einen bestimmten Ausgangszustand mit darauffolgender Bewertung trainiert hat.

Üblicherweise wird die Umgebung als sogenannter Markov-Entscheidungsprozess (MEP) modelliert (vgl. Lüth 2019, S. 2; Schmitz 2017, S. 7). Der nach dem russischen Mathematiker Markov benannte MEP dient zur Modellierung eines Entscheidungsproblems, bei dem ausgehend von einem gewissen Zustand eines Systems beim Ausführen einer Aktion mit einer gewissen Wahrscheinlichkeit ein bestimmter Folgezustand erreicht wird. Jeder Zustandsübergang wird darüber hinaus durch eine Kostenfunktion bewertet (vgl. Böhm 2016, 14 f.). Außer der Beschreibung der Umgebung als Simulationsmodell ist darüber hinaus kein zusätzliches, menschliches Fachwissen in aufbereiteter Form für das Training erforderlich (vgl. Schmitz 2017, S. 12). Das Simulationsmodell muss darüber hinaus kein vollständig mathematisches oder stochastisches Modell sein, was insbesondere in komplexen Umgebungen mit vielen Einflussfaktoren zum Vorteil wird (vgl. Lüth 2019, S. 3).

Durch die Anwendung des „Trial-and-Error“ Prinzips folgt, dass das Training vorab auch entfallen und stattdessen im laufenden Betrieb einer Umgebung durchgeführt werden kann. In der Praxis ist das jedoch allenfalls für unkritische, unterstützende Anwendungen möglich, bei der der Nutzen durch Einsatz von RL einen eventuellen Schaden während der Trainingsphase eliminiert. Man stelle sich jedoch im Gegenzug vor, wo die Forschung rund um das autonome

Fahren nach heutigem Stand wäre, hätte man ein RL-Training für ein Fahrzeug wie im letzten Beispiel tatsächlich im öffentlichen Straßenverkehr durchgeführt.

2.4 Künstliche Intelligenz und maschinelles Lernen

Die Begriffe künstliche Intelligenz und maschinelles Lernen oder auch Maschinenlernen genannt (ML) tauchen oft im selben Kontext auf. In den letzten Jahren werden diese Schlüsselwörter dem Anschein nach besonders häufig dann verwendet, wenn ein Projekt nach außen als besonders innovativ oder forschungsintensiv dargestellt werden soll. Stellenweise werden beide Begriffe auch als Synonym verwendet. In diesem Kapitel werden KI, ML und verwandte Begriffe aus diesem Fachgebiet voneinander abgegrenzt.

Zur Verdeutlichung der Abgrenzung zwischen KI und ML sei das folgende Beispiel konstruiert:

In einem Eingabefeld können Nutzende Haltestellennamen eingeben und aus einem Dropdown mit Vorschlägen die gewünschte Haltestelle auswählen. Dabei werden Tippfehler in gewissen Grenzen toleriert und Vorschläge auch dann angezeigt, wenn keine exakte Übereinstimmung gefunden wurde. Werden Abkürzungen verwendet, die zu keiner direkten Übereinstimmung führen, merkt das System sich, welche Haltestelle die Nutzenden nach ihrer Eingabe aus den Vorschlägen auswählen und setzt diesen Vorschlag beim nächsten Suchvorgang im Ranking nach oben.

Die nachfolgend ausgeführten Leitfragen lauten nun:

- Handelt es sich bereits um eine Form der KI?
- Kommen hierbei Ansätze aus dem ML zum Einsatz?

Handelt es sich bereits um eine Form der KI?

Das System ist in der Lage, kleinere Schreibfehler zu korrigieren und den Nutzenden trotzdem passende Suchvorschläge anzuzeigen. Wird so beispielsweise „Kallsruh Hbf“ statt „Karlsruhe Hbf“ eingegeben, ist das System dennoch in der Lage, die korrekte Haltestelle „Karlsruhe Hbf“ zu ermitteln. Würde einem Menschen mit entsprechendem Fachwissen dieselbe Aufgabe stellen, wäre auch dieser in der Lage trotz des offensichtlichen Fehlers passende Ergebnisse vorzuschlagen.

Eine Möglichkeit, um solche Ergebnisse zu erhalten, ist die Verwendung sogenannter *unscharfer Mengen*. Im Gegensatz zu einer klassischen Menge, welche bezogen auf das obige Beispiel eine

einfache Liste aller verfügbaren Haltestellennamen im Klartext wäre, würde eine unscharfe Menge die Haltestellennamen unterteilt in Buchstabenblöcken zu jeweils drei Buchstaben enthalten. Diese Buchstabenblöcke werden auch als *Trigramme* bezeichnet. Aus dem Wort „Karlsruhe Hbf“ würden sich demnach die Trigramme

KAR ARL RLS LSR SRU RUH UHE HBF

ergeben.

Der durch die Nutzenden eingegebene Text wird ebenfalls in seine Trigramme zerlegt und für jedes Element in der unscharfen Menge ermittelt, wie viele Trigramme aus der Eingabezeichenfolge im jeweiligen Element der unscharfen Menge enthalten sind. Das Ergebnis ist ein Überdeckungsgrad auf dem Intervall $[0; 1]$.

Am Beispiel der Eingabezeichenfolge „Kallsruh Hbf“ ergäben sich die Trigramme

KAL ALL LLS **LSR** SRU RUH **HBF**

wovon die fett markierten Trigramme auch im entsprechenden Element der unscharfen Menge enthalten sind. Durch den entstehenden Überdeckungsgrad > 0 kann mit einer gewissen Wahrscheinlichkeit davon ausgegangen werden, dass mit der Eingabezeichenfolge eigentlich das korrespondierende Element der unscharfen Menge gesucht ist, wenngleich keine hundertprozentige Übereinstimmung mit der Eingabezeichenfolge vorliegt.

Die folgende Abbildung zeigt die graphische Darstellung einer klassischen und einer unscharfen Menge am Beispiel von Temperaturen im Vergleich:

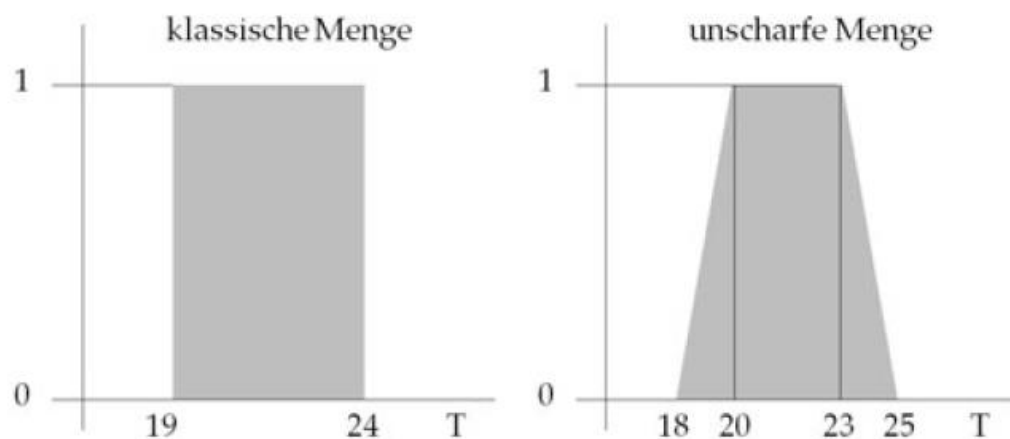


Abbildung 3: Darstellung einer klassischen und einer unscharfen Menge (Quelle: Nahrstedt 2012, S. 237)

Die Temperaturen werden in der Klassifikation „tief“, „normal“ und „hoch“ durch Verwendung der unscharfen Mengen nicht exakt abgegrenzt, sondern bilden das tatsächliche

Empfinden verschiedener Menschen ab. So liegt eine „niedrige“ Temperatur bei einer Person schon bei 15° C, bei einer anderen Person aber erst bei 5° C. Umgekehrt empfindet eine Person eine Umgebungstemperatur von 20° C bereits als „warm“, während eine andere Person hier erst im Bereich von „angenehm“ ankommt.

Diese Überdeckung von verschiedenen Temperaturempfinden ist in der folgenden Abbildung graphisch dargestellt:

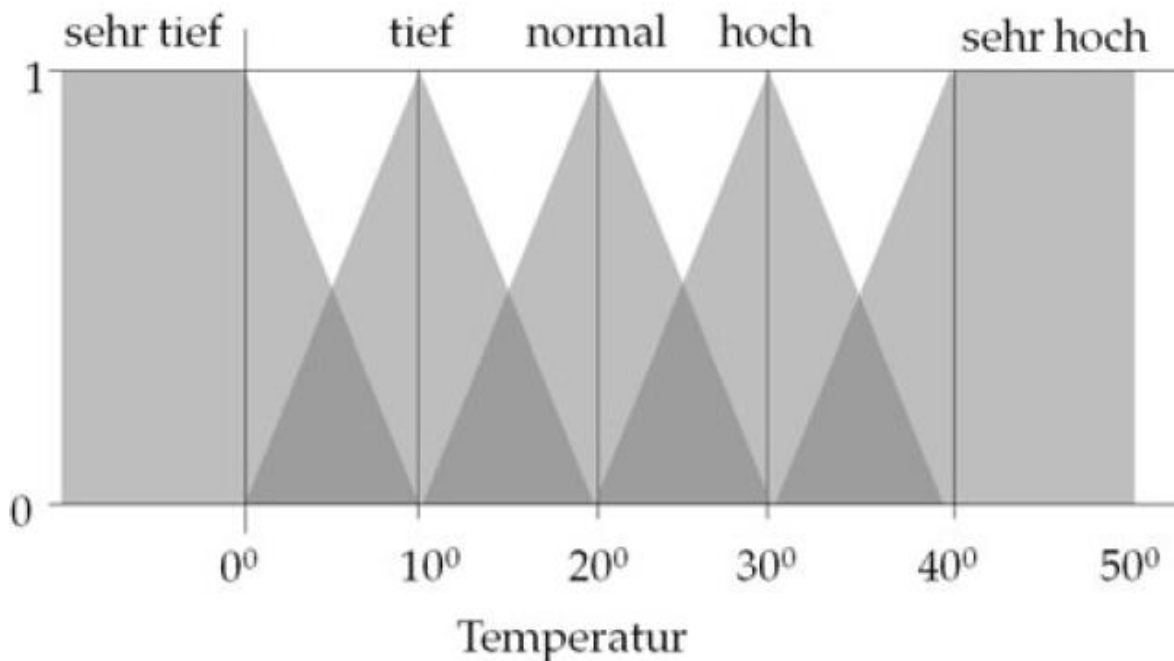


Abbildung 4: Graphische Darstellung der Überdeckung unscharfer Mengen (Quelle: Nahrstedt 2012, S. 238)

Dasselbe Prinzip lässt sich auf die Suche nach Haltestellenamen übertragen. Die Zerlegung der Eingabezeichenfolge und der tatsächlichen Haltestellenamen bilden mehrere unscharfe Mengen, die sich zu einem gewissen Grad überdecken können und damit eine teilweise Übereinstimmung feststellbar machen. Wäre die Suche hingegen in einer klassischen Menge durchgeführt worden, hätte dies zu keinem Ergebnis geführt, da die Eingabezeichenfolge mit keinem der Elemente exakt übereinstimmt. Dieses Verfahren wird auch als *unscharfe Suche* oder *Fuzzy-Suche* bezeichnet und ist eine bekannte KI-Anwendung (vgl. Engfer 2002, S. 3).

Die erste Bedingung, um von einer KI gemäß der zuvor festgelegten Arbeitsdefinition ausgehen zu können ist damit erfüllt. Die Qualität der Ergebnisse der KI ist im Nachhinein nicht von der Qualität der Ergebnisse eines Menschen mit entsprechendem Fachwissen zu unterscheiden. Folglich darf beim verwendeten Beispiel von einer Form der KI ausgegangen werden.

Kommen hierbei Ansätze aus dem ML zum Einsatz?

Grundsätzlich ist ML eine Möglichkeit, die „Programmen die Möglichkeit gibt, mit Hilfe von Daten zu lernen, ohne explizit programmiert zu werden“ (Niebler 2018, S. 10). Diese Bedingung ist im Beispiel der Haltestellensuche durch die Zuordnung von möglichen Eingaben zu tatsächlich existenten Haltestellen gegeben.

Die Haltestellensuche lässt sich mit diesen Erkenntnissen jedoch nicht exakt dem überwachten oder unüberwachten Lernen zuordnen. Das bestärkende Lernen scheidet per Definition bereits aus. Vielmehr hängt eine genaue Einordnung von der Implementierung der Haltestellensuche innerhalb eines Systems ab. Generell lässt sich jedoch festhalten, dass die Haltestellensuche aus dem Beispiel durchaus lernfähig ist, ohne dabei in die Implementierung einzugreifen. Angelehnt an die Definition von Niebler (2018) kann die Haltestellensuche zumindest klar dem Machine Learning zugeordnet werden.

2.5 Zusammenfassender Vergleich von ML-Verfahren

In den vorangegangenen Kapiteln wurde der Begriff des Machine Learning von der künstlichen Intelligenz abgegrenzt und in überwachtes, unüberwachtes und bestärkendes Lernen aufgeteilt. In diesem Kapitel werden die drei ML-Verfahren einander gegenübergestellt und zusammenfassend miteinander verglichen.

Überwachtes Lernen

Das überwachte Lernen arbeitet mit einem Eingangsdatensatz, in dem bestimmte Eigenschaften ihren korrekten Werten, den sogenannten Labels, zugeordnet sind. In der Trainingsphase wird das Modell mit dem Eingangsdatensatz trainiert. Nach Abschluss des Trainings wird das Modell mit einem zweiten Datensatz geprüft. Ziel ist die Evaluation des Modells und eine Überanpassung zu vermeiden. Das überwachte Lernen eignet sich besonders für Anwendungen, über die eine Vielzahl an möglichst umfangreichen Daten vorliegen. Optimalerweise sind diese Daten bereits gelabelt. Andernfalls muss zunächst menschliches Wissen eingebracht werden, um dem Modell eine Trainingsgrundlage zu schaffen

Unüberwachtes Lernen

Im Gegensatz dazu enthält der Trainingsdatensatz beim unüberwachten Lernen keine Labels, die Evaluation mittels eines zweiten Datensatzes entfällt ebenso. Das unüberwachte Lernen eignet sich damit besonders, um unbekannte Zusammenhänge zwischen Daten zu finden oder Daten in bestimmte Cluster einzuordnen. Menschliches Vorwissen ist folglich nicht erforderlich.

Bestärkendes Lernen

Abweichend von den beiden zuvor genannten ML-Verfahren benötigt das bestärkende Lernen, auch Reinforcement Learning genannt, gar keinen Trainingsdatensatz. Stattdessen findet das Training in einer Simulation des späteren Einsatzgebietes statt, das Modell trainiert sich selbst anhand eines "Trial-and-Error" Verfahrens. Damit eignet sich das bestärkende Lernen besonders für Anwendungen mit komplexen Einflussfaktoren, ohne dass zuvor alle möglichen Rückschlüsse durch menschliches Vorwissen ausreichend trainiert wurden.

2.6 Algorithmen aus dem Reinforcement Learning

In Kapitel 2 wurde der Begriff des Machine Learning bereits in die drei Kategorien überwachtes, unüberwachtes und bestärkendes Lernen - oder Reinforcement Learning - eingeordnet. Reinforcement Learning scheint für die Anwendung in dieser Arbeit besonders geeignet, da es weder einen gelabelten Eingangsdatensatz noch menschliches Vorwissen voraussetzt. Lediglich die Modellierung der Umgebung mit ihren wichtigsten Eigenschaften wird benötigt. Daher wird das Reinforcement Learning und die dem RL ungeordneten Algorithmen in diesem Kapitel einer tiefergreifenden Betrachtung unterzogen. Ziel ist es, zwei zur Implementierung im Prototyp und anschließendem Vergleich geeignete Algorithmen zu identifizieren.

Typisch für RL-Algorithmen ist, dass sie ihr Wissen selbstständig durch ein "Trial-and-Error" Verfahren in einer Simulation ihrer späteren Umgebung erarbeiten. Das Modell der Umgebung ähnelt dabei einem Markov-Entscheidungsprozess (vgl. Lüth 2019, S. 2). Für eine Umgebung mit einer überschaubaren Anzahl an Zuständen könnten die Ergebnisse des Trainingsprozesses einfach in einer Lookup-Table gespeichert und bei Bedarf abgerufen werden. Reelle Umgebungen können jedoch schnell eine unüberschaubare Anzahl an Zuständen annehmen, wie das folgende Beispiel verdeutlichen soll.

Im Beispieldatensatz in dieser Arbeit, welcher alle mit einem Linienbus befahrbaren Strecken im Einzugsgebiet zwischen der Gemeinde Engelsbrand und der Stadt Pforzheim enthält, sind

insgesamt 1245 Wege vorhanden, auf denen mögliche Routen berechnet werden können. Die Anzahl der Wege sei fortan als $|w|$ bezeichnet. Ein Systemzustand wird einerseits durch die Position eines Fahrzeuges auf einem der Wege und andererseits durch die Befahrbarkeit aller vorhandenen Wege ausgedrückt. Ein Fahrzeug kann sich je Zustand nur auf einem Weg befinden, es können jedoch mehrere Wege je Zustand in ihrer Befahrbarkeit eingeschränkt sein. Die Befahrbarkeit entspricht dabei einem binären Wert, der angibt, ob ein Weg befahrbar ist, oder nicht. Die Anzahl möglicher Zustände ergibt sich demnach angenähert aus:

$$|W| + 2^{|w|} = 1245 + 2^{1245} = 1245 + 1,071509 \cdot 10^{301}$$

Obwohl es sich um einen kleinen Datensatz handelt, sind das immerhin dreimal so viele theoretisch mögliche Zustände wie es Atome im gesamten Universum gibt! Eine Speicherung und zeitnahe Abfrage dieser Menge an Zuständen innerhalb einer Lookup-Table wäre zu ineffizient. Aus diesem Grund werden anstelle der Werte selbst die Parameter des gewählten RL-Algorithmus gespeichert und während dem Training angepasst, sodass nach dem Training eine mögliche Entscheidungsfindung in einer realen Umgebung möglichst gut approximiert wird (vgl. Dammann o.D., S. 9). Die Approximation hat außerdem den positiven Nebeneffekt, dass das Ergebnis des Trainings allgemeingültig definiert bleibt und nicht nur fix auf bereits trainierte Zustände angewandt werden kann (vgl. Schmitz 2017, S. 18).

2.6.1 Wert- und Strategieapproximation

Weiter gilt es zu unterscheiden zwischen Wert- und Strategie-Approximationsalgorithmen. Diese werden respektive auch als Off-Policy- und On-Policy-Algorithmen bezeichnet. Während On-Policy-Algorithmen dieselbe Funktion zur Bewertung und Steuerung des Agenten eingesetzt wird, können diese Funktionen bei Off-Policy-Algorithmen diese beiden Funktionen getrennt betrachtet werden (vgl. Lorenz 2020, S. 62). In diesem Fall gibt es also jeweils eine Value- und eine Policy-Funktion. Während erstere für die Bewertung der Ausführung einer Aktion in einem bestimmten Zustand (vgl. Schmitz 2017, S. 13) verantwortlich ist, steuert letztere das Verhalten des Agenten in der Umgebung durch die Auswahl einer Aktion in einem bestimmten Zustand anhand ihrer Auswahlwahrscheinlichkeit (vgl. Schmitz 2017, S. 16). In diesem Unterkapitel werden die wesentlichen Unterschiede beider Approximationsarten erläutert.

Bei Off-Policy-Algorithmen wird eine Wert-Funktion definiert, welche einzelne mögliche Zustände oder mögliche Aktionen innerhalb eines Zustandes bewertet. Hierzu wird das Ursprungsproblem in mehrere Teilprobleme zerlegt, deren optimale Teillösungen dann zur einer optimalen Gesamtlösung führen. Nach Ausführen einer Aktion und dem Erreichen eines Folgezustandes wird die erwartete und die tatsächlich erhaltene Belohnung des Agenten überprüft und die Parameter der Wert-Funktion entsprechend angepasst (vgl. Dammann o.D.,

S. 6). Durch die Initialisierung der Wert-Funktion mit niedrigen Bewertungen bleiben die die Bewertungen von Zuständen oder Aktionen in bestimmten Zuständen nach jedem Schritt entweder gleich oder werden verbessert, bis sie schließlich gegen eine Maximalbewertung konvergieren. Unterschreitet die maximale Bewertungsänderung nach einem Schritt einen festzulegenden Schwellenwert, kann das Training als beendet betrachtet werden. Das Ausführen der Aktionen, welche zur entsprechend höchsten Bewertung führen, beschreibt dann die optimale Strategie eines Agenten (vgl. Lorenz 2020, S. 23).

Im Gegensatz dazu wird bei On-Policy-Algorithmen versucht, direkt die Parameter der Strategie-Funktion anzupassen. Der Weg über die Bewertungsfunktion entfällt dabei. Die Strategie-Funktion beschreibt eine Wahrscheinlichkeit dafür in einem Zustand eine bestimmte Aktion zu wählen (vgl. Schmitz 2017, S. 16). Bei der Strategieapproximation gilt es, die Parameter der Strategie-Funktion derart anzupassen, dass Aktionen mit möglichst hohem erwartetem Gewinn gewählt werden. Das Parameterupdate kann dazu nach jedem Einzelschritt, aber auch erst nach Abschluss einer gesamten Episode durchgeführt werden (vgl. Schmitz 2017, S. 16). In gefährlichen Umgebungen, in denen auch kleine Fehler schwerwiegende Folgen haben können, sind On-Policy-Algorithmen besser geeignet, da sie Verluste aus dem Training berücksichtigen und generell Aktionen mit höherer Bewertung bevorzugen (vgl. Williams 1992, S. 288).

Jeder RL-Algorithmus nutzt zumindest eine Wert- oder Strategiefunktion zur Steuerung seines Agenten (vgl. Schmitz 2017, S. 18). Einer der Hauptunterschiede beider Strategien ist, dass Off-Policy-Algorithmen zunächst den erwarteten Gewinn des Agenten schätzen, jedoch erst nach dem tatsächlichen Ausführen einer Aktion Kenntnis über den tatsächlichen Gewinn erlangen. On-Policy-Algorithmen arbeiten hingegen ausschließlich auch den Erfahrungswerten tatsächlich ausgeführter Episoden.

2.6.2 Monte-Carlo- und Temporal-Difference-Methoden

Außerdem können RL-Algorithmen in den Monte-Carlo-Methoden (MC-Methoden) oder Temporal-Difference-Methoden (TD-Methoden) zugeordnet werden.

MC-Algorithmen basieren auf einer zufallsbasierten Exploration. Zunächst wird eine hinreichend große Anzahl Episoden ausgehend von einem bestimmten Zustand durchgeführt, welche im Nachgang bewertet werden. Die Bewertungen der unterschiedlichen Episoden werden anschließend gemittelt (vgl. Wagner 2018, S. 31). Die Bewertung führt anschließend zu einem Update der Bewertungsfunktion (vgl. Lorenz 2020, S. 56). Entsprechend setzen MC-Algorithmen voraus, dass es mindestens einen Terminalzustand gibt (vgl. Wagner 2018, S. 31). Da MC-Algorithmen ausschließlich auf der Erhebung und Auswertung von

Zufallsexperimenten basieren, kommen sie gänzlich ohne ein mathematisches Modell aus. Man spricht daher auch von *modellfreien Algorithmen*.

Im Gegensatz dazu stehen TD-Algorithmen, welche anstelle von ganzen Episoden jeweils Einzelschritte in zeitlichen Abständen betrachten und bewerten (vgl. Lorenz 2020, S. 59). Die Bezeichnung rührt aus der Annahme, dass die Einzelschritte zeitlich betrachtet nacheinander ausgeführt und bewertet werden. TD-Algorithmen nutzen die Eigenschaft der Bellmangleichung aus, der zufolge nach die Bewertung eines Folgezustandes direkt von der Bewertung des aktuellen Zustandes abhängt (Wagner 2018). Folglich ist ausgehend von einem bestimmten Zustand möglich, die erwartete Belohnung des Agenten für alle möglichen Folgezustände abzuschätzen. Wie MC-Algorithmen auch, können TD-Algorithmen komplett ohne ein Modell der Umgebung auskommen und zählen damit ebenso zu den modellfreien Algorithmen (vgl. Arnold 2021, S. 13). Durch Berücksichtigen einer Lernrate können die Wahrscheinlichkeiten aus einem Markov-Modell mit einbezogen werden. Im Verlauf des Trainings wird die Lernrate beispielsweise mit jedem Besuch eines Zustandes verkleinert werden, sodass die Bewertung schließlich stabilisiert wird (vgl. Lorenz 2020, S. 60). Die Berücksichtigung von Wahrscheinlichkeiten aus einem Markov-Modell setzt allerdings voraus, dass die Umgebung zumindest mit ihren relevantesten Eigenschaften als solches definiert ist.

2.6.3 Einordnung bekannter RL-Algorithmen

In den vorhergehenden Unterkapiteln wurden zwei Klassifizierungsarten für RL-Algorithmen eingeführt. In diesem Unterkapitel werden ausgewählte RL-Algorithmen vorgestellt und in ihre entsprechende Klasse eingeordnet. Diese Einordnung ermöglicht später eine gezielte Auswahl zweier zum Vergleich geeigneter RL-Algorithmen.

2.6.3.1 Q-Learning

Der Q-Learning-Algorithmus, nachfolgend abgekürzt als Q-Learning bezeichnet ist der älteste RL-Algorithmus aus dem Jahr 1989 und kommt ohne Modell einer Umgebung aus (vgl. Lorenz 2020, S. 61). Q-Learning basiert auf einer Bewertung durch eine sogenannte *Action-Value-Funktion* aus der Klasse der Wertfunktionen. Die Action-Value-Funktion gibt an, welchen Gewinn der Agent zu erwarten hat, wenn er ausgehend von einem bestimmten Zustand eine bestimmte Aktion ausführt. Ein trainierter Agent ist in der Lage, eine optimale Strategie für Handlungen in einem MEP zu finden. Voraussetzung dafür ist eine hinreichend große Anzahl an Trainingsversuchen mit entsprechend ausführlicher Exploration in der Trainingsphase (vgl. Arnold 2021, S. 14). Nach dem Training wird in jedem Zustand jene Aktion gewählt, die dem Agenten den höchsten erwarteten Gewinn einbringt (vgl. Schmitz 2017, S. 13). Das gierige Verhalten entspricht einer Greedy-Strategie (vgl. Gass und Fu 2013, S. 666).

Das Training in Q-Learning kann beispielsweise durch Hinzufügen eines zusätzlichen Zufallsparameters in die Strategie-Funktion erfolgen. Mit diesem Zufallsparameter wird festgelegt, dass der Agent mit der Wahrscheinlichkeit des Zufallsparameters die Aktion mit der höchsten Bewertung auswählt. Alternativ wird eine beliebige, andere Aktion mit einer niedrigeren Bewertung gewählt. Je größer der Zufallsparameter ist, desto höher ist die Explorationsrate, durch die der Agent verschiedene Aktionen auswählt und letztendlich trainiert wird (vgl. Dammann o.D., S. 8). Der Zufallsparameter kann zu Beginn des Trainings sehr groß gewählt werden, um die Exploration zu fördern und mit fortschreitendem Training schrittweise verkleinert werden. Bedingt durch den Zufallsparameter ist es dadurch auch möglich, dass die tatsächlich gewählte Aktion nicht jene mit dem höchsten erwarteten Gewinn ist. Die Episode mehrerer Aktionen spielt für die im Q-Learning erarbeitete Strategie daher keine Rolle (vgl. Schmitz 2017, S. 14).

Da Wert- und Strategiefunktion in diesem Fall getrennt betrachtet werden können, kann Q-Learning also den Off-Policy-Algorithmen zugeordnet werden. Die schrittweise, nicht-episodische Bewertung macht außerdem eine Einordnung des Q-Learning bei den TD-Algorithmen möglich.

2.6.3.2 Deep Q-Learning

Eine Sonderform des Q-Learning stellt das Deep Q-Learning dar. Nachfolgend wird das Deep Q-Learning auch als Deep Q-Network (DQN) bezeichnet. Hierbei werden klassisches Q-Learning und Deep Learning kombiniert. Ein künstliches neuronales Netz (KNN) wird dabei als Funktionsapproximator für die Wert-Funktion eingesetzt (vgl. Larsson 2018, S. 19). Als Eingangsdaten erhält das KNN den aktuellen Zustand, als Ergebnis erhält man die Bewertungen der einzelnen Aktionen (vgl. Dammann o.D., S. 11; Lüth 2019, S. 8). Anstelle einer initial definierten Wert-Funktion wird lediglich eine Bewertungs-Funktion benötigt, welche „beurteilt, wie gut die gewählten Input-Output-Paare sind“ (Huber 2018, S. 23). Eine Anpassung der gesonderten Strategie-Funktion durch das Training erfolgt nicht, weshalb auch das Q-Learning den Off-Policy-Verfahren zugeordnet werden kann (vgl. Huber 2018, S. 27). Die Updates erfolgen jeweils nach jedem Einzelschritt, weshalb eine Zuordnung zu den TD-Algorithmen nahe liegt.

Das Training von KNN mit sequenziellen Updates, wie es beim RL der Fall ist, kann bedingt durch unerwünschte Korrelationen zwischen den Daten zu einer Verfälschung durch Fehlanpassung des KNN führen (vgl. Larsson 2018, S. 19). Der DQN-Algorithmus kompensiert dieses Verhalten zum einen durch die Verwendung zweier KNN mit unterschiedlichen Parametern und zum anderen durch die regelmäßige Einbindung von Erfahrungswerten während des Trainings. Die beiden KNN werden jeweils als *Online Network* und als *Target Network* bezeichnet. Zwar führt diese Verfahrensweise zu einem langsameren, dafür stabileren

Training des KNN und folglich zu einer exakteren Approximation der Wert-Funktion (vgl. Larsson 2018, 19 f.).

2.6.3.3 SARSA

Der SARSA-Algorithmus lernt im Vergleich zum Q-Learning nicht aus dem nächsten beobachteten Zustand, sondern aus dem tatsächlich erreichten Gewinn, der durch den Übergang in einen Zustand erreicht wurde (vgl. Lorenz 2020, S. 62). Der wesentliche Unterschied zum Q-Learning besteht in der Steuerung des Agenten. Im SARSA-Algorithmus wird der Agent ausschließlich über seine Strategie-Funktion gesteuert, die gleichzeitig explorativ ausgeprägt ist (vgl. Lorenz 2020, S. 62). SARSA ist folglich den On-Policy-Algorithmen zuzuordnen.

SARSA ist ein TD-Algorithmus, das heißt, die Updates folgen jeweils nach dem nächsten Schritt (vgl. Arruda et al. 2020, S. 4). Alternativ besteht beim SARSA-Algorithmus auch die Möglichkeit, eine sogenannte *Aktionshistorie* zu führen. In dieser Aktionshistorie werden alle Aktionen, die Teil einer bestimmten Episode sind, rückwirkend gespeichert. Über einen Parameter wird geregelt, wie viele Episoden rückwirkend berücksichtigt werden. Hat dieser Parameter den Wert 1, wird exakt eine rückwirkende Episode berücksichtigt. SARSA wird dadurch zu einem MC-Algorithmus (vgl. Lorenz 2020, S. 70). Je nach Ausprägung des Parameters werden die Eigenschaften von MC- und TD-Algorithmen kombiniert. Oft genutzte Aktionen, welche Teil vieler erfolgsversprechender Episoden sind, bremsen allerdings die Exploration aus, da diese Aktionen nur schwer wieder verlassen werden (vgl. Lorenz 2020, S. 71). Letztendlich kommt es also auf die genaue Implementierung des SARSA-Algorithmus an, in welcher Klasse SARSA eingeordnet werden kann.

Verglichen mit Q-Learning kann SARSA Ergebnisse näher an der Realität erzielen, da nicht ausschließlich die Aktionen mit dem erwarteten höchsten Gewinn gewählt werden, sondern insbesondere auch Erfahrungswerte aus vergangenen Episoden mit in die Entscheidungsfindung der Strategie-Funktion mit einbezogen werden. Außerdem findet auch nach dem Training Exploration statt, die jedoch besonders in gefährlichen Umgebungen kritisch zu beurteilen ist.

Wie auch Q-Learning kann SARSA durch Nutzung eines KNN umgesetzt werden. In diesem Fall spricht man von Deep SARSA, bei dem dann die Strategie-Funktion durch ein KNN approximiert wird. Die Funktionsweise ist dabei gleich, wie auch beim Q-Learning.

2.6.3.4 REINFORCE

Im Jahr 1992 von Williams definiert, arbeitet der klassische REINFORCE-Algorithmus Ergebnisse aus dem Sampling der Gesamtbewertung verschiedener Episoden (vgl. Schmitz 2017, S. 16). Er ist damit klar den MC-Algorithmen zuzuordnen. Prinzipiell ist die Funktionsweise des REINFORCE-Algorithmus ähnlich wie beim Q-Learning. Wesentliche Unterschiede bestehen allerdings zum einen darin, welche Rückschlüsse aus der approximierten Funktion gezogen werden und andererseits in der Art und Weise des Trainings (vgl. Schmitz 2017, 17 f.).

Die approximierte Strategie-Funktion gibt an, mit welcher Wahrscheinlichkeit eine bestimmte Aktion in einem bestimmten Zustand gewählt wird (vgl. Schmitz 2017, S. 16). Entsprechend handelt es sich nicht um eine deterministische, sondern um eine stochastische Strategie-Funktion. Eine Wert-Funktion gibt es bei REINFORCE nicht, daher zählt REINFORCE zu den On-Policy-Algorithmen. Beim Training wird statt einzelner Schritte jeweils eine ganze Episode ausgeführt, weshalb REINFORCE zweifelsfrei den MC-Algorithmen zuzuordnen ist. Zwar könnte das Update auch nach jedem Schritt erfolgen, allerdings steht der zu berücksichtigende, erreichte Gesamtgewinn erst nach Abschluss einer Episode fest (vgl. Schmitz 2017, S. 17). Wie auch bei SARSA kann bei REINFORCE eine Aktionshistorie geführt werden, die dann in der Strategie-Funktion fortlaufend eingebunden wird (vgl. Schmitz 2017, S. 17).

2.7 Grundbegriffe aus dem ÖPNV-Betrieb

In diesem Unterkapitel werden der Reihe nach einige wichtige Grundbegriffe aus dem Betrieb des öffentlichen Personennahverkehrs definiert, welche für das spätere Verständnis der Arbeit von Nöten sind.

2.7.1 Betriebstag

Generell betrachtet geht man davon aus, dass ein Tag 24 Stunden hat. Diese Sichtweise stößt dann an ihre Grenzen, wenn Fahrten abgebildet werden sollen, die über 23:59 Uhr hinaus verkehren. Im Fachjargon spricht man daher von *Betriebstagen*, die auch länger als 24 Stunden sein können. Entsprechend gibt es im betrieblichen Sinne auch Zeiten wie 25:08 Uhr. Gemeint ist damit beispielsweise der folgende Kalendertag um 01:08 Uhr morgens. Der *Betriebsschluss* gibt die Zeit an, zu der die letzte Fahrt endet. Zur Vereinheitlichung wurde der Betriebsschluss

weitestgehend einheitlich auf 03:00 Uhr festgelegt. Ein Betriebstag hat demzufolge 26 Stunden und 59 Minuten.

2.7.2 Linie und Linienvariante

Typischerweise sind ÖPNV-Netze in *Linien* eingeteilt. Eine Linie ist per Definition eine Streckenführung mit einem regelmäßig in beide Richtungen verkehrenden Fahrtenangebot entlang fest definierter Haltestellen (vgl. Reinhardt 2018, S. 456). Einzelne Fahrten können in Teilen von der Linienführung abweichen und beispielsweise nur einen Abschnitt der Linie bedienen. Ein typisches Beispiel hierfür sind Schnellbuslinien, die als Zu- und Abbringer zum nächstgrößeren Verkehrsknotenpunkt dienen. Die einzelnen Fahrwege aller Fahrten einer Linie werden in diversen Planungssystemen als *Linienvariante* hinterlegt. Auch im Fachjargon unter Verkehrsunternehmen hat sich dieser Begriff durchgesetzt. Entsprechend besteht eine Linie grundlegend aus einer Linienvariante für die Hin-Richtung und einer Linienvariante für die Rück-Richtung. Weitere Linienvarianten kommen für jeden abweichenden oder auch verkürzten Fahrweg respektive hinzu.

2.7.3 Umlauf- und Dienstplan

Jede Fahrt innerhalb einer Linie muss mit mindestens einem Fahrzeug besetzt sein. Alle Fahrten, die ein Fahrzeug an einem Betriebstag durchführt, werden durch den sogenannten *Umlauf* zusammengefasst. Anhand eines Umlaufes können alle Fahrten ermittelt werden, die von einem bestimmten Fahrzeug planmäßig durchgeführt werden. Muss ein Umlauf bedingt durch Abweichungen im Betriebsablauf neu besetzt werden, gilt es dabei eine Vielzahl an Parametern zu beachten. Auszugsweise genannt seien hier Wendezeiten an den Endhaltestellen, eventuelle Folgefahrten- und Umläufe eines Fahrzeuges und zugewiesene Stellplätze in Betriebshöfen und Abstellanlagen (vgl. Reinhardt 2018, 496 f.). Während der Umlauf den geplanten Einsatz eines Fahrzeuges vorgibt, regelt ein *Dienst* die Abfolge von Tätigkeiten, die das Fahrpersonal über einen Betriebstag hinweg verteilt durchführt. Analog dazu gilt es einen noch größeren Umfang an Parametern bei der Planung von Diensten und eventuellen Eingriffen im Tagesverlauf zu beachten. Allen voran seien hier das Arbeitszeitgesetz, die Lenk- und Ruhezeiten und eventuell anzuwendende Regeln aus einem Tarifvertrag genannt. In bestimmten betrieblichen Konstellationen können Dienst- und Umlaufplan gleich sein.

2.7.4 Betriebsstabilität und dispositive Maßnahme

Das Ziel während einem Betriebstag ist es, den Sollfahrplan so gut wie möglich einzuhalten. Man spricht von einem entsprechend *stabilen Betrieb*. Dennoch kommt es im Verlauf des Betriebstages immer wieder zu Abweichungen, die ein Eingreifen notwendig machen. Optimalerweise erfolgt dieser Eingriff durch die Betriebsleitstelle oder eine andere befugte Person. Vielfach angewandt wird der sogenannte *Verspätungsausgleich*, bei dem eine Fahrt gegen Ende vorzeitig beendet und auf die Folgefahrt gewechselt wird. Dadurch entfällt zwar ein Teil einer der ursprünglichen Fahrt, dafür kann die Folgefahrt im Optimalfall pünktlich gestartet werden. Wird hingegen ein Streckenabschnitt kurzfristig über längere Zeit gesperrt, muss eine *Umleitung* angeordnet werden. Optimalerweise erfolgt die Umleitung so, dass ersatzweise Haltestellen bedient werden, die geographisch möglichst nahe an den planmäßigen Haltestellen einer Fahrt liegen, zur Sicherung der Betriebsstabilität können aber einzeln zu bestimmende Halte gezielt entfallen. Ziel der Umleitung ist es, eine größere Verspätung des Fahrzeuges abzufedern, was jedoch nicht immer erreicht werden kann. Solche und vergleichbare Eingriffe in den Betriebsablauf werden als *Dispositionsmaßnahmen* bezeichnet (vgl. Schranil 2013, S. 124). In Extremfällen kann es notwendig sein, mehrere dispositive Maßnahmen nacheinander anzuordnen, um die Betriebsstabilität zu gewährleisten. Zusätzlich zu den Beispielen von Umlauf- und Dienstplanung seien hier das Fahrgastaufkommen und mögliche Anschlüsse an Knotenpunkten genannt. Letztendlich hängt es von einer Vielzahl von Einflussfaktoren ab, welche Maßnahmen in welchem Fall geeignet sind. Im Fokus dieser Arbeit steht die automatische Suche nach Umleitungen für Linienbusse im Störfall, andere dispositive Maßnahmen werden daher nicht weiter betrachtet.

2.7.5 Betriebsstörung

Als Arbeitsdefinition für den Begriff der Störung kann zunächst jede Veränderung eines Systems betrachtet werden, welche eine Abweichung vom Plan erforderlich macht oder unmittelbar zur Folge hat. Zu unterscheiden ist dabei in *technische* und *betriebliche Störung*. Während erstere eine bestimmte Funktion außer Kraft setzt, hat sie nicht zwangsläufig einen Einfluss auf den laufenden Betrieb. Letztere ist für diese Arbeit von Relevanz und beschreibt eine Störung des planmäßig geregelten Betriebsablaufs (vgl. Schranil 2013, S. 16). So handelt es sich beispielsweise beim Ausfall der Haltestellenbeleuchtung um eine technische Störung, die jedoch betrieblich keine unmittelbare Auswirkung auf den laufenden Betrieb hat. Störungen auf der Strecke machen hingegen die Umleitung einzelner Fahrten erforderlich und sind damit den betrieblichen Störungen zuzuordnen. Eine Fahrzeugstörung kann sowohl den technischen als auch den betrieblichen Störungen zugeordnet werden, da die zunächst technische Störung

unmittelbare Auswirkungen auf den Betrieb hat. In dieser Arbeit wird stellvertretend für alle betrieblichen Störungen der Begriff *Betriebsstörung* festgelegt.

2.7.6 Innerbetriebliche und öffentliche Information

Der *innerbetriebliche und öffentliche Informationsfluss* ist essenziell zum Aufrechterhalten der Betriebsstabilität. Bei dispositiven Maßnahmen wollen Fahrgäste erwiesenermaßen zeitnah über die Störung selbst und mögliche Alternativen informiert werden (vgl. Brezina et al. 2012, S. 11). Die innerbetriebliche Information stellt darüber hinaus einen geregelten und gleichmäßigen Betriebsablauf sicher (vgl. Scherm et al. 2001, 44 f.). Die Pünktlichkeit und damit einhergehend die Betriebsstabilität ist ähnlich hoch zu bewerten (vgl. Findl et al. 2022, S. 6). Qualifiziertes und erfahrenes Leitstellenpersonal ist in der Lage, Informationen zielgerichtet und korrekt einzubringen und trägt damit einen erheblichen Teil zu einer stabilen Betriebsabwicklung und damit nicht zuletzt auch zur wahrgenommenen Qualität des Verkehrsangebotes auf Seiten der Fahrgäste bei.

2.7.7 Rechnergestütztes Betriebsleitsystem

Das *rechnergestützte Betriebsleitsystem* (RBL) ist die zentrale Steuereinheit zwischen Betriebsleitstelle und Fahrzeug im ÖPNV-Betrieb. Sie umfasst „die Steuerung der Informations- und Kommunikationsmöglichkeit zwischen Fahrzeugen und Leitstelle, die Steuerung des Fahrbetriebs und die Aktualisierung der Fahrgastinformation in den Fahrzeugen und an den Haltestellen“ (Reinhardt 2018, 520 f.). Zwischenzeitlich ist der Begriff *Intermodal Transportation Control System* (ITCS) als Synonym anstelle des RBL getreten. Zur Vereinheitlichung wird in dieser Arbeit jedoch ausschließlich letztere Bezeichnung verwendet. Über das ITCS besteht eine bidirektionale Kommunikationsschnittstelle zwischen Betriebsleitstelle und Fahrzeug beziehungsweise Fahrpersonal. Durch die Ortung im Fahrzeug hat die Betriebsleitstelle stets Kenntnis vom aktuellen Standort eines Fahrzeugs. Ist zudem ein Soll-Fahrplan im System hinterlegt, kann die Betriebsleitstelle auf einen Blick Fahrzeuge sehen, die vom Soll abweichen. Das betrifft sowohl Fahrzeiten als auch Fahrweg. Über eine Mobilfunkschnittstelle kann die Betriebsleitstelle dem Fahrpersonal Anweisungen erteilen und das Fahrpersonal die Betriebsleitstelle außerdem über Störungen im Betriebsablauf informieren. Ebenso können über das ITCS Fahrgäste über die Informationsbildschirme im Fahrzeug und an den Haltestelle über Abweichungen informiert werden (vgl. Reinhardt 2018, 520 f.).

2.7.8 Bordrechner

Der *Bordrechner*, oftmals auch als Bord-Unit bezeichnet, ist das Gegenstück zum ITCS auf Fahrzeugebene. Der Bordrechner hält den Sollfahrplan vor, sodass selbst bei unterbrochener Mobilfunkverbindung ein autonomer Betrieb mit funktionierenden Haltestellenansagen und Fahrastinformation möglich ist und bündelt darüber hinaus alle Informationen, die im Fahrzeug zusammenlaufen (vgl. Reinhardt 2018, S. 521). Hierzu zählen unter anderem Standort, Wegzählerimpuls und Protokolle über Handlungen des Fahrpersonals. Durch den Bordrechner werden all diese Daten schließlich dem ITCS übergeben.

3 Theoretische Modellierung und Konzeption

In diesem Kapitel werden zunächst die theoretischen Grundlagen zur Modellierung des Problems erörtert. Hierzu werden zunächst praxisnahe Beispielszenarien konstruiert, welche im Nachgang als Grundlage zur Evaluation dienen sollen. Im folgenden Unterkapitel wird ermittelt, welcher Umfang an Daten zumindest benötigt wird. Anschließend werden basierend auf der Verfügbarkeit der Daten und den ersten Erkenntnissen aus **Kapitel 2** zwei geeignet erscheinende Algorithmen aus dem Bereich des Reinforcement Learning ausgewählt. Das Kapitel schließt mit der Aufstellung eines theoretischen Modells, welches im folgenden Kapitel prototypisch für Versuchszwecke umgesetzt wird.

3.1 Aufstellung geeigneter Beispielszenarien

Im Rahmen der Arbeit sollen die Ergebnisse zweier ML-Algorithmen miteinander verglichen werden. Hierzu ist es zunächst erforderlich, geeignete Szenarien festzulegen, welche einen praxisnahen objektiven und subjektiven Vergleich der Ergebnisse ermöglichen. Die Szenarien sollen zumindest folgende Anforderungen erfüllen:

- Es sollen sowohl städtische als auch ländliche Gebiete erfasst werden
- Stadt- und Regionalbuslinien sollen gleichermaßen inkludiert sein
- Betrachtete Buslinien sollen möglichst unterschiedliche Regelfahrwege oder Haltestellenfolgen aufweisen
- Je Szenario soll ein Störfall trainiert werden

Basierend auf diesen Maßregeln werden zwei Szenarien aus dem Raum Pforzheim und Umland ausgewählt. Betroffen sind jeweils zwei Regionalbuslinien, eine davon als Schnellbuslinie und eine Stadtbuslinie. Bedingt durch die unterschiedliche Haltepolitik der Linien ergibt sich je nach geographischem Betrachtungsraum ein unterschiedlicher Fahrweg mit entsprechend voneinander abweichenden Haltestellenfolgen.

Stadt-Szenario

Betrachtet werden soll der Abschnitt der Kaiser-Friedrich-Straße in der Stadt Pforzheim. Auf diesem Abschnitt verkehren die Regionalbuslinien 743 und 744, sowie die Stadtbuslinie 2 jeweils in beide Fahrtrichtungen. Alle Linien haben auf diesem Abschnitt denselben Regelfahrweg, die Regionalbuslinien bedienen im Gegensatz zur Stadtbuslinie jedoch zwei

Haltestellen nur zum Ausstieg. Die Regionalbuslinien verkehren in einem 15/30min-Takt, die Stadtbuslinie verkehrt im 15min-Takt. Aufgrund eines schweren Verkehrsunfalls ist die Kaiser-Friedrich-Straße im östlichen Teil für mindestens zwei Stunden voll gesperrt, sodass die Buslinien weiträumig umgeleitet werden müssen. Die folgende Abbildung zeigt den betrachteten Raum als Kartenansicht:

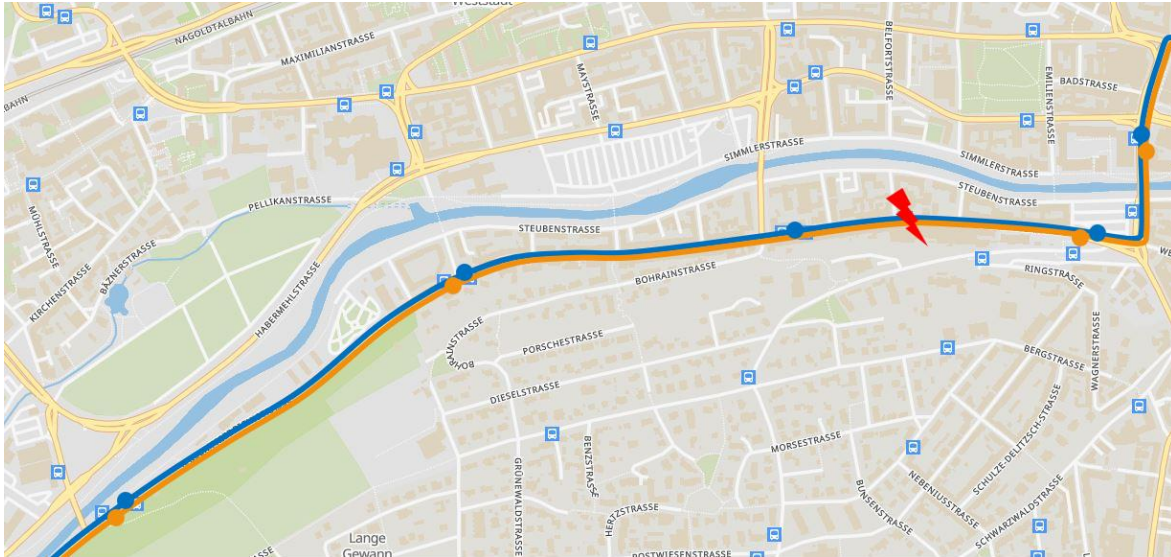


Abbildung 5: Beispielszenario für Linienbusse im Stadtverkehr

In blau markiert sind der Fahrweg der Stadtbuslinie und deren zum Einstieg bediente Haltestellen, in orange markiert entsprechend die der Regionalbuslinien. Darüber hinaus sind weitere Haltestellen im Stadtgebiet mit entsprechenden Symbolen dargestellt. Der rote Blitz markiert die Unfallstelle. Die Karte ist in vergrößerter Ansicht im Anhang zu finden.

Land-Szenario

Betrachtet werden soll der Abschnitt zwischen den Gemeinden Langenbrand mit Ortsteil Brückenäcker im Südwesten und Engelsbrand mit den Ortsteilen Salmbach und Grunbach im Norden. Auf diesem Abschnitt verkehren die Regionalbuslinien 743 und 744. Die Linie 743 ist eine Schnellbuslinie und bedient in der Gemeinde Engelsbrand nur den Ortsteil Salmbach und fährt dann direkt weiter in Richtung Pforzheim. Die Linie 744 beginnt im Ortsteil Salmbach, bedient dann die Ortsteile Engelsbrand und Grundbach und fährt dann in Richtung Pforzheim. Einige Fahrten der Linie 744 beginnen bereits in Kapfenhardt und verkehren ab Salmbach dann über den regulären Fahrweg der Linie 744 in Richtung Pforzheim. Die Linie 743 verkehrt im 60min-Takt, die Linie 744 im 30min-Takt jeweils in beide Fahrtrichtungen. Aufgrund eines umgestürzten Baumes ist die Landstraße zwischen den Orten Langenbrand und Salmbach in Richtung Salmbach einseitig gesperrt, sodass die Busse aus Richtung Schömberg und

Kapfenhardt kommend umgeleitet werden müssen. Die Gegenrichtung ist mit vorsichtiger Fahrweise befahrbar. Die folgende Abbildung zeigt den betrachteten Raum als Kartenansicht:

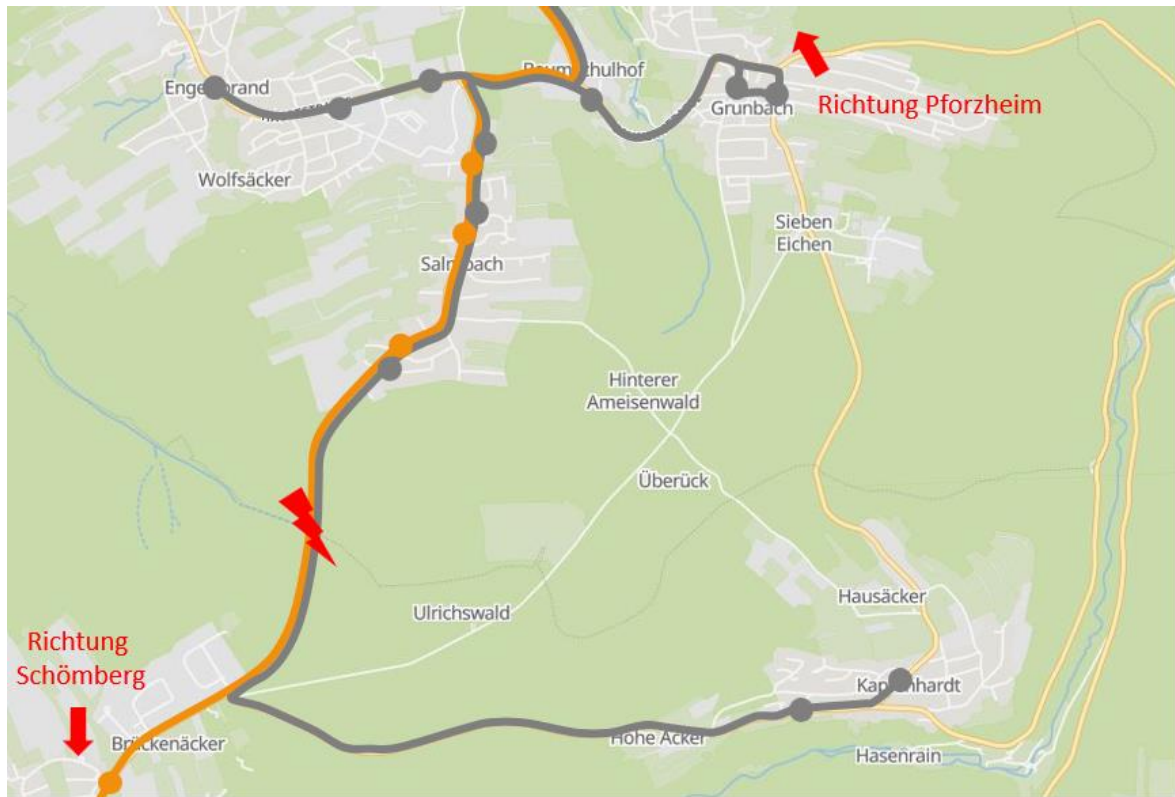


Abbildung 6: Beispielszenario für Linienbusse im Regionalverkehr

Zu sehen sind die Regionalbuslinien 743 in orange, die Linie 744 in grau mit den entsprechend planmäßig bedienten Haltestellen. Weitere Haltestellen sind in dieser Ansicht nicht gesondert dargestellt, da diese bedingt durch die Zoomstufe der Karte eher zu ein Überfrachtung führen würden, als zweckdienlich zu sein. Zur Orientierung sind außerdem die Fahrtrichtungen gekennzeichnet. Der rote Blitz markiert die Stelle, an der die Straße einseitig unterbrochen ist. Die Karte ist in vergrößerter Ansicht im Anhang zu finden.

3.2 Auswahl verfügbarer Eingangsdaten

Anstatt eine vollständige Auflistung aller teils sogar frei unter OpenData-Lizenzen verfügbaren Daten anzustreben, ist es sinngemäßer zunächst nach dem gewünschten Zweck zu kategorisieren und dann dazu passende Daten auszuwählen. In diesem Unterkapitel werden

zunächst die Zwecke aufgelistet und passend dazu jeweils verfügbare Datenquellen erläutert. Zudem werden notwendige Transformationen und Anpassungen an den Daten beschrieben.

3.2.1 Betriebliche Daten für Fahrplan und Liniennetz

Grundlage des klassischen Linienverkehrs ist ein Fahrplan (vgl. Reinhardt 2018, S. 456). Dieser bildet darüber hinaus auch den Maßstab für die Arbeit des Leitstellenpersonals.

Ein bekanntes Format zum Austausch von Soll-Fahrplandaten wurde vom Verband Deutscher Verkehrsunternehmen (VDV) mit der VDV-Schrift 452 veröffentlicht. Seitdem gilt VDV 452 als de-facto zum Datenaustausch von Fahrplandaten zwischen verschiedensten Systemen im ÖPNV und ist als Schnittstelle zwischen zahlreichen Systemen auch im produktiven Einsatz.

Ein weiteres Fahrplandatenformat ist das von der EU spezifizierte Format NeTEx. Dabei handelt es sich um XML-Dateien, die Daten gemäß der NeTEx-Spezifizierung enthalten. Bedingt durch den großen Anwendungsraum von NeTEx wurden zwischenzeitlich auch für einige Länder nationale Dialekte, sogenannte Länderprofile, erstellt. Diese Länderprofile ermöglichen wiederum eine Abbildung von länderspezifischen Eigenschaften. Bei NeTEx handelt es sich damit um ein alternatives Format zu VDV 452, welches aber im Regelfall gesondert aus einem System exportiert beziehungsweise in ein weiteres System importiert werden muss.

Seit der Änderung des Personenbeförderungsgesetz sind unter Anderem auch Verkehrsunternehmen dazu verpflichtet, Fahrplan- und Netzdaten öffentlich zur Verfügung zu stellen (vgl. Bundesministerium für Verkehr, §3a). In den meisten Fällen werden die Daten im von Google definierten *General Transfer Feed Specification* (GTFS) Format bereitgestellt. Dabei handelt es sich um CSV-ähnliche Textdateien mit Informationen zu Fahrzeiten, Fahrten, Linien und Haltestellen, wie wiederum in einer ZIP-Datei verteilt werden (vgl. Google 2022). Für betriebliche Anwendungen, zu denen ein ITCS nebst zugehöriger Bordrechner gehört, ist GTFS nur eingeschränkt geeignet, da wichtige Betriebsdaten wie Umläufe in der Spezifikation nicht vorgesehen sind.

Um Zwischenschritte für möglicherweise nötige Datenkonvertierungen zu sparen, wird für den Einsatz in einem produktiven ITCS das Format VDV 452 empfohlen. Für die Verwendung im Prototyp sind GTFS-Daten aber allemal zu gebrauchen. Aufgrund der einfachen Zugänglichkeit und des gut verarbeitbaren Datenformates werden GTFS-Daten als Grundlage für den Prototyp und dessen Evaluation genutzt.

3.2.2 Kartendaten und Routing

An dieser Stelle muss zunächst der spätere Einsatzzweck differenziert werden. Der Fokus liegt auf der Umleitungssuche für Linienbusse im Störfall. Hierbei ist das Routing von essenzieller Bedeutung, es gilt das Routing im Sinne einer Stabilisierung des Betriebes im Störfall zu optimieren. An dieser Stelle bietet sich die Verwendung offener Geodaten an. Diese enthalten nicht nur geographische Informationen, die später wieder für die Anzeige in einer Karte genutzt werden können, sondern bieten darüber hinaus auch die Möglichkeit, mit gängigen Frameworks Informationen und Daten zur weiteren Verarbeitung extrahieren zu können.

Der wohl bekannteste Anbieter für offene Geodaten im deutschsprachigen Raum dürfte OpenStreetMap (OSM) sein. Die Nutzungsbedingungen lassen fast alle Verwendungsarten, darunter auch kommerzielle, zu. Einzige Voraussetzung ist, dass die OSM-Mitwirkenden als Quelle genannt werden. Der Anbieter Geofabrik bietet darüber hinaus täglich aktualisierte, vorab zugeschnittene Datensätze für einzelne Städte, Landkreise oder Bundesländer zum Download als OSM-Datei an. Eine OSM-Datei enthält zunächst jedoch alle auf einem Kartenausschnitt enthaltenen Daten in einer XML-Struktur. Bedingt durch den damit einhergehenden Overhead können selbst Datensätze von kleinen Kartenausschnitten beachtliche Mengen an Daten enthalten, die dann weiterverarbeitet werden müssten. Die Daten in einem OSM-Datensatz sind generell unterteilt in Punkte (Nodes), Wege (Ways) und sogenannte Relationen (Relations). Eine Relation fasst bestehende Punkte und Wege zu neuen Informationen zusammen und baut damit hierarchisch auf dem OSM-Datenmodell auf.

Für das Routing werden nur Straßendaten benötigt, Daten über Häuser und POIs sind nicht notwendig. Eine nennenswerte Alternative zu den fertig verfügbaren Datensätzen der Geofabrik bietet die sogenannte Overpass-API mit dem Webinterface Overpass-Turbo. Mittels einer skriptähnlichen Beschreibungssprache können gezielt Daten selektiert und in alle gängigen Formate exportiert werden. Neben dem Eingabefeld für die Abfrage enthält Overpass-Turbo außerdem eine Vorschau, in der nach Ausführung einer Abfrage die selektierten Daten hervorgehoben werden. Somit ist auf einfachem Weg möglich zu prüfen, ob die Abfrage zum gewünschten Ergebnis führt. Die von Straßendaten lässt sich über das Attribut *highway* eingrenzen, in dem die jeweilige Straßenkategorie abgelegt ist.

Die folgende Tabelle gibt eine Übersicht über die in OSM vergebenen Straßenkategorien:

Attribut (highway=)	Straßenkategorie
motorway / motorway_link	Autobahn
trunk / trunk_link	Schnellstraße
primary / primary_link	Bundesstraße
secondary / secondary_link	Kreisstraße / Landstraße
tertiary / tertiary_link	Vorfahrtstraße (Innerorts)
residential	Innerortsstraße
living_street	Spielstraße
pedestrian	Fußgängerzone

Tabelle 1: Straßenkategorien und deren Schlüssel in OSM-Daten (Quelle: OSM-Highway 2022)

Bei dieser Tabelle handelt sich nicht um eine vollständige Auflistung aller verfügbaren Werte für das Attribut `highway`, sondern ausschließlich um jene Straßenkategorien, die mit Linienbussen befahren werden können. Feldwege, Behelfs- und Privatstraßen sind bewusst nicht berücksichtigt. Spielstraßen und Fußgängerzonen sind deshalb mit inbegriffen, weil diese in einigen Fällen durchaus für den Linienverkehr freigegeben sind. Beispiele sind in nahezu jeder größeren Stadt zu finden. Für Fußgängerzonen wird in OSM-Daten bei einer entsprechenden Freigabe für den Linienverkehr das Attribut `PSV` für *Public Services Vehicle* gesetzt (vgl. OSM-PSV 2022).

Die folgende Abfrage liefert zunächst alle Straßendaten aus dem aktuell angezeigten Bereich, der sogenannten *BoundingBox* in Overpass-Turbo:

```
[out:xml][timeout:120];

(
  way["highway"="motorway"]({{bbox}});
  way["highway"="motorway_link"]({{bbox}});
  way["highway"="trunk"]({{bbox}});
  way["highway"="trunk_link"]({{bbox}});
  way["highway"="primary"]({{bbox}});
  way["highway"="primary_link"]({{bbox}});
  way["highway"="secondary"]({{bbox}});
  way["highway"="secondary_link"]({{bbox}});
  way["highway"="tertiary"]({{bbox}});
  way["highway"="tertiary_link"]({{bbox}});
);

out meta;
>;
out meta qt;
```

Abbildung 7: Overpass-Abfrage für Straßendaten im Primärnetz

Enthalten sind dabei alle Straßen mit einer Kategorie zwischen Autobahn und Innerorts-Vorfahrtstraße. Im weiteren Verlauf wird dieses Straßennetz als *Primärnetz* bezeichnet. Bestätigt wird diese Selektion durch die Vorschau in Overpass-Turbo, wie in der folgenden Abbildung zu sehen ist:

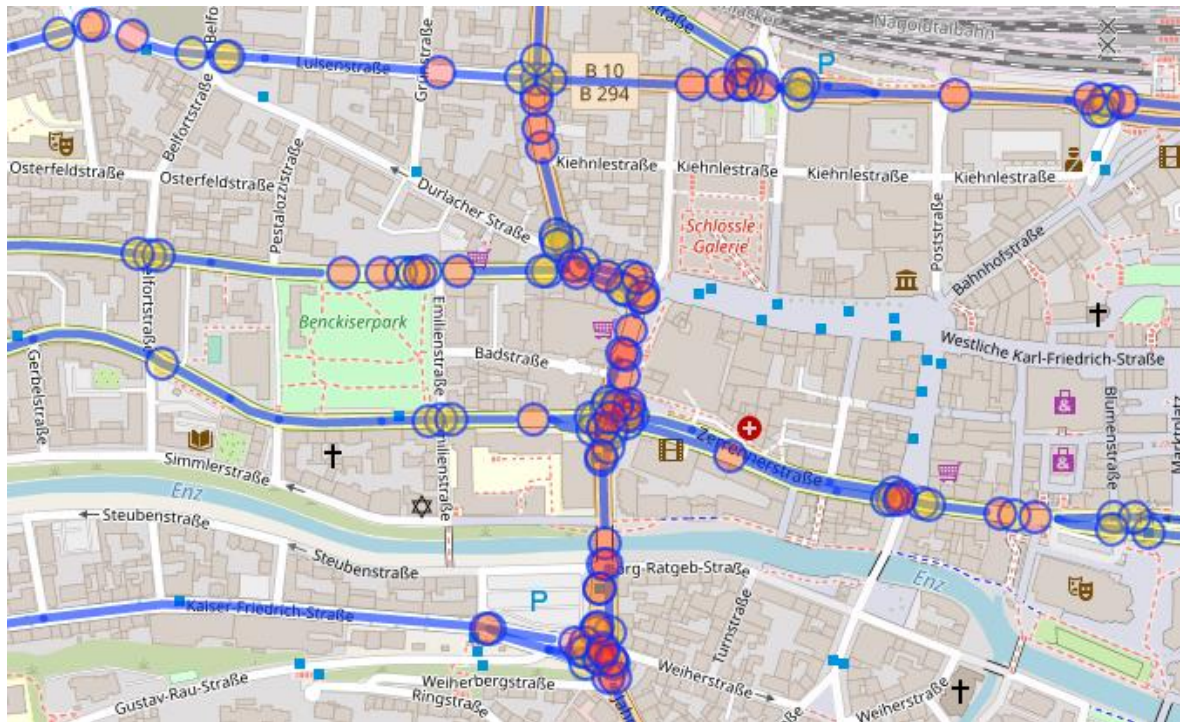


Abbildung 8: Overpass-Vorschau für Straßendatendaten im Primärnetz

In der Auswahl fehlen jedoch wichtige, ebenfalls vom Linienverkehr genutzte Innerortsstraßen und die für den Linienverkehr freigegebene Fußgängerzone. Während letztere durch Einbeziehung des Attributes PSV mit selektiert werden könnte, wird diese Eingrenzung für Innerortsstraßen schwieriger, da diese nicht weiter kategorisiert oder attribuiert sind. Werden also Innerortsstraßen mit in die Abfrage einbezogen, sind automatisch alle Innerortsstraßen enthalten, selbst wenn diese mit einem Linienbus gar nicht befahren werden können. Auch im Routing wäre eine Unterscheidung bestenfalls anhand der verfügbaren Straßenbreite möglich, was jedoch eine adäquate Versorgung der OSM-Daten mit diesen notwendigen Informationen erforderlich macht. Bei offenen, gemeinschaftlich gepflegten Daten aus der OpenStreetMap-Gemeinschaft ist das generell kritisch zu sehen (vgl. Josi 2020, S. 10), daher eignet sich der direkte Export des gesamten Straßennetzes nicht für das zuverlässige Routing eines Linienbusses.

Ziel muss es sein, das Primärnetz generell zu selektieren und alle darunter liegenden Straßenkategorien nur dann mit einzubeziehen, wenn diese bekanntermaßen für Linienbusse befahrbar sind. Diese zusätzlichen Straßen werden fortan unter dem Begriff des *Sekundärnetzes* zusammengefasst. Hierzu bietet sich ein zweistufiges Verfahren an.

In der ersten Stufe werden zunächst Relationen als Selektor gewählt, welche eine Buslinie in den OSM-Daten abbilden und dadurch auf die entsprechenden Wege verweist. Dann werden alle Wege selektiert, die zu diesem festgelegten Selektor passen. Hierdurch sind alle Straßen unabhängig von ihrer Kategorie enthalten, die durch die hinterlegten Linienwege folglich auch mit einem Linienbus befahren werden können. Was jedoch nach wie vor fehlt, sind Innerorts- und Spielstraßen und Fußgängerzonen, welche zwar mit Linienbussen befahrbar sind, jedoch planmäßig nicht befahren werden.

Hier kommt die nächste Stufe ins Spiel, bei der die von den Bussen aufgezeichneten GPS-Trajektorien mit Hilfe eines sogenannten MapMatching-Algorithmus einer entsprechenden Straße im Sekundärnetz zugeordnet werden. Zum MapMatching bietet sich beispielsweise der ST-Algorithmus an, da dieser auch für das Matching von GPS-Trajektorien mit geringerer Samplingrate eine hinreichende Genauigkeit liefert (vgl. Simeonov 2017, S. 21). Dass die GPS-Trajektorien der Busse immer erst am Ende eines Betriebstages und nicht in Echtzeit dem Matching zugeführt werden, trägt der Genauigkeit bei (Simeonov 2017, S. 22). Da bei diesen GPS-Trajektorien auch Leer- und Betriebsfahrten enthalten sind, ergeben sich auf diesem Weg neue Fahrwege auch außerhalb der planmäßigen Linienwege. Anschließend wird die Vereinigungsmenge aus dem bestehenden Primärnetz und dem Sekundärnetz gebildet. Hierdurch erhält man ein vollständiges Straßennetz im Geodatenformat, welche dann von Geoinformationssystemen oder zum Routing weiterverwendet werden kann. Die folgende Abbildung fasst schematisch die Ableitung eines für das Routing von Linienbussen geeigneten Straßennetzes aus OSM-Daten zusammen:

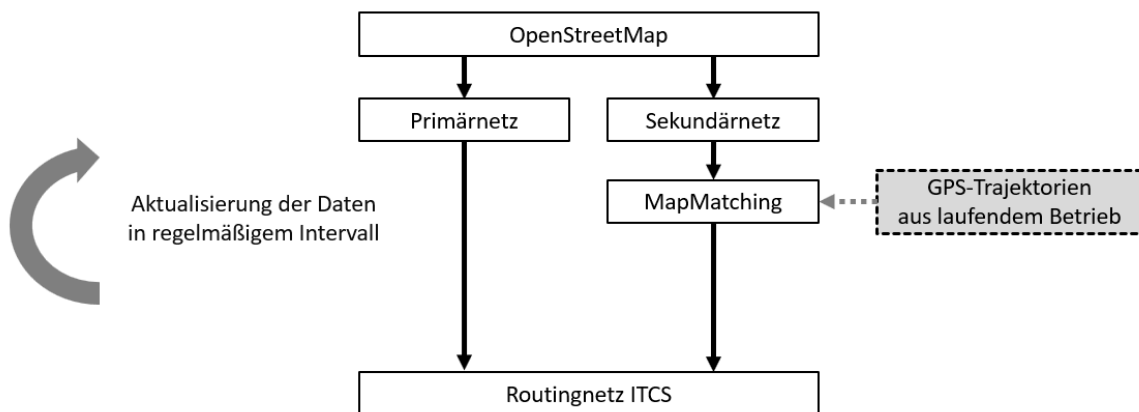


Abbildung 9: Ableitung eines für Linienbusse geeigneten Netzes aus OSM-Daten

Die Aktualisierung des Straßennetzes sollte in regelmäßigen Abständen durchgeführt werden, um stets ein aktuelles Abbild der tatsächlich verfügbaren Verkehrsinfrastruktur vor Ort zur Verfügung zu haben. In einem Produktivsystem könnte die Aktualisierung beispielsweise nachts zur Betriebsruhe durch einen automatischen Hintergrundprozess durchgeführt werden.

3.2.3 Störungsmeldungen und Daten zur Verkehrssituation

Elementar für das automatische Einleiten einer Dispositionsmaßnahme ist für das ITCS die Kenntnis über eine Störung. Anders als bei den Betriebs- und Netzdaten stellt sich die Suche nach geeigneten Datenquellen hier komplexer dar.

Zunächst setzt jede Störungsinformation ein Ereignis mit entsprechender Diagnose voraus (vgl. Schranil 2013, 69 f.). Durch die Kenntnis über das zu Grunde liegende Ereignis können dann die Relevanz eingeschätzt und Prognosen zur Dauer der Störung getroffen werden. Die voraussichtliche Störungsdauer ist spätestens dann wichtig, wenn entschieden werden muss, ob eine einzelne Fahrt umgeleitet werden muss, oder die Störung aller Voraussicht nach beim Eintreffen des Fahrzeuges schon behoben sein wird.

Im Sinne der Digitalisierung von Kommunen kommt damit schnell der Begriff der *Smart City* auf. In diesem Zusammenhang wird regelmäßig auch auf den Verkehrssektor verwiesen (vgl. Soike und Libbe 2018, S. 11; Herzner und Schmidpeter 2022, S. 70). So sind führen Meier und Portmann Lösungsansätze an, die „in Echtzeit flexibel die aktuelle Situation“ (Meier und Portmann 2016, S. 261) am Beispiel von Staus berücksichtigen können. Schaaf und Wilke (2015) konstruieren ein noch umfangreicheres Beispiel, in dem als Folge eines Verkehrsunfalls beispielsweise Rettungskräfte automatisch alarmiert und zur Einsatzstelle geschickt werden und Linienbusse, die durch den Einsatz von Echtzeitdaten selbstständig planmäßige Anschlüsse über die entstehende Verspätung informieren können (vgl. Schaaf und Wilke 2015, S. 563). Alle regelmäßigen ÖV-Nutzenden wissen aus Erfahrung, dass solche Showcase-Beispiele von der Realität der meisten Verkehrsbetriebe und deren Umfeld weit divergieren. Smart City-Lösungen setzen erwartungsgemäß die Aufnahme und Verteilung einer entsprechenden Meldung über ein Ereignis mit einem sichergestellten Wahrheitsgrad voraus. Falschmeldungen könnten insbesondere in Bezug auf das Beispiel mit den Rettungskräften drastische Folgen haben. Ferner ist es für ein flächendeckend funktionierendes System eine hinreichend große Ausstattung mit aktueller Sensorik erforderlich, um notwendige Datenmengen sammeln und in adäquater Latenzzeit verarbeiten zu können (vgl. Meier und Portmann 2016, S. 266). Inwiefern Daten aus vorhandenen Smart City-Plattformen für den Einsatz als Störungsmelder in einem ITCS dienen kann, hängt also maßgeblich vom Umsetzungsgrad entsprechender Systeme ab. Dieser wurde 2018 vom Deutschen Institut für Urbanistik für die 200 Städte mit dem höchsten Bevölkerungsgrad in Deutschland umfangreich erhoben.

Die folgende Grafik zeigt den Umsetzungsgrad verschiedener Smart City-Projekte in diesen Städten:

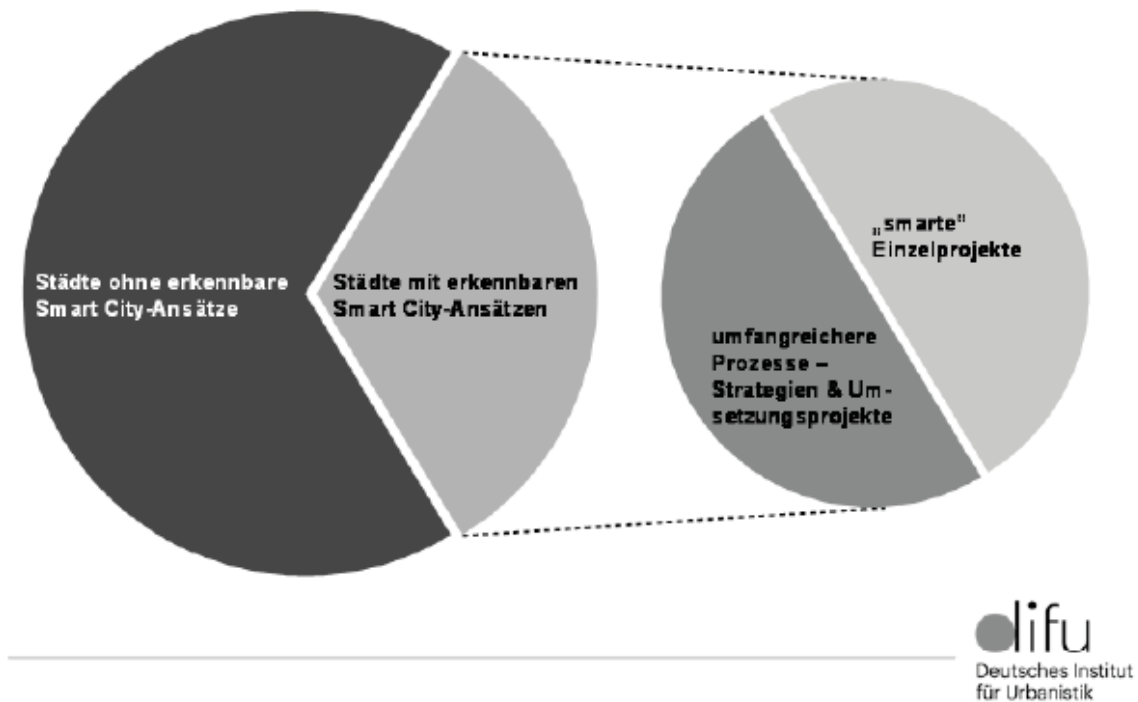


Abbildung 10: Umsetzungsgrad von Smart City Projekten in den 200 größten Städten (Quelle: Soike und Libbe 2018, S. 6)

Es ist zu erkennen, dass gut zwei Drittel aller betrachteten Städte mit Stand 2018 keine Umsetzung von Smart City-Projekten forciert haben. Vom verbleibenden Drittel sind wiederum nur die Hälfte von umfangreicherer Natur und spiegeln konkrete Ansätze zur intermodalen Einbindung in kommunale Dienste vor (vgl. Soike und Libbe 2018, S. 7). Die Stadt Pforzheim belegt dabei immerhin den 63. Platz (vgl. Statistisches Bundesamt 2020; zitiert nach de.statista.com) und bietet nach aktuellem Stand sogar öffentliche Smart City-Projekte für die Bürgerschaft an. Bezug zu Verkehrsstörungen hat jedoch nach aktuellem Stand (2022) keines dieser Projekte. Zusammenfassend lässt sich festhalten, dass die Mehrheit der deutschen Städte bislang keine Bestrebungen zum Ausbau von Smart City-Projekten zeigt und sich die aktiven Städte in der Hauptsache auf Großstädte reduzieren lassen (vgl. Soike und Libbe 2018, S. 8). Beachtet man zusätzlich noch die fehlende, gesellschaftlich durchgesetzte Definition des Begriffes der Smart City (vgl. Herzner und Schmidpeter 2022, S. 67), erschwert dies die Beurteilung des Umsetzungsgrades in deutschen Städten zusätzlich. Smart City-Daten scheiden daher als alleinige Quelle für Störungsinformationen in einem ITCS aus und können bestenfalls als zusätzliche Datenquelle angesehen werden. Die Konzentration auf Großstädte verhindern außerdem eine flächendeckende Nutzung in einem ITCS, welches auch Regionalbusse überwachen soll. Ob in den verteilten Daten Informationen zum zu Grunde liegenden Ereignis

oder alternativ direkt zur voraussichtlichen Störungsdauer enthalten sind, hängt von der konkreten Implementierung einer Smart City-Plattform ab. Es bleibt offen, inwieweit sich weitere Smart City-Projekte in naher Zukunft etablieren und welchen Informationsgehalt diese mitbringen werden.

Ohne die direkte Initiative einer Kommune kommt hingegen das deduktive Monitoring aus. Nach diesem Prinzip ermittelt der Anbieter Google Daten zur aktuellen Verkehrssituation. Hierbei werden anonymisierte Standortdaten von Android-Mobilgeräten ausgewertet und auf das Verkehrsaufkommen umgelegt (vgl. Exner 2012, S. 25). Google stellt diese Daten gegen Entgelt auch über Programmschnittstellen (API) zur Verfügung. Gerade kommerzielle Anbieter mit deduktivem Monitoring in ihrem Portfolio stehen insbesondere wegen datenschutzrechtlicher Belange immer wieder in der Kritik (vgl. Exner 2012, S. 26)

Eine Alternative zum Abgriff von Daten Dritter aus Smart City-Systemen oder Datenbrokern bieten viele ITCS bereits in Form sogenannter *kodierter Meldungen* an (vgl. DELFI 2020, S. 122). Unter diesem zunächst undurchsichtigen Begriff verstehen sich vordefinierte Textnachrichten, welche das Fahrpersonal an die Leitstelle zum Austausch von Informationen senden kann. Der Inhalt dieser Nachrichten kann in jedem System von der Administration festgelegt werden, sodass nahezu alle Formen von Nachrichten ausgetauscht werden können. Gängige Beispiele sind in der folgenden Abbildung zu sehen:

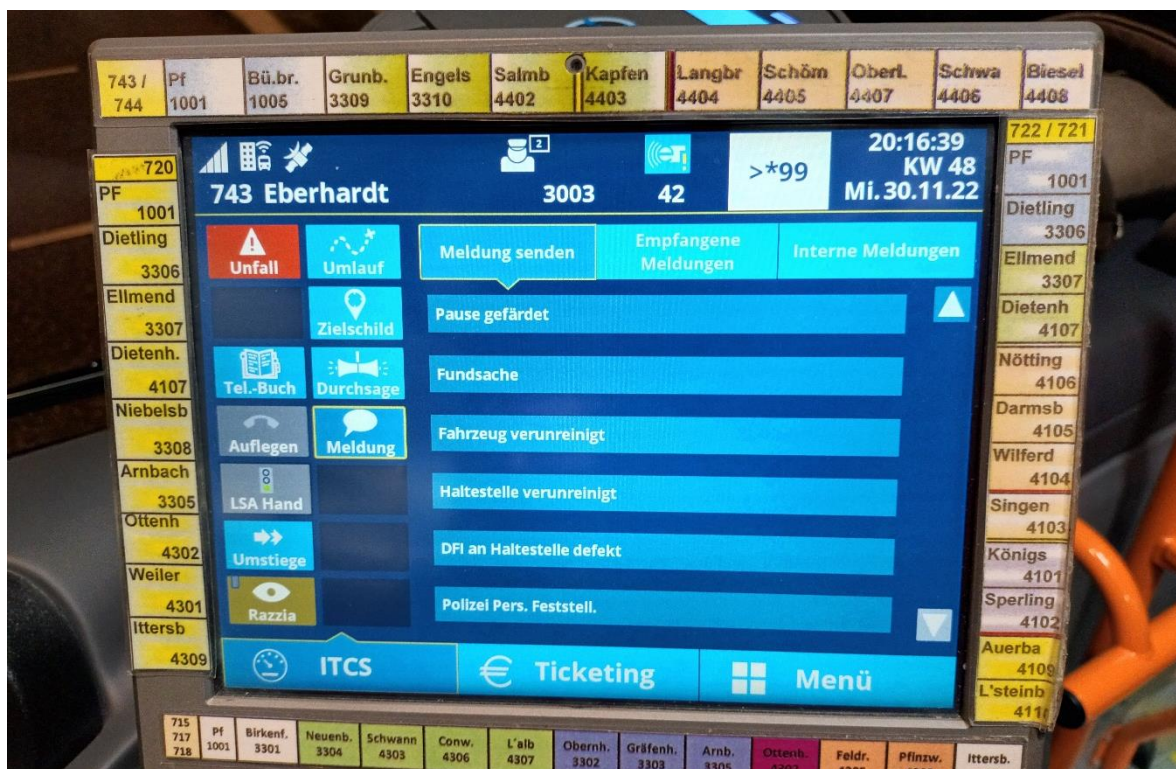


Abbildung 11: Vordefinierte Meldungen auf einem Bordrechner in einem Linienbus

Anstatt in solch einem Fall eine Sprechfunk- oder Telefonverbindung zur Leitstelle herzustellen, genügt es beim Eintreten eines entsprechenden Ereignisses eine entsprechende Meldung abzusetzen, um die Leitstelle über einen bestimmten Zustand zu informieren. Dem Leitstellenpersonal wird die Meldung dann zusammen mit Informationen zur Linie, Verspätungslage und Standort und anderen betrieblich relevanten Daten angezeigt. Diese Funktion ist für Meldungen vom Fahrpersonal an die Leitstelle initiiert, die „keine zeitsynchrone Bearbeitung“ erfordern und um den Sprechfunk zu entlasten (vgl. VDV-Schrift 730, S. 56). Es stellt sich also die Frage, wie diese Informationen aus erster Hand gewinnbringend als Datenquelle für eine automatische Anordnung von Umleitungen eingesetzt werden können.

Es sind zumindest folgende Gesichtspunkte in Betracht zu ziehen:

- **Richtigkeit:** Meldungen können auch versehentlich abgesetzt werden. Gründe hierfür können von einer ein Versehen, eine fehlerhafte Unterweisung oder aber auch eine Fehlbedingung aufgrund von Sprachbarrieren des Fahrpersonals sein. In jedem Fall muss also die Richtigkeit der Meldung sichergestellt werden, um darauf basierend weitere Entscheidungen zu treffen.
- **Genauigkeit** Es müssen so viele Ereignisse wie möglich als Meldung hinterlegt werden, um möglichst viele Details zur Störung herleiten zu können. Zeitgleich darf das Fahrpersonal aber auch nicht mit Informationen überfrachtet werden, da andernfalls die Richtigkeit und Qualität der Meldungen leiden könnte. Für die richtige Bedienung sind Einfachheit, Verständlichkeit und Intuitivität essenziell.
- **Dauer:** Die Dauer der Störung oder zumindest eine Prognose der Dauer ist nötig, um festzulegen, welche Fahrten von der Störung betroffen sein werden. Ebenso ist es nötig, ein eventuell früheres Störungsende erkennen zu können, um den Regelbetrieb gegebenenfalls wieder früher herstellen zu können. Auf die Notwendigkeit zusätzlicher Handlungen des Fahrpersonals sollte hierbei verzichtet werden, da die Erkennung eines früheren Störungsendes sonst im Zweifel bei Unterlassen dieser Handlung nicht möglich ist.

Eine mögliche Umsetzungskonzeption soll in Bezug zum Stadt-Szenario aus Kapitel 3.1 wir nachfolgend vorgestellt. Hierzu seien folgende Annahmen getroffen:

Die erste Fahrt, welche von Westen kommend in den gesperrten Abschnitt einfährt, kommt wenige Fahrzeuge hinter der Unfallstelle zum Stehen. Nachdem das Fahrpersonal die Situation überblickt hat, setzt es eine entsprechende Meldung vom Typ „Polizeieinsatz“ ab. Die Folgefahrt nähert sich im zeitlichen Abstand von 10min von hinten an.

Nach dem Eintreffen der Meldung geht das ITCS schrittweise folgendermaßen vor:

1. Zunächst werden – falls verfügbar – weitere Quellen hinsichtlich einer ähnlichen Störungsmeldung bezogen auf Ort und Zeit durchsucht. Gibt es an dieser Stelle bereits signifikante Übereinstimmungen, ermittelt das ITCS die voraussichtlich betroffenen Fahrten und ordnet für diese eine geeignete Umleitung an.
2. Sind hingegen keine weiteren Datenquellen verfügbar, überwacht das ITCS zunächst die Position des Fahrzeuges, von dem die Meldung abgesetzt wurde. Da von einer Streckensperrung ausgegangen wird, darf sich die Position des Fahrzeuges im Überwachungszeitraum von beispielsweise einer Minute nicht mehr signifikant verändern. Ist diese Bedingung erfüllt, ermittelt das ITCS die voraussichtlich betroffenen Fahrten und ordnet für diese eine geeignete Umleitung an.

Die folgende Abbildung zeigt die Validierung einer Meldung als Flussdiagramm:

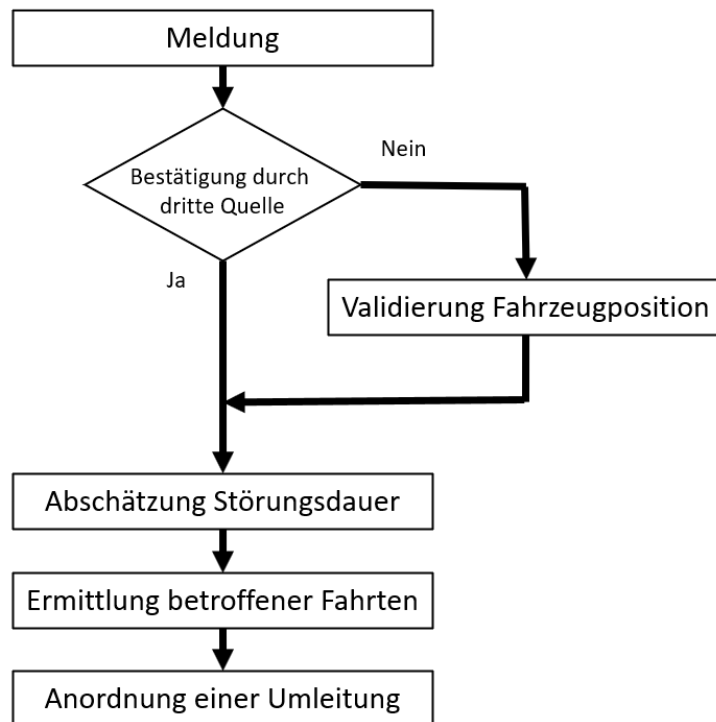


Abbildung 12: Validierung einer systemseitigen Meldung als Flussdiagramm

Nachdem entweder die statistisch betrachtet zu erwartende Störungsdauer erreicht ist oder alternativ eine Positionsänderung des ersten Fahrzeuges zu verzeichnen ist, geht das ITCS folgendermaßen vor:

1. Das erste betroffene Fahrzeug wird erneut hinsichtlich der GPS-Position überwacht. Ist hier noch immer keine signifikante Positionsänderung zu verzeichnen, wird auch für die folgenden Fahrten eine Umleitung angeordnet.
2. Kann hingegen eine Fortbewegung des Fahrzeuges festgestellt werden, wird die Umleitung für Fahrzeuge, die sich bereits auf der Umleitungsstrecke befinden, beibehalten und für alle anderen Fahrten zurückgenommen. Eine gesonderte Handlung des Fahrpersonals ist damit nicht erforderlich.

Bezüglich der Genauigkeit bietet es sich an, Störungsursachen aus der Vergangenheit zu analysieren und in Klassen einzuordnen. Oft kristallisieren sich dabei einige wenige Störungsursachen heraus, welche dann dem Fahrpersonal gezielt zur Auswahl angeboten werden können (vgl. Schranil 2013, S. 188). Die Störungsdauer lässt sich basierend auf statistischen Verfahren ebenfalls aus Störungsdaten aus der Vergangenheit abschätzen (vgl. Schranil 2013, S. 193). Einzige Voraussetzung hierzu ist, dass vergangene Störungen in ausreichendem Maß dokumentiert wurden. Für die Richtigkeit, Genauigkeit und Abschätzung der Dauer ist damit systemseitig gesorgt. Wurde eine Meldung aus dem ITCS selbst erfolgreich validiert, kann diese Störungsmeldung auch über Datendrehscheiben an weitere ITCS von anderen Verkehrsunternehmen kommuniziert werden. Hierzu eignet sich der SIXI-SX-Dienst. Mit Hilfe dieses Dienstes lassen sich innerhalb einer Störungsmeldung auch direkt Haltestellen oder Linien referenzieren (vgl. VDV-Schrift 736-2, S. 34), sodass die Informationen in einer Störungsmeldung übertragbar zwischen verschiedenen ITCS sind.

Ohne die Verfügbarkeit weiterer Datenquellen mit Detailinformationen, beispielsweise zu gesperrten Fahrtrichtungen, muss entweder grundsätzlich von einer Vollsperrung ausgegangen oder eine Fahrt pro Richtung eine entsprechende Meldung absetzen. Andernfalls würden Umleitungen für Fahrten angeordnet, die betrieblich betrachtet gar nicht umgeleitet werden müssen. Als weitere Datenquelle kann beispielsweise auch ein manuelles Eingreifen des Leitstellenpersonals angesehen werden. An dieser Stelle muss im Einzelfall ein betrieblich vertretbarer Trade-Off durch entsprechende Systemkonfiguration gefunden werden.

3.3 Auswahl geeigneter RL-Algorithmen

In Kapitel 2.6.3 wurden wichtige Algorithmen aus dem Bereich des RL vorgestellt. Wie die folgende Abbildung zeigt, handelt es sich jedoch nur um einen kleinen Teil der erforschten oder bekannten RL-Algorithmen:

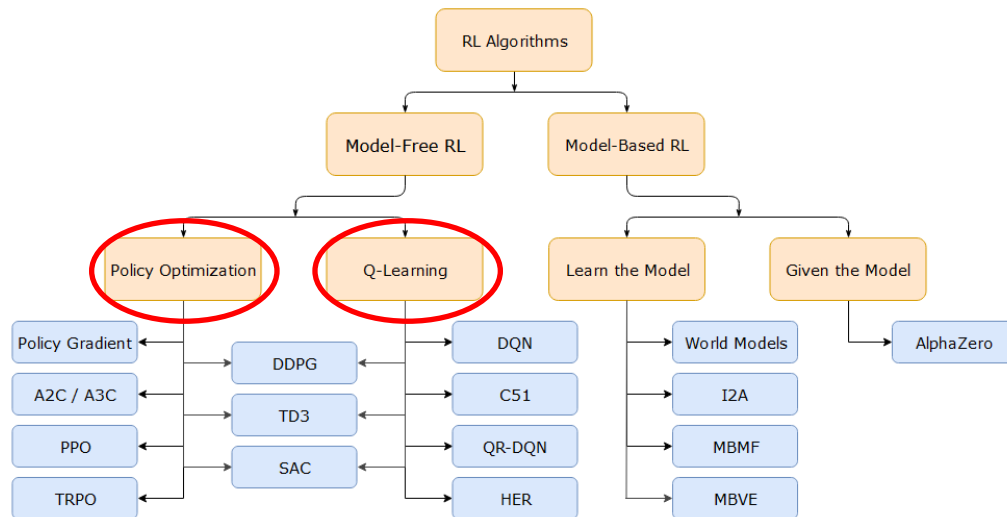


Abbildung 13: Bekannte RL-Algorithmen (Quelle: OpenAI 2018)

Alle bisher vorgestellten Algorithmen sind der Klasse der modellfreien Algorithmen zuzuordnen. Problematisch ist grundsätzlich bei modellfreien Algorithmen, dass sie ausschließlich zahlenbasiert auf der Gewinnfunktion arbeiten. Modellbasierte Algorithmen sind hingegen in der Lage, auch eine Vielzahl im Modell berücksichtigter Informationen aus der Umwelt zu berücksichtigen. Einbußen sind dafür insbesondere bei der Laufzeitkomplexität modellbasierter Algorithmen zu verzeichnen (vgl. Pong et al. 2018, 1 f.). Modellbasierte Algorithmen bringen außerdem ein zusätzliches Maß an Komplexität mit, da die Umwelt zuerst in einem Modell zusammengefasst werden muss. Fragwürdig ist an dieser Stelle, inwieweit ein Gesamtvorteil für die ursprüngliche Problemstellung, die Umleitungssuche für Linienbusse, durch Nutzung eines modellbasierten Algorithmus erreicht werden kann. Diese Frage ist jedoch nicht Teil der vorliegenden Arbeit. Vielmehr liegt der Fokus auf dem Vergleich von On- und Off-Policy-Algorithmen, die auch in **Abbildung 13** rot markiert sind. Bei beiden Varianten handelt es sich um modellfreie Algorithmen. Zur Implementierung und zum anschließenden Vergleich der Performance und erzielten Ergebnisse gilt es nun, zwei der vorgestellten Algorithmen auszuwählen.

Zur besseren Übersicht werden die vorgestellten Algorithmen aus Kapitel 2.6.3 in der folgenden Grafik nochmals zusammenfassend nach On- und Off-Policy, sowie nach MC- und TD-Algorithmen differenziert:

On-Policy	REINFORCEMENT SARSA	SARSA
Off-Policy		Q-Learning Deep Q-Learning
	MC-Algorithmen	TD-Algorithmen

Abbildung 14: Einordnung der vorgestellten RL-Algorithmen

An dieser Stelle ist anzumerken, dass es sich auch hierbei keineswegs um eine annähernd vollständige Auflistung aller Subvarianten dieser Algorithmen handelt. Zu nahezu jede der vorgestellten Algorithmen wurden bereits eine Vielzahl von Kombinationen mit anderen Strategien oder statistischen Verfahren erforscht. Der Algorithmus REINFORCE steht als On-Policy-Algorithmus klar dem Q-Learning und der daran ansetzenden Variante des Deep Q-Learning als Off-Policy-Algorithmus gegenüber. SARSA nimmt aufgrund seiner in Kapitel 2.6.3.3 näher erläuterten Möglichkeit zur Einbindung einer Aktionshistorie eine Sonderrolle ein und kann sowohl als TD-, als auch als MC-Algorithmus implementiert werden.

Diese Sonderrolle macht SARSA zu einem interessanten Algorithmus im Hinblick auf den Einsatz in einem Betriebsleitsystem. Q-Learning ist als ältester und zeitgleich wohl auch am weitest entwickelter, modellfreier RL-Algorithmus prädestiniert für einen Vergleich. Durch die Verwendung eines tiefen KNN zur Funktionsapproximation werden beide Algorithmen in einem vergleichbaren Rahmen im Prototyp implementiert. Im weiteren Verlauf der Arbeit sollen also die Algorithmen

Deep SARSA und Deep Q-Learning

in einem Prototyp zur Betriebslenkung von Linienbussen im Störfalls eingesetzt und hinsichtlich ihrer Ergebnisse und Performance miteinander verglichen.

3.4 Modellierung der Umwelt zur Simulation

Zur Implementierung des Prototyps ist zunächst notwendig, die Umgebung zu modellieren. Mit Modellierung im engeren Sinne ist hier die Überführung aller relevanten Bedingungen aus der Praxis in eine allgemeine, mathematisch verwertbare Form gemeint. Diese Transformation komplexer, reeller Eigenschaften einer Umgebung in eine vereinfachte mathematische Darstellung entspricht per Definition einem mathematischen Modell (vgl. Rellensmann 2019, S. 5; Nahrstedt 2012, S. 201). Zwar wurden im vorhergehenden Unterkapitel explizit modellfreie Algorithmen für die weitere Betrachtung ausgewählt, das Training erfolgt für gewöhnlich jedoch nicht in einer reellen Umgebung, sondern in einer Simulation derselben (vgl. Huber 2018, S. 23). Zunächst muss also die Umgebung modelliert werden.

3.4.1 Anlehnung an Markov-Entscheidungsprozesse

Zur Modellierung der Simulationsumgebung für den Agenten eignet sich ein MEP (vgl. Lorenz 2020, S. 15; Witt 2019, S. 7). Sie werden durch das Tupel (S, A, p, r, γ) beschrieben, wobei

- S die Menge der möglichen Zustände
- A die Menge der möglichen Aktionen
- p die Übergangswahrscheinlichkeitsfunktion
- r die Gewinnfunktion
- und γ der Diskontierungsfaktor

sind.

Der Zustand der Umgebung ergibt sich aus den in den OSM-Daten enthaltenen Wegen. Ein Zustand wird durch die Fahrzeugposition und die Befahrbarkeit der Wege charakterisiert. Die Fahrzeugposition entspricht dabei einfach einem Weg, die Befahrbarkeit wird durch ein binäres Attribut ausgedrückt. In der Praxis entspräche dies dem aktuell befahrenen Streckenabschnitt eines Fahrzeuges und der Kenntnis über eventuell gesperrte Abschnitte. In Kapitel 2.6 wurden ausgehend von einem Streckennetz von 1245 Wegen etwa

$$|W| + 2^{|W|} = 1245 + 2^{1245} = 1245 + 1,071509 \cdot 10^{301}$$

theoretisch mögliche Zustände abgeleitet. In der Praxis dürfte die relevante Zustandsmenge S jedoch viel kleiner sein, da die theoretische Annahme darauf beruht, dass auch keiner der Wege, oder im Gegenzug aber auch alle Wege in ihrer Befahrbarkeit eingeschränkt sein könnten. In der Realität werden es eher 0 bis 25 Wege sein, die in ihrer Befahrbarkeit zeitgleich eingeschränkt sind, wobei die möglichen Dimensionen hier natürlich von der Größe des vorgehaltenen Netzes abhängt.

Innerhalb der Zustandsmenge S kann weiter nach verschiedenen Zustandstypen differenziert werden. Folgende Zustandstypen lassen sich direkt ableiten:

- *Startzustände* $S_S \subseteq S$ sind Zustände, in denen sich das Fahrzeug auf dem regulären Fahrweg befindet und mindestens ein Streckenabschnitt auf dem Regelfahrweg nicht befahrbar ist
- *Abweichungszustände* $S_D \subseteq S$ sind Zustände, in denen sich das Fahrzeug nicht auf dem Regelfahrweg befindet
- *Terminalzustände* $S_T \subseteq S$ sind alle Zustände, bei denen das Fahrzeug durch Übergang aus einem Abweichungszustand wieder den Regelfahrweg erreicht

Voraussetzung für die Entstehung möglicher Startzustände ist, dass ein Streckenabschnitt nicht befahrbar ist. Andernfalls wäre keine Umleitung des Linienbusses und folglich auch kein Startzustand notwendig. Ebenso kann ein Abweichungszustand nur von einem Startzustand aus erreicht werden, da das Fahrzeug andernfalls auf dem Regelfahrweg bleiben würde. Bei den Terminalzuständen gilt zusätzlich die Einschränkung, dass der Streckenabschnitt, an dem das Fahrzeug wieder seinen Regelfahrweg erreicht, geographisch betrachtet nach dem nicht befahrbaren Streckenabschnitt liegen muss. Andernfalls könnte der Agent bereits nach zwei Schritten einen Terminalzustand erreichen, wenn er zunächst in einen Abweichungszustand und von dort aus direkt wieder in einen Terminalzustand übergeht. Sofern jener Terminalzustand aber das Anfahren des gesperrten Streckenabschnittes nicht verhindert, müsste der Agent spätestens an der Stelle erneut umgeleitet werden, was im Widerspruch zum Erreichen praxistauglicher Ergebnisse steht. Alle Aktionen, die von einem bestimmten Startzustand $s_s \in S_S$ zu einem Terminalzustand $s_t \in S_T$ führen, bilden zusammen eine Episode (vgl. Lorenz 2020, S. 16).

Außerdem dürfen Zustände, bei denen das Fahrzeug sich auf einem als gesperrt gekennzeichneten Abschnitt befindet, nicht erreicht werden. Das würde per Definition bereits ausgeschlossen, da ein gesperrter Abschnitt nicht befahren werden kann. Zustände, auf die diese Kriterien zutreffen, können aus der Zustandsmenge S exkludiert werden. In einer Implementierung käme dies einer temporären Löschung des gesperrten Streckenabschnitts aus dem Streckennetz gleich, da dieser folglich auch nicht mehr befahren werden könnte.

Zur Verdeutlichung der Bedeutung der unterschiedlichen Zustandsarten wird in den folgenden Abbildungen das Stadt-Szenario in der Rohdatenansicht der OSM-Daten gezeigt. Zu sehen sind anstelle der gewöhnlichen Kartenansicht die einzelnen Wege in den OSM-Daten als blaue Linien. In Gelb sind die Knotenpunkte dargestellt, an denen zwei oder mehr Wege miteinander verbunden sind. Zur besseren Übersicht ist die Abbildung auch im Anhang nochmals zu finden.

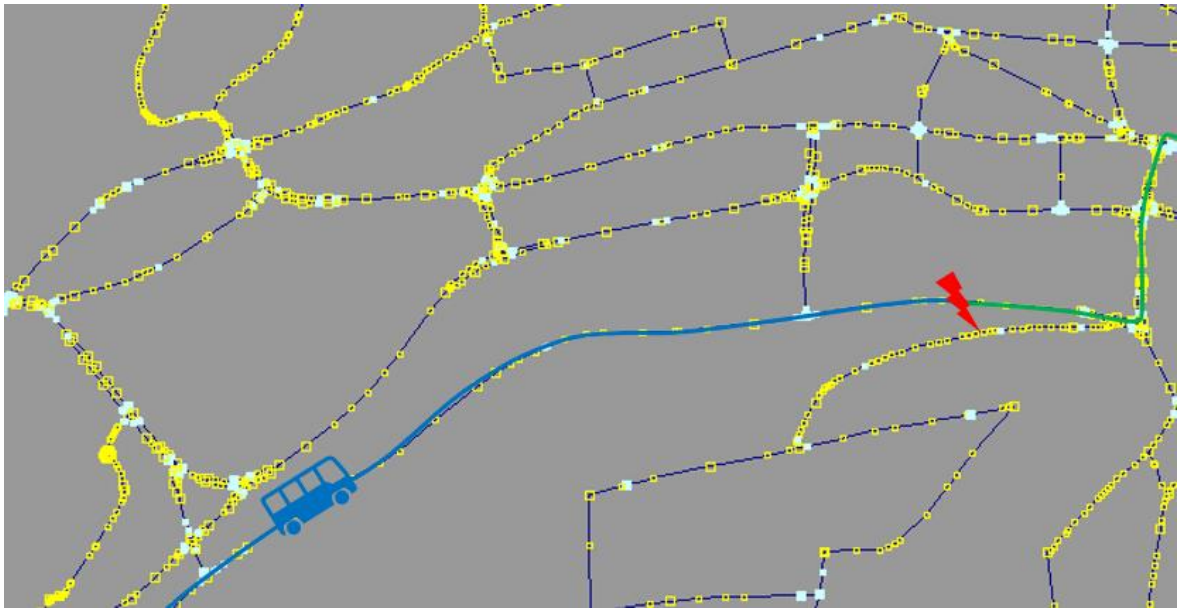


Abbildung 15: Geographische Darstellung von Start- und Terminalzuständen

Wie auch in der Kartenansicht zeigt der rote Blitz den unterbrochenen Streckenabschnitt. Der blaue Bus im Westen markiert die aktuelle Fahrzeugposition. Die dickere Linie stellt den Regelfahrweg dar, wobei die Startzustände in blau und Terminalzustände in grün ausgeführt sind. Auf die gesonderte Darstellung der Verbindungsstellen wurde verzichtet. Alle übrigen, nicht gesondert hervorgehobenen Wege bilden folglich die Abweichungszustände. Aus dieser geographischen Darstellung ist gut zu erkennen, dass der Agent zunächst von einem Startzustand in mindestens einen Abweichungszustand übergehen muss, um schließlich einen Terminalzustand zu erreichen und dabei den gesperrten Streckenabschnitt zu umfahren.

Neben Zuständen erfordert ein MEP auch die Definition von Aktionen. Mögliche Aktionen sind in der Menge A zusammengefasst. Eine Aktion beschreibt den Übergang von einem beliebigen Zustand s_t in einen Folgezustand s_{t+1} , wobei die Menge der tatsächlich zur Verfügung stehenden Aktionen $A_s \subseteq A$ vom aktuellen Zustand s_t der Umgebung abhängt. Ein Zustand ist dabei ausschließlich abhängig vom vorhergehenden Zustand und der darin gewählten Aktion. Folgezustände oder weitere Vorgängerzustände haben keinen Einfluss auf den Zustand, der durch die Auswahl einer Aktion erreicht wird. Diese Bedingung entspricht der Markov-Eigenschaft (vgl. Witt 2019, S. 7) und ist Voraussetzung dafür, dass überhaupt von einem MEP gesprochen werden kann. Letztendlich ermöglicht erst die Annahme eines MEP eine derartige Vereinfachung der Realität im Modell (vgl. Witt 2019, S. 7). In Worten ausgedrückt würde eine Aktion also beispielsweise „jetzt in die XYZ-Straße einbiegen“ lauten. Der erreichte Streckenabschnitt ergibt sich erst aus der Aktion und dem letzten befahrenen Abschnitt, also dem letzten Zustand. Ob der Streckenabschnitt dabei von Westen oder von Osten aus kommend erreicht wurde oder welcher Abschnitt als nächstes folgt, ist unerheblich. Die Anzahl verfügbarer Aktionen entspricht der Anzahl der Wege, da jeder Weg eine Aktion hat, mit der

er vom Agenten erreicht werden kann. Lediglich die Wege, die per Definition nicht erreicht werden können, da dies zu einem unzulässigen Zustand führen würde, benötigen streng genommen keine Aktionen. Es gilt daher näherungsweise

$$|A| \approx |W|$$

Die Übergangswahrscheinlichkeitsfunktion p beschreibt die Wahrscheinlichkeit ausgehend von einem Zustand s_t unter Anwendung einer Aktion a_t einen Zustand s_{t+1} überzugehen als

$$p(s_{t+1}, r_{t+1} | s_t, a_t)$$

wobei r_{t+1} der erwartete Gewinn nach Übergang in den Zustand s_{t+1} ist (vgl. Witt 2019, S. 7). Unterschiedliche Übergangswahrscheinlichkeiten können entstehen, da ein Zustand jeweils immer von der aktuellen Fahrzeugposition und der Befahrbarkeit der Strecken abhängt. Durch Ausführen einer Aktion wechselt das Fahrzeug seine Position im Netz, es kommt daher in jedem Fall zu einem Zustandsübergang. Während der Folgezustand in den meisten Fällen einfach durch die geänderte Fahrzeugposition ausgemacht wird, könnte es sein, dass sich mit demselben Zeitschritt auch die Befahrbarkeit eines Streckenabschnittes ändert. In diesem Fall käme es zu einem Übergang in einen anderen Zustand. Der erwartete Gewinn wird wiederum definiert durch die Gewinnfunktion $r(s_t, s_{t+1})$, die den Gewinn beim Übergang von einem Zustand in den nächsten definiert.

Zur Angabe der Gewinnfunktion ist es zunächst notwendig, sich mit relevanten Zustandsübergängen vertraut zu machen. Von Interesse sind besonders jene Zustandsübergänge, die zu einer messbaren Veränderung des laufenden Betriebes führen. Das können sowohl negative als auch positive Veränderungen sein. Folgende Fälle sind theoretisch möglich:

- Das Fahrzeug, im RL der Agent, geht von einem möglichen Startzustand $s_t \in S_S$ in den nächsten möglichen Startzustand $s_{t+1} \in S_S$ über; der Agent erhält dafür einen positiven Gewinn, da der Linienweg um einen weiteren Zeitschritt beibehalten wurde
- Das Fahrzeug verlässt den regulären Fahrweg aus einem Startzustand $s_t \in S_S$; der Agent erhält einen negativen Gewinn, da das Verlassen des Linienweges grundsätzlich erst einmal negativ zu beurteilen ist
- Das Fahrzeug geht von einem Abweichungszustand $s_t \in S_D$ in einen anderen Abweichungszustand $s_{t+1} \in S_D$ über; der Agent erhält dafür weder einen negativen noch einen positiven Gewinn, da er den Linienweg weder verlassen noch wieder erreicht hat
- Das Fahrzeug erreicht aus einem Abweichungszustand $s_t \in S_D$ einen Terminalzustand $s_t \in S_T$; der Agent erhält dafür einen positiven Gewinn, da der Linienweg wieder erreicht wurde

Zusätzlich kann eine weitere Besonderheit berücksichtigt werden. Befindet sich im erreichten Abweichungszustand $s_{t+1} \in S_D$ eine Haltestelle, deren Luftlinienabstand einen bestimmten Grenzwert zu einer Haltestelle des regulären Fahrwegs unterschreitet, erhält der Agent dafür einen positiven Gewinn. So wird erreicht, dass das Fahrzeug – sofern vorhanden – entlang möglicher Ersatzhaltestellen und nicht auf beliebigem Weg umgeleitet wird.

Ziel des Agenten muss es sein, jeweils das Maximum an Gewinn bis zum Erreichen eines Terminalzustands zu erhalten (vgl. Mainzer 2019, S. 119). Dabei werden ausgehend von einem Zustand $s_t \in S$ alle möglichen Aktionen und Folgezustände rekursiv betrachtet, bis ein Terminalzustand $s_t \in S_T$ erreicht wird. Jener Aktionspfad mit dem höchsten erwarteten Gewinn wird dann zur Bewertung eines Zustands oder einer Aktion ausgehend von einem Zustand in der State-Value-Funktion herangezogen. Der Gewinn berechnet sich durch Addition der einzelnen Gewinne nach jedem Zustandsübergang (vgl. Witt 2019, S. 8) als

$$R = r(s_t, s_{t+1}) + \gamma r(s_{t+1}, s_{t+2}) + \gamma^2 r(s_{t+2}, s_{t+3}) + \dots + \gamma^k r(s_{t+k}, s_{t+k+1}) = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, s_{t+k+1})$$

Durch festlegen des Diskontierungsfaktors γ auf $0 < \gamma < 1$ wird erreicht, dass zeitlich nähere Aktionen mit positivem Gewinn stärker gewichtet werden, als Aktionen mit positivem Gewinn, die in größerem zeitlichen Abstand liegen (vgl. Witt 2019, S. 8; Dammann o.D., S. 4). Zur Veranschaulichung stelle man sich vor, dass es für eine Streckensperrung eine mögliche Umleitung auf direktem, dafür etwas langsameren Weg und eine weiträumigere, dafür vermeintlich schnellere Umleitung gibt. Durch den Diskontierungsfaktor wird sichergestellt, dass der Agent stets den kürzesten Weg wählt, da der weitere Weg zwar möglicherweise einen höheren erwarteten Gewinn aufweist, dafür aber durch den Diskontierungsfaktor nicht mehr so stark ins Gewicht fällt. Ein Diskontierungsfaktor von 1 bedeutet entsprechend, dass alle Wege gleich gewichtet werden, ein Diskontierungsfaktor nahe 0 würde hingegen fast ausschließlich kurze Wege zulassen. In der Praxis muss über den Diskontierungsfaktor ein geeigneter Trade-Off zwischen möglichst schnellem Erreichen des regulären Linienweges und einer möglicherweise stabileren, dafür umfangreicheren Umleitung gefunden werden.

Die *Bellmann-Eigenschaft* besagt, dass die Lösung eines Problems dann optimal ist, wenn die Lösung aller Teilprobleme optimal ist (vgl. Schmitz 2017, S. 14). Umgekehrt bedeutet das also, dass eine optimale Lösung dann erreicht werden kann, wenn auf dem Weg zur Gesamtlösung stets optimale Teillösungen gewählt werden. Ausgehend von den bisherigen Annahmen kann also zu jeder Aktion ausgehend von einem bestimmten Zustand ein skalarer Wert ermittelt werden. Man spricht vom sogenannten Q-Wert, in diesem Fall ein *State-Action-Wert*, der für das Q-Learning definiert ist als

$$Q(s, a) = r + \gamma * \max_a Q(s_{t+1}, a)$$

wobei r der bisher erreichte Gewinn ist und über zum Q-Learning gehörende Strategiefunktion $\max_a Q(s_{t+1}, a)$ stets die Aktion ausgewählt wird, deren Q-Wert bezogen auf den Folgezustand am höchsten ist.

Für SARSA ist die Q-Funktion als

$$Q(s, a) = r + \gamma * Q(s_{t+1}, a_{t+1})$$

Der Unterschied liegt in der Strategiefunktion und in der Betrachtung des Folgezustandes. Abweichend vom Q-Learning kann die Auswahl der Aktion auch mit einer anderen Funktion als Greedy ausgewählt werden. Dadurch wird ausschließlich die tatsächlich gewählte Aktion berücksichtigt, selbst dann, wenn es möglicherweise eine Aktion mit einem höheren State-Action-Wert gibt. Dadurch ist ein gewisses Maß an Exploration im SARSA-Algorithmus sichergestellt.

Durch diese Grundkonstruktion wird erreicht, dass der final trainierte Agent, respektive das Fahrzeug

- den Linienweg so lange wie möglich beibehält und nur dann verlässt, wenn es keine andere Möglichkeit mehr gibt
- so schnell wie möglich versucht, den Linienweg wieder zu erreichen, wenn der Linienweg zuvor verlassen wurde
- dabei möglichst weitere, potenzielle Ersatzhaltestellen in der Routenplanung berücksichtigt

3.4.2 Bedeutung von erwartetem und erreichtem Gewinn

Die Anlehnung an einen MEP ausgehend von der Erfüllung der Markov-Eigenschaft machte eine einfach definierte Gewinnfunktion möglich. In der bisherigen Definition der Gewinnfunktion ist allerdings nur der *erwartete* Gewinn berücksichtigt. Dabei nehmen deterministische Kennwerte Einfluss auf den Wert der Gewinnfunktion, erst im Nachhinein erfahrene Verschlechterungen beispielsweise durch Erreichen eines Verspätungsgrenzwertes oder das Verpassen planmäßiger Anschlüsse sind dabei jedoch nicht erfasst. Diese und vergleichbare Faktoren führen zu einer Minderung des *erreichten* Gewinns.

Ähnlich wie das Bedienen von Ersatzhaltestellen können solche Ereignisse bei der Bewertung bei einem Zustandsübergang als Faktoren berücksichtigt werden, die den erreichten Gewinn entsprechend verkleinern. Allerdings stehen diese erst nach Abschluss einer Episode endgültig fest. Weiter muss berücksichtigt werden, dass diese Einflüsse in einer Simulation, die zum

Training des Agenten genutzt wird, nur dann auftreten können, wenn sie auch realitätsnah modelliert wurden. Andernfalls können diese Einflüsse höchstens abgeschätzt werden, eine Anpassung an die reelle Umgebung erfolgt aber erst beim Einsatz des Systems in einer solchen und auch nur dann, wenn reelle Erfahrungswerte berücksichtigt werden. Das ist bei Off-Policy-Algorithmen nur durch Erweiterungen der Fall, bei On-Policy-Algorithmen passiert diese Anpassung durch fortlaufende Exploration automatisch (vgl. Lorenz 2020, S. 61).

Nicht zuletzt ist daher spannend, ob es im Anwendungsfall dieser Arbeit signifikante Qualitätsunterschiede zwischen Deep SARSA als On-Policy-Algorithmus und DQN als Off-Policy-Algorithmus gibt. In diesem Fall könnte davon ausgegangen werden, dass der tatsächlich erreichte Gewinn einen erheblichen Einfluss auf die Qualität in einer realitätsnahen Implementierung des Systems hat.

3.4.3 Approximation der Value-Funktion durch neuronale Netze

Anstelle eines linearen Funktionsapproximators kommen bei Deep SARSA und DQN jeweils ein KNN als Funktionsapproximator zum Einsatz. KNN zählen zu den nichtlinearen Funktionsapproximatoren (vgl. Schmitz 2017, S. 20) und tauchen als Begriff besonders oft im Zusammenhang mit ML und KI auf. Die notwendige Bewertungsfunktion wurde in Kapitel 3.4.1 bereits definiert und ist die einzig notwendige Zusatzinformation (vgl. Huber 2018, S. 23). In diesem Kapitel wird nun kurz die Funktionsweise eines KNN betrachtet und eine Verbindung zur bisherigen Modellierung hergestellt.

Ein KNN ist ein mathematisches Konstrukt bestehend aus Knoten, sogenannten Neuronen, die in Schichten angeordnet sind. Jedes der Neuronen kann eine Information erhalten, entsprechend gewichten und an ein anderes Neuron über eine Verbindung weiterreichen. Die erste Schicht, welche die Eingangsdaten entgegennimmt, wird auch als *Eingangsschicht* bezeichnet. Die letzte Schicht, an der die Ergebnisse des KNN abgefragt werden können, wird *Ausgangsschicht* genannt. Zwischen Ein- und Ausgangsschicht können beliebig viele weitere Schichten liegen. Da diese Schichten nach außen hin nicht in Erscheinung treten, werden sie als *verdeckte Schichten* bezeichnet (vgl. Hildebrand 2018, S. 9). Ein KNN, welches Daten ausschließlich aus der Eingangsschicht kommend in Richtung der Ausgangsschicht verarbeiten kann, wird als *Feedforward-Netzwerk* bezeichnet (vgl. Kerkhoff 2020, S. 17) und bildet die „grundlegende Architektur künstlicher neuronaler Netze“ (Terhag 2018, S. 16). Als Funktionsapproximator nimmt das KNN den aktuellen Umgebungszustand entgegen und liefert an der Ausgangsschicht die Bewertung für alle Aktionen ausgehend vom Umgebungszustand (vgl. Lüth 2019, S. 9).

Der aktuelle Zustand wird durch einen Vektor $\vec{x} = (x_0, \dots, x_{d_1}) \in \{0, 1\}^{d_1}$ mit binären Werten an die Eingangsschicht übergeben, wobei $d = 2 * |W|$ die Größe des Vektors \vec{x} ist. Diese ergibt sich aus der zweifachen Anzahl der verfügbaren Wege im Streckennetz. Die Größe ergibt sich aus der Notwendigkeit, zum einen die Fahrzeugposition und zum anderen die Befahrbarkeit aller Wege im Streckennetz zu repräsentieren. In der ersten Hälfte von \vec{x} darf maximal eine Komponente den Wert 1 annehmen, sie repräsentiert die Fahrzeugposition auf einem Weg. In der zweiten Hälfte von x nimmt jede Komponente den Wert 1 an, es sei denn, der repräsentierte Weg ist nicht befahrbar. Die folgende Abbildung verdeutlicht dieses Schema nochmals graphisch:

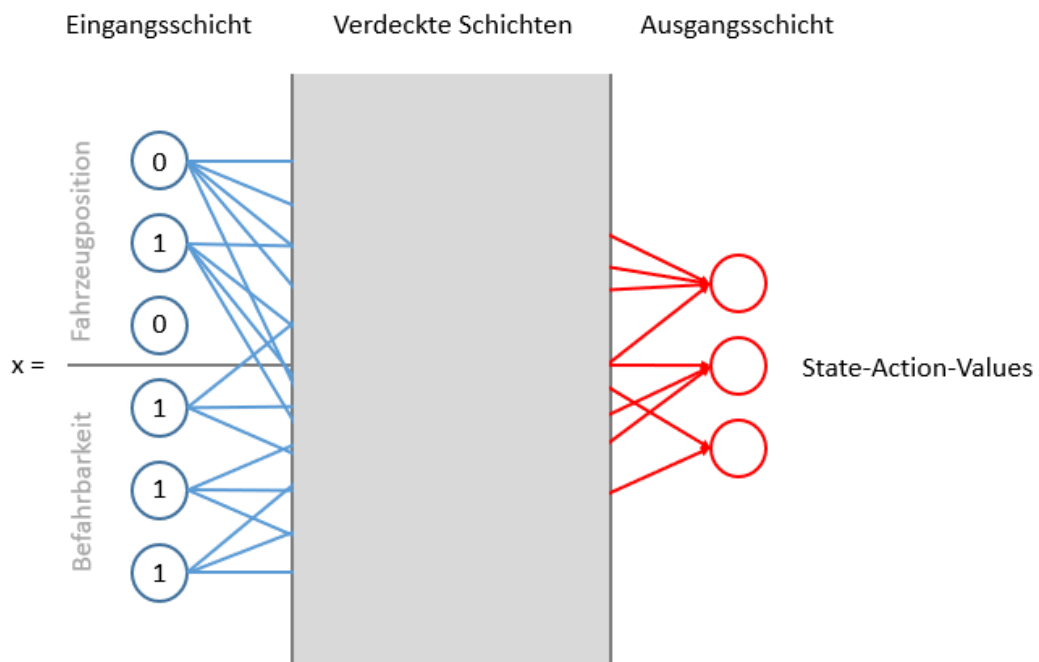


Abbildung 16: Schematische Darstellung der Ein- und Ausgangsdaten im KNN

Zur besseren Übersicht ist diese Darstellung auf drei Wege reduziert. Zu erkennen ist die Angabe der aktuellen Fahrzeugposition. Außerdem sind alle Wege befahrbar. Die Ausgangsschicht liefert die State-Action-Values für jede Aktion $a \in A$ ausgehend vom aktuellen Zustand $s_t \in S$.

Zwischen den einzelnen Neuronen werden die Informationen verschieden gewichtet. Die einzelnen Gewichtungen werden in einer Gewichtungsmatrix $w_{i,j}^u$ abgelegt und nach der Übergabe der Informationen an die nächste Schicht im folgenden Neuron angewandt. Der Index u gibt dabei an, für welche Schicht die Gewichtungsmatrix gilt. Aufbauend auf **Abbildung 16** zeigt die folgende Abbildung die Einbindung der Gewichtungsmatrix $w_{i,j}^u$ im KNN:

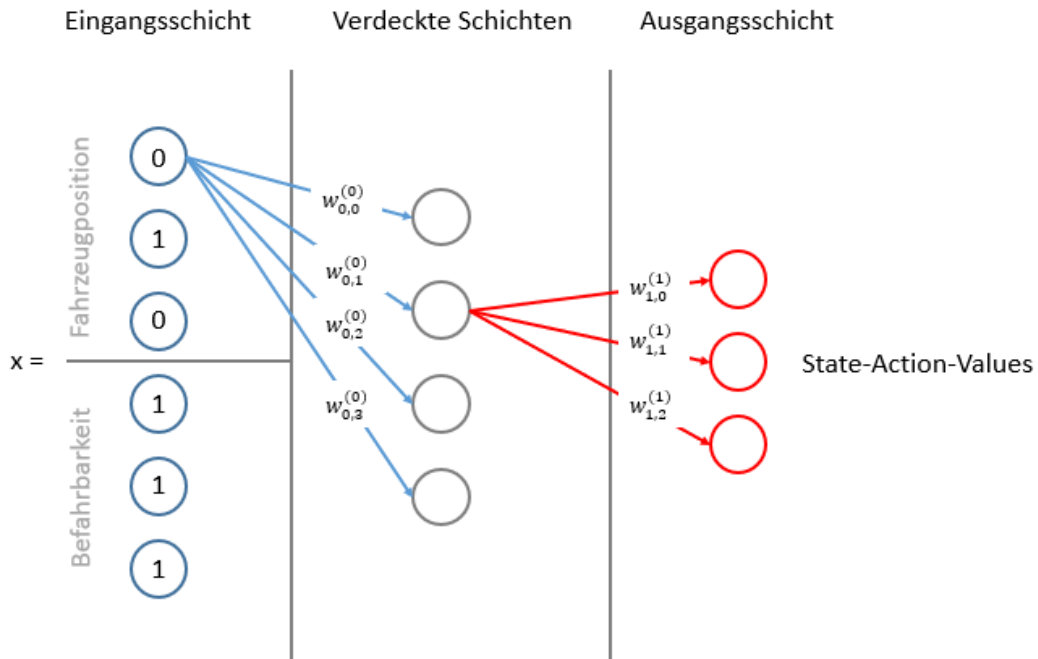


Abbildung 17: Einbindung der Gewichtungsmatrizen in einem KNN

Innerhalb eines Neurons wird die Eingabe mit der entsprechenden Gewichtung aus $w_{i,j}^u$ und dem Bias aus dem Bias-Vektor \vec{b}_j^u der entsprechenden Schicht zur sogenannten *Netzeingabe* verrechnet (vgl. Kerkhoff 2020, S. 17). Die Netzeingabe ist definiert als

$$net_j^u = \sum_{i=0}^d w_{i,j}^{u-1} x_i + b_j^u$$

Bei späterer Anwendung des RL werden Gewichtungsmatrix $w_{i,j}^u$ und Bias-Vektor \vec{b}_j^u zu einem Parametervektor $\vec{\theta}_j^u$ vereinigt (vgl. Kerkhoff 2020, S. 21).

Zur Berechnung des Ausgangswertes o_j^u eines Neurons wird anschließend die Aktivierungsfunktion ϕ^u auf die Netzeingabe angewandt. Es gilt daher

$$o_j^u = \phi^u(net_j^u)$$

Über die Aktivierungsfunktion wird geregelt, wann ein Neuron in einem KNN aktiv wird, also basierend auf einer Eingabe eine Ausgabe liefert und welchen Wert diese Ausgabe annehmen kann. Dieser Prozess wird als *Forward Propagation* bezeichnet. Die folgende Abbildung zeigt den Ablauf innerhalb eines Neurons graphisch:

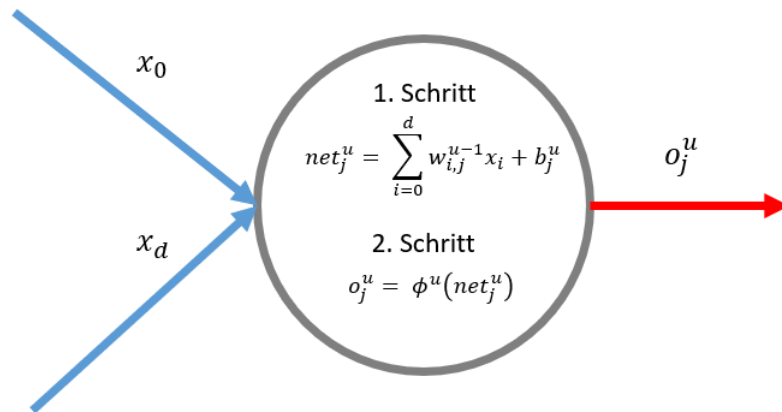


Abbildung 18: Schematische Darstellung der Berechnungsschritte innerhalb eines Neurons

Die Rolle der Aktivierungsfunktion kann theoretisch von jeder beliebigen mathematischen Funktion eingenommen werden. Vielfach haben sich jedoch nichtlineare Funktionen durchgesetzt, die durch ihr nichtlineares Verhalten für eine signifikante Veränderung der Ausgabe sorgt, wenn die Eingabe einen gewissen Grenzwert erreicht (vgl. Lorenz 2020, S. 123). Altbewährte Vertreter der Aktivierungsfunktionen sind die *Sigmoid*- und die *Tangens-Hyperbolicus*-Funktion, auch als *TanH*-Funktion bezeichnet.

Die folgende Abbildung zeigt die Funktionsgraphen der Sigmoid- und TanH-Funktion:

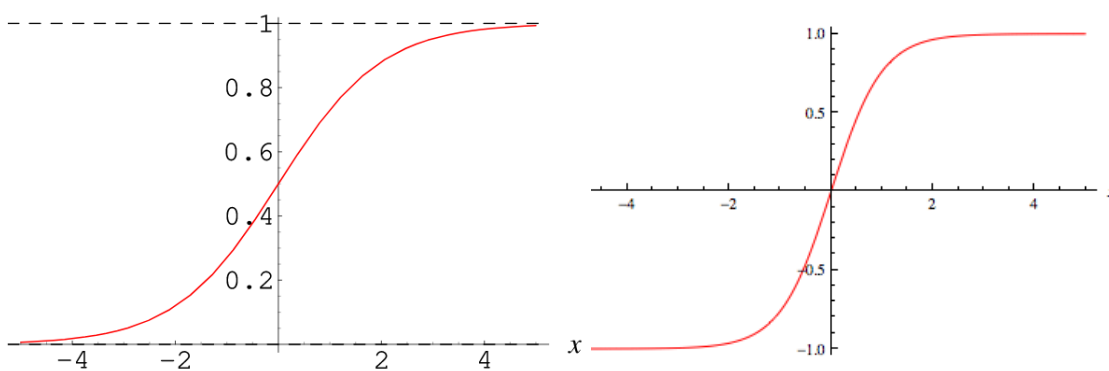


Abbildung 19: Funktionsgraphen der Sigmoid- und TanH-Funktion

Auf den ersten Blick sehen beide Graphen zunächst sehr ähnlich aus. Bei genauerer Betrachtung fallen jedoch zwei Dinge auf, dass die Sigmoid-Funktion zum einen nicht nullzentriert ist, das bedeutet für einen x-Wert von null verläuft die Funktion nicht durch den Ursprung. Die TanH-

Funktion ist hingegen nullzentriert. Zum anderen fällt auf, dass beide Funktionen für x -Werte außerhalb des Intervalls $[-1; 1]$ in Sättigung gehen. Sie liefern dann annäherungsweise immer denselben Wert, es kommt zu keiner signifikanten Änderung mehr. Die Ableitung beider Funktionen wird strebt in diesem Fall gegen null. Für das Training basierend auf Gradienten wird jedoch gerade die Ableitung wichtig (vgl. Ebert 2019, 30 ff.; Mishra et al. 2019, S. 875), sodass sowohl die Sigmoid- als auch die TanH-Funktion nur für x -Werte im Intervall $[-1; 1]$ nutzbar sind.

Eine Alternative zu den beiden gezeigten Aktivierungsfunktionen ist die *Rectified Linear Unit*-Funktion, auch *ReLU*-Funktion genannt. Sie wird in KNN verwendet, welche eine Vielzahl von Features verarbeiten, ist jedoch wenig geeignet für Datensätze mit komplexen, nichtlinearen Zusammenhängen (vgl. Mishra et al. 2019, S. 874). Die folgende Abbildung zeigt den Funktionsgraphen der ReLU-Funktion und der *Leaky ReLU*-Funktion als Abwandlung der ReLU-Funktion:

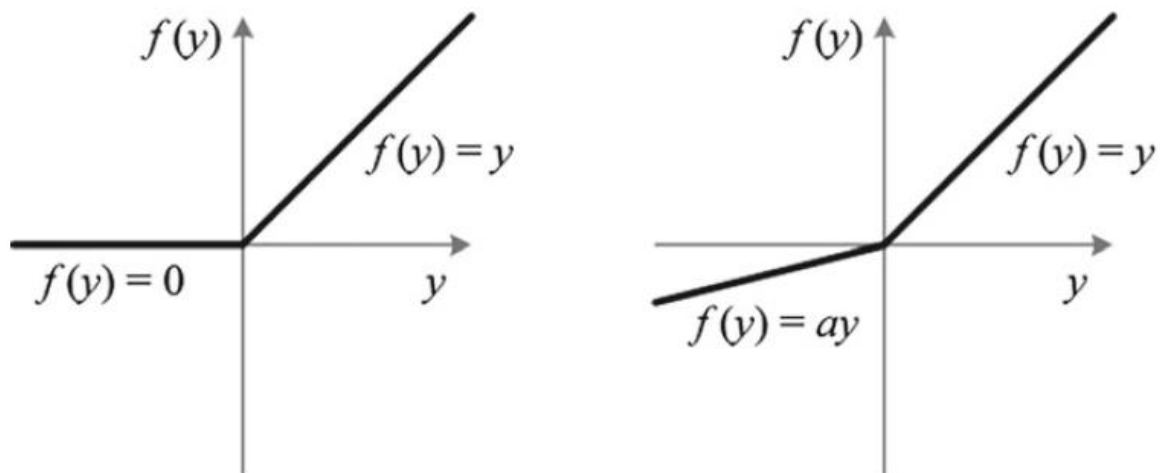


Abbildung 20: Funktionsgraphen der ReLU- und Leaky ReLU-Funktion (Quelle: Mishra et al. 2019, S. 875)

Durch Anwendung der ReLU-Funktion entspricht die Ausgabe exakt dem Wert der Aktivierung, solange $x > 0$ gilt. Für $x \leq 0$ liefert ReLU ausschließlich null als Ergebnis. Durch besonders kleine oder negative Eingangswerte neigen Neuronen bei Anwendung von ReLU als Aktivierungsfunktion dazu, im KNN nicht mehr aktiviert zu werden (vgl. Mishra et al. 2019, S. 874). Abhilfe schafft an dieser Stelle eine Abwandlung von ReLU, die Leaky ReLU-Funktion. Diese liefert für $x > 0$ denselben Wert wie auch die ReLU-Funktion. Für Werte $x \leq 0$ wird der Eingangswert mit einer weiteren linearen Komponente α zwar anders gewichtet als ein positiver Eingangswert, es kommt jedoch dennoch zu einer Aktivierung des Neurons, wodurch das Neuron weiter einen Beitrag zum Trainingsprozess leistet.

Im Gegensatz zur Sigmoid- und TanH-Funktion bringt ReLU außerdem den Vorteil mit sich, dass keine komplexen Rechenoperationen wie Exponentialrechnungen notwendig sind.

Dadurch wird das Training erheblich ressourcenschonender. Zusätzlich bleibt ReLU für $x > 0$ vollständig differenzierbar, sodass gradientenbasierte Lernverfahren angewandt werden können. Bei Leaky ReLU ist die Differenzierbarkeit auch für $x \leq 0$ gesichert. Aus diesem Grund wird in dieser Arbeit, sofern nicht zwingend anders erforderlich, Leaky ReLU als Aktivierungsfunktion genutzt.

Aus der Definition nach Kerkhoff (2020) geht hervor, dass jede Schicht eine eigene Aktivierungsfunktion besitzen kann. Folglich ist es also möglich, verschiedene Schichten mit verschiedenen Funktionen zu aktivieren. Genutzt wird dies beispielsweise dann, wenn die Ausgangsschicht zwingend einen Ergebnisvektor mit Werten im Intervall $[0; 1]$ liefern muss.

3.4.4 Trainingsprozess mit neuronalen Netzen im RL

Der Lernprozess eines KNN besteht in der fortlaufenden Anpassung dieser Gewichtungen und Parameter zwischen den einzelnen Neuronen (vgl. Kerkhoff 2020, S. 19). Typischerweise werden KNN im überwachten Lernen genutzt (vgl. Kerkhoff 2020, 18 f.) und durch einen Trainingsdatensatz angepasst. Der vielfach verwendete *Backpropagation* Algorithmus verwendet eine Fehlerfunktion, die eine Differenz zwischen der letzten Ausgabe des KNN und dem Label aus den Trainingsdaten berechnet (vgl. Schmitz 2017, S. 21). In vielen Fällen wird der *mittlere quadratische Fehler* als Fehlerfunktion verwendet (vgl. Terhag 2018, S. 21; Hildebrand 2018, S. 12; Ebert 2019, S. 38). Zum Training wird die Fehlerfunktion nach jedem einzelnen Parameter partiell differenziert und ein Gradientenvektor berechnet. Der Gradient zeigt dann genau in die Richtung der größten Steigung der Fehlerfunktion. Ziel ist es, die Fehlerfunktion zu minimieren (vgl. Hildebrand 2018, S. 12). An dieser Stelle wird auch deutlich, warum die Differenzierbarkeit der Aktivierungsfunktion von solcher Bedeutung ist. Möglich wird dieses Verfahren erst, weil die Ergebnisse des KNN auf einer Verkettung differenzierbarer Funktionen besteht (vgl. Terhag 2018, S. 21). Anschließend werden die Parameter rekursiv durch die Regel

$$\vec{\theta} \leftarrow \vec{\theta} - \eta * \nabla L$$

aktualisiert, wobei ∇L der Gradient der Fehlerfunktion und η die Lernrate des KNN ist. Die Lernrate gibt an, wie stark der Einfluss des Gradienten ist (vgl. Hildebrand 2018, S. 12). Eine zu große Lernrate lässt den Fehler den Fehlerwert im schlechtesten Fall immer größer werden, eine zu kleine Lernrate kann hingegen zum Auffinden eines lokalen Minimums der Verlustfunktion führen (vgl. Ebert 2019, S. 33). Mit dem nächsten Datensatz beginnt dieser Trainingsprozess von vorne, bis der Fehler hinreichend klein ist.

Da im RL das Training jedoch nicht auf gelabelten Trainingsdaten, sondern auf einem Trial-And-Error Verfahren aufbaut, muss eine Lösung zum Training des KNN innerhalb dieses Trial-

And-Error Verfahrens gefunden werden. Anstelle des Trainingsdatensatzes könnte beispielsweise die Q-Funktion eingesetzt werden, um ein überwachtes Lernen mit einem KNN im RL möglich zu machen.

Zu Beginn wird dem KNN als Eingabe ein bestimmter Zustand $s_t \in S$ übergeben. Als Ausgabe des KNN erhält man dann die Q-Werte aller Aktionen $a \in A$. Anschließend wird die Differenz zwischen den erhaltenen Q-Werten und tatsächlichen Werten der Q-Funktion berechnet und zur Backpropagation verwendet. Dieser Prozess wird solange wiederholt, bis das KNN für die Eingabe eines Zustandes näherungsweise die Werte der Q-Funktion erreichen (vgl. Lorenz 2020, S. 146). Da der Parametervektor des KNN vor Beginn des Trainings zufällig initialisiert wird, ergeben sich in den ersten Iterationen stark differierende Werte, ehe durch die schrittweise Anpassung des Parametervektors schließlich eine Konvergenz der durch das KNN approximierten Funktion erreicht wird. Nach Abschluss des Trainings muss lediglich der Parametervektor $\vec{\theta}_j^u$ anstelle einer vollständigen Bewertungstabelle gespeichert werden. Auf Basis des Parametervektors lässt sich das KNN dann als Schätzer für die State-Value-Funktion nutzen. Die Ausgabe des KNN kann im Anschluss von einer Strategiefunktion verwendet werden, um möglichst günstige Aktionen auszuwählen.

Der Unterschied zwischen DQN und Deep SARSA ergibt sich durch Betrachtung der beiden Pseudocodes an den markierten Stellen:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
 Initialize S
 Loop for each step of episode:
 Choose A from S using policy derived from Q (e.g., ε -greedy)
 Take action A , observe R, S'
 $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
 $S \leftarrow S'$
 until S is terminal

Abbildung 21: Pseudocode des Q-Learning (Quelle: Sutton und Barto 2018, S. 131)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$ 
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$ 
Loop for each episode:
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
  Loop for each step of episode:
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A';$ 
  until  $S$  is terminal
```

Abbildung 22: Pseudocode des SARSA-Algorithmus (Quelle: Sutton und Barto 2018, S. 130)

Wie in Kapitel 3.4.1 bereits erläutert, besteht der wesentliche Unterschied zwischen Q-Learning und SARSA im Vorhandensein einer separaten Strategiefunktion und in der Berücksichtigung der tatsächlich gewählten Aktion (vgl. Sutton und Barto 2018, 130 f.). Das Training im KNN erfolgt bei Deep SARSA genauso wie für DQN, das Ergebnis ist jedoch unterschiedlich zu interpretieren. Bei Deep Q-Learning wird eine State-Action-Value Funktion, bei Deep SARSA direkt eine Strategiefunktion approximiert.

3.4.5 Ziel- und Strategienetzwerk

Bereits in Kapitel 2.6.3.2 wurden mögliche Fehlanpassungen verursacht durch Korrelationen in den Eingangsdaten angesprochen. Da die durch das KNN errechneten Q-Werte zudem auch direkt vom Parametervektor $\vec{\theta}$ abhängen, kann es zu unerwünschten Oszillationen der Werte in $\vec{\theta}$ kommen (vgl. Kerkhoff 2020, S. 25). Folglich kann es dazu kommen, dass die approximierte Funktion divergiert.

In DQN wird das umgangen durch den Einsatz zweier verschiedener Netze. Diese werden auch als *Ziel- und Strategienetzwerk* bezeichnet. Dabei werden zwei in ihrer Architektur identische KNN verwendet, die jedoch mit zwei zunächst gleichen Parametervektoren $\vec{\theta}$ und $\vec{\theta}'$ arbeiten. Im Training, welches im Strategienetzwerk durchgeführt wird, wird der Parametervektor $\vec{\theta}$ mit jedem Aktualisierungsschritt angepasst. In einem vorher fest definierten zeitlichen Abstand werden die Komponenten von $\vec{\theta}$ in den Parametervektor $\vec{\theta}'$ und damit in das Zielnetzwerk kopiert. Nach Abschluss des Trainings wird nur noch das Zielnetzwerk genutzt (vgl. Kerkhoff 2020, S. 25; Larsson 2018, 19 f.).

3.4.6 Gesamtarchitektur

Aufbauend auf den Entscheidungen der bisherigen Unterkapitel zur theoretischen Modellierung wird in diesem Kapitel zusammenfassend die Gesamtarchitektur in einer möglichen Systemumgebung beschrieben.

4 Literaturverzeichnis

Alaliyat, Saleh (2022): Video -based Fall Detection in Elderly's Houses.

Arnold, Jakob (2021): Reinforcement Learning für ein Umplanungsproblem im Scheduling. Masterarbeit. Karl-Franzens-Universität Graz, Graz. Institut für Operations und Information Systems. Online verfügbar unter <https://unipub.uni-graz.at/obvugrhs/download/pdf/6690076?originalFilename=true>, zuletzt geprüft am 11.12.2022.

Arruda, Carlos E.; Moraes, Pedro F.; Agoulmine, Nazim; Martins, Joberto S. B. (2020): Enhanced Pub/Sub Communications for Massive IoT Traffic with SARSA Reinforcement Learning.

Böhm, Robin (2016): Aktionen autonomer Systeme als Elemente relationaler nebenläufiger Markov-Entscheidungsprozesse. Diplomarbeit. Universität Stuttgart, Stuttgart. Institut für Parallele und Verteilte Systeme. Online verfügbar unter <https://elib.uni-stuttgart.de/handle/11682/9652>, zuletzt geprüft am 14.11.2022.

Brezina, Tadej; Emberger, Günter; Rollinger, Wolfgang (2012): Vom Beschwerde- zum Anregungsmanagement im Österreichischen öffentlichen Verkehr. Online verfügbar unter https://d1wqtxts1xzle7.cloudfront.net/34231246/brezina-emberger-2012_anregungsmanagement-with-cover-page-v2.pdf?Expires=1668671995&Signature=DBnAREvXAnDgUHEoCOKurfqbkrq7bAri9Iuu3tJrb8wVptyn2a2WV8EhtbrPKyE3ES1YLBv5TUbvl2BvXzbHPT2B42AGGf6dF77CsQ9eJM9vJmE7AzKyH0RIZSuRTNuZacyvi-Jix~X2h8-Qzn0OAa5kk-9Ff66q6HYuiqRAc501ywUSy~gicq7au5rPkI0udvjb4jiAEJTgT69xwoUoLFyQoWI~KU2tw1azHEtkNXEQB0ekoGP0cpeSo9NHmKWGx5cNz72OatR0tYZBo2xGN~WMFWuMvPkywHiKUo2

EqqvLPCgk8bZrhXMzv4SWybMJv3QDpOhkgX9AcD2v63Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA, zuletzt geprüft am 17.11.2022.

Bundesministerium für Verkehr: Personenbeförderungsgesetz. PBefG. Online verfügbar unter <https://www.gesetze-im-internet.de/pbefg/index.html#BJNR002410961BJNE001802305>, zuletzt geprüft am 18.11.2022.

Chan, Leong; Hogaboam, Liliya; Cao, Renzhi (2022): Applied Artificial Intelligence in Business. Cham: Springer International Publishing.

Dammann, Patrick (o.D.): Einführung in das Reinforcement Learning. Heidelberg Collaboratory for Image Processing (HCI). Heidelberg. Online verfügbar unter https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/541645681/dammann_reinforcement-learning-report.pdf, zuletzt geprüft am 24.11.2022.

DELFI (2020): Entwicklung von Instrumenten zur Umsetzung der delegierten Verordnung (EU) 2017/1926 (ÖV-Daten für NAP). Schlussbericht zum FE-Projekt: 70.0946. Verein zur Förderung einer durchgängigen elektronischen Fahrgastinformation – DELFI e.V. Frankfurt a.M. Online verfügbar unter https://fops.de/wp-content/uploads/2020/09/70.0946_Schlussber.OeV-Daten-f.d.NAP_.pdf, zuletzt geprüft am 24.11.2022.

Ebert, David (2019): Bildklassifizierung mit Neuronalen Netzen. Masterarbeit. Hochschule Merseburg, Merseburg. Online verfügbar unter https://opendata.uni-halle.de/bitstream/1981185920/14186/1/EbertDavid_Bildklassifizierung_mit_neuronalen_Netzen.pdf, zuletzt geprüft am 16.12.2022.

Engfer, Andreas (2002): Fuzzy Logik. Ausarbeitung zur Vorlesung Ausarbeitung zur Vorlesung „Methoden der Künstlichen Intelligenz“. Ausarbeitung. Fachhochschule Furtwangen, Furtwangen. Fachbereich Wirtschaftsinformatik.

Exner, Jan-Philipp (2012): SURVEY BEFORE PLAN 2.0. Neue Ansätze von Raumsensorik und Monitoring für die Raumplanung. In: *PLANERIN* (5/12), S. 24–26.

Findl, Renate; Dahmen-Zimmer, Katharina; Kostka, Markus; Zimmer, Alf (2022): Nutzerorientierte Systementwicklung für den ÖPNV. Paper. Universität Regensburg, Regensburg. Lehrstuhl für Experimentelle und Angewandte Psychologie. Online verfügbar unter <https://psydok.psycharchives.de/jspui/bitstream/20.500.11780/3433/1/findl.pdf>, zuletzt geprüft am 17.11.2022.

Gass, Saul I.; Fu, Michael (Hg.) (2013): Encyclopedia of operations research and management science. Third edition. New York, NY: Springer Science + Business Media (Springer Reference).

Google (2022): GTFS-Static Überblick. Hg. v. Google. Online verfügbar unter <https://developers.google.com/transit/gtfs?hl=de>, zuletzt aktualisiert am 06.10.2022, zuletzt geprüft am 18.11.2022.

Herzner, Alexander; Schmidpeter, René (Hg.) (2022): CSR in Süddeutschland. Unternehmerischer Erfolg und Nachhaltigkeit im Einklang. Springer-Verlag GmbH. Berlin, Heidelberg: Springer Gabler (Management-Reihe Corporate Social Responsibility). Online verfügbar unter <http://www.springer.com/>.

Hildebrand, Hendrik (2018): Deep Reinforcement Learning zur Minderung von Verspätungen im ÖPNV. Bachelorarbeit. Technische Universität, Dortmund. Online verfügbar unter http://kissen.cs.uni-dortmund.de:8080/PublicPublicationFiles/hildebrandt_2018a.pdf, zuletzt geprüft am 15.12.2022.

Huber, Tobias (2018): Tiefes bestärkendes Lernen: Grundlagen, Approximationseigenschaft und Implementierung multimodaler Erklärungen. Masterarbeit. Universität Augsburg, Augsburg.

Josi, Damian (2020): Qualität von OpenStreetMap-Daten. Einführung und Arten der Qualitätsbeurteilung. Paper. Universität Bern, Bern. Institut für Wirtschaftsinformatik. Online verfügbar unter https://www.digitale-nachhaltigkeit.unibe.ch/unibe/portal/fak_naturwis/a_dept_math/c_iinfamath/abt_digital/content/e90958/e490158/e900462/e977579/e977582/e980453/OpenData2020_DamianJosi_Vertiefungsartikel_ger.pdf, zuletzt geprüft am 18.11.2022.

Kerkhoff, Felix (2020): Tiefes Q-Lernen mit Demonstrationen. Bachelorarbeit. Rheinische Friedrichs-Wilhelms-Universität, Bonn. Institut für numerische Simulation. Online verfügbar unter https://ins.uni-bonn.de/media/public/publication-media/BA_Skript_Felix_Kerkhoff.pdf, zuletzt geprüft am 15.12.2022.

Krämer, Andreas (2022): exeo OpinionTRAIN. 9-Euro-Ticket: Blick zurück und nach vorne - Nutzerprofil, Nutzung und Bewertungen. exeo Strategic Consulting AG. Bonn, 08.09.2022. Online verfügbar unter https://www.researchgate.net/publication/363417545_exeo_OpinionTRAIN_IV_9_EUR_Ticket_Post_I_220907, zuletzt geprüft am 10.09.2022.

Larsson, Emil (2018): Evaluation of Pretraining Methods for Deep Reinforcement Learning. Universität Uppsala, Uppsala.

Liggieri, Kevin; Müller, Oliver (2019): Mensch-Maschine-Interaktion. Stuttgart: J.B. Metzler.

Lorenz, Uwe (2020): Reinforcement Learning. Berlin, Heidelberg: Springer Berlin Heidelberg.

Lüth, Carsten (2019): A Review of: Human-Level Control through deep Reinforcement Learning. Seminararbeit. Universität Heidelberg, Heidelberg. Department of Computer

Science. Online verfügbar unter https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/213797145/report_carsten_lueth_human_level_control.pdf, zuletzt geprüft am 13.11.2022.

Mainzer, Klaus (2019): Künstliche Intelligenz – Wann übernehmen die Maschinen? Berlin, Heidelberg: Springer Berlin Heidelberg.

Meier, Andreas; Portmann, Edy (2016): Smart City. Wiesbaden: Springer Fachmedien Wiesbaden.

Mishra, Sukumar; Sood, Yog Raj; Tomar, Anuradha (Hg.) (2019): Applications of Computing, Automation and Wireless Systems in Electrical Engineering. Singapore: Springer (Springer eBooks Engineering, 553).

Nahrstedt, Harald (2012): Algorithmen für Ingenieure. Wiesbaden: Vieweg+Teubner Verlag.

Niebler, Paul (2018): Datenbasiert Entscheiden. Ein Leitfaden Für Unternehmer und Entscheider. Unter Mitarbeit von Dominic Lindner. Wiesbaden: Gabler (Essentials Ser). Online verfügbar unter <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5566817>.

OSM-Highway (2022): Erklärung zum Schlüssel HIGHWAY. Hg. v. OpenStreetMap Community. Online verfügbar unter <https://wiki.openstreetmap.org/wiki/Key:highway>, zuletzt aktualisiert am 26.10.2022, zuletzt geprüft am 18.11.2022.

OSM-PSV (2022): Erklärung zum Schlüssel PSV. Hg. v. OpenStreetMap Community. Online verfügbar unter <https://wiki.openstreetmap.org/wiki/DE:Key:psv?uselang=de>, zuletzt aktualisiert am 29.08.2022, zuletzt geprüft am 18.11.2022.

Pong, Vitchyr; Gu, Shixiang; Dalal, Murtaza; Levine, Sergey (2018): Temporal Difference Models: Model-Free Deep RL for Model-Based Control.

Reinhardt, Winfried (2018): Öffentlicher Personennahverkehr. Technik – rechts- und betriebswirtschaftliche Grundlagen. 2., aktualisierte Auflage. Wiesbaden: Springer Vieweg. Online verfügbar unter <http://www.springer.com/>.

Rellensmann, Johanna (2019): Mathematisches Modellieren. In: Johanna Rellensmann (Hg.): Selbst erstellte Skizzen beim mathematischen Modellieren. Wiesbaden: Springer Fachmedien Wiesbaden (Studien zur theoretischen und empirischen Forschung in der Mathematikdidaktik), S. 5–30.

Rimscha, Markus von (2008): Algorithmen kompakt und verständlich. Lösungsstrategien am Computer. 1. Aufl. Wiesbaden: Vieweg + Teubner (Programmiersprachen, Datenbanken und Softwareentwicklung).

Schaaf, Marc; Wilke, Gwendolin (2015): Ereignisverarbeitung zur Flexiblen Dynamischen Informationsverarbeitung in Smart Cities. In: *HMD* 52 (4), S. 562–571. DOI: 10.1365/s40702-015-0149-x.

Scherm, Jürgen; Hübener, Reinhard; Dobeschinsky, Harry; Kühne, Reinhart D. (2001): Opti*Bus : Optimierungschancen für das Verkehrssystem Bus im ÖPNV ; Ergebnisse des Kongresses im Themenbereich Verkehr und Raumstruktur. Unter Mitarbeit von Universität Stuttgart.

Schmitz, Martin (2017): Ein Vergleich von Reinforcement Learning Algorithmen für dynamische und hochdimensionale Probleme. Bachelorarbeit. Universität Koblenz / Landau, Koblenz. Institute for Web Science and Technology. Online verfügbar unter <https://west.uni-koblenz.de/assets/theses/vergleich-von-reinforcement-algorithmen.pdf>, zuletzt geprüft am 13.11.2022.

Schranil, Steffen (2013): Prognose der Dauer von Störungen des Bahnbetriebs. ETH Zurich.

Searle, John R. (1980): Minds, brains, and programs. In: *Behav Brain Sci* 3 (3), S. 417–424. DOI: 10.1017/S0140525X00005756.

Simeonov, Georgi (2017): Ein interaktiver visueller Ansatz für das Map Matching von großen Bewegungsdatensätzen. Bachelorarbeit. Universität Stuttgart, Stuttgart. Institut für Visualisierung und Interaktive Systeme. Online verfügbar unter https://www2.informatik.uni-stuttgart.de/bibliothek/ftp/medoc.ustuttgart_fi/BCLR-2017-62/BCLR-2017-62.pdf, zuletzt geprüft am 23.11.2022.

Soike, Roman; Libbe, Jens (2018): Smart Cities in Deutschland. Eine Bestandsaufnahme. Hg. v. Deutsches Institut für Urbanistik. Berlin (Difu-Papers). Online verfügbar unter <https://repository.difu.de/jspui/handle/difu/248050>, zuletzt geprüft am 23.11.2022.

Statistisches Bundesamt (2020): Einwohnerzahl der größten Städte in Deutschland am 31. Dezember 2020. Hg. v. Statistisches Bundesamt. Online verfügbar unter <https://de.statista.com/statistik/daten/studie/1353/umfrage/einwohnerzahlen-der-grossstaedte-deutschlands/>, zuletzt geprüft am 23.11.2022.

Sutton, Richard S.; Barto, Andrew (2018): Reinforcement learning. An introduction. Second edition. Cambridge, Massachusetts, London, England: The MIT Press (Adaptive computation and machine learning).

Terhag, Felix (2018): Reinforcement Learning zur Erstellung zustandsorientierter Putzstrategien in konzentrierenden solarthermischen Kraftwerken. Masterarbeit. Universität zu Köln, Köln. Mathematisches Institut. Online verfügbar unter https://elib.dlr.de/125485/1/__ASFS01_Group_Students_felix_terhag_Master%20Arbeit_MasterArbeitFTFinal.pdf, zuletzt geprüft am 16.12.2022.

VDV-Schrift 730, 2015: VDV-Schrift 730. Online verfügbar unter <https://knowhow.vdv.de/documents/730/>, zuletzt geprüft am 07.12.2022.

VDV-Schrift 736-2, 2019: VDV-Schrift 736-2. Online verfügbar unter <https://www.vdv.de/736-2-sds.pdf>, zuletzt geprüft am 24.11.2022.

Wagner, Stefan Sylvius (2018): Entwicklung eines Reinforcement Learning basierten Flugzeugautopiloten unter der Verwendung von Deterministic Policy Gradients. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg, Hamburg. Online verfügbar unter <https://autosys.informatik.haw-hamburg.de/papers/2018Wagner.pdf>, zuletzt geprüft am 07.12.2022.

Williams, Ronald J. (1992): Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: *Mach Learn* 8 (3-4), S. 229–256. DOI: 10.1007/BF00992696.

Witt, Laura Jasmin (2019): Evaluierung von Reinforcement Learning Algorithmen zur Erweiterung eines bestehenden Trajektorienfolgeregelungskonzeptes. Masterarbeit. Freie Universität Berlin, Berlin. Online verfügbar unter https://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Master_Diploma-theses/2019/Witt/MA-Witt.pdf, zuletzt geprüft am 14.12.2022.

Anhang

