HAW HAMBURG

BACHELOR THESIS
Ersan Baran

Tensorbasierte Agenten zur Ausbreitungssimulation von Infektionskrankheiten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Ersan Baran

Tensorbasierte Agenten zur Ausbreitungssimulation von Infektionskrankheiten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Prof. Dr. Jürgen May

Eingereicht am: 11. Juli 2024

Ersan Baran

Thema der Arbeit

Tensorbasierte Agenten zur Ausbreitungssimulation von Infektionskrankheiten

Stichworte

Simulation, Tensoren, Multi-Agenten-System, Infektionskrankheit, Neuronale Netze, Backpropagation, Differentialgleichung, Transformer

Kurzzusammenfassung

Infektionskrankheiten stellen eine anhaltende Bedrohung für die globale Gesundheit dar. Die COVID-19-Pandemie hat die Notwendigkeit präziser Simulationsmodelle verdeutlicht. Traditionelle epidemiologische Modelle stoßen jedoch an ihre Grenzen, wenn es darum geht, die Komplexität realer Ausbreitungsprozesse abzubilden. Diese Bachelorarbeit kombiniert die Stärken von Multi-Agenten-Systemen und neuronalen Netzen, um ein neuartiges Simulationsmodell zu entwickeln, das individuelle Verhaltensweisen, räumliche Heterogenität und dynamische Interaktionen berücksichtigt. Dieser innovative Ansatz verspricht realistischere performante Simulationen und somit eine bessere Unterstützung bei der Entwicklung effektiver Strategien zur Bekämpfung von Infektionskrankheiten.

Ersan Baran

Title of Thesis

Tensor-based agents for epidemic simulation

Keywords

Simulation, Tensors, Multi-agent systems, Infectious diseases, Neural networks, Backpropagation, Differential equation, Transformer

Abstract

Infectious diseases pose a persistent threat to global health. The COVID-19 pandemic has highlighted the need for accurate simulation models. However, traditional epidemiological models struggle to capture the complexity of real-world spreading processes. This

bachelor thesis combines the strengths of multi-agent systems and neural networks to develop a novel simulation model that accounts for individual behaviors, spatial heterogeneity, and dynamic interactions. This innovative approach promises more realistic and performant simulations, thus providing better support for developing effective strategies to combat infectious diseases.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Hintergrund und Motivation

Infektionskrankheiten stellen aktuell eine der größten globalen Herausforderungen dar. Die COVID-19-Pandemie hat verdeutlicht, wie anfällig die Gesellschaft gegenüber sich schnell ausbreitenden Krankheitserregern ist. Die Fähigkeit, das Ausbreitungsverhalten solcher Krankheiten präzise vorherzusagen, ist entscheidend zum Finden von effektiven Entscheidungen im Bereich der öffentlichen Gesundheit, Ressourcenplanung und Pandemieprävention. Traditionelle epidemiologische Modelle wie das SIR-Modell (Susceptible Infected Recovered) haben wertvolle Einblicke in die grundlegende Dynamik von Infektionskrankheiten geliefert (vgl. Hethcote 2000). Diese Modelle sind jedoch oft zu simpel, um die Komplexität realer Ausbreitungsprozesse vollständig abzubilden. Insbesondere vernachlässigen sie die Unterschiede im Verhalten, die räumliche Heterogenität der Bevölkerung und die dynamische Interaktion zwischen Individuen und ihrer Umgebung.

1.2 Multiagentensysteme, Tensoren und neuronale Netze

Multiagentensysteme (MAS) bieten einen vielversprechenden Ansatz zur Modellierung komplexer Systeme, in denen eine Vielzahl von Individuen interagiert (vgl. Dorri u. a. 2018). Im Kontext von Infektionskrankheiten können MAS die individuellen Verhaltensweisen, sozialen Kontakte und räumlichen Bewegungen von Personen simulieren. Dies ermöglicht eine realistischere Darstellung der Ausbreitungsprozesse als herkömmliche Modelle.

Tensoren, als mathematische Objekte, die mehrdimensionale Daten repräsentieren können, spielen eine zentrale Rolle in der modernen Datenanalyse und im maschinellen Lernen (vgl. Kossaifi u. a. 2019). In der Simulation bieten Tensoren eine effiziente Möglich-

keit, komplexe Beziehungen zwischen Individuen, deren Zuständen und der Umgebung darzustellen.

Neuronale Netze (vgl. LeCun u. a. 2015), insbesondere Transformer-Architekturen (vgl. Vaswani u. a. 2017), haben in den letzten Jahren bemerkenswerte Fortschritte in der Verarbeitung natürlicher Sprache und im maschinellen Lernen erzielt. Ihre Fähigkeit, komplexe nichtlineare Beziehungen zu erfassen und aus großen Datenmengen zu lernen, macht sie zu einem vielversprechenden Werkzeug für die Simulation von Infektionskrankheiten.

1.3 Zielsetzung und Fragestellung

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines Simulationsmodells für Infektionskrankheiten, das auf einem Multi-Agenten-System und Tensoren basiert. Dieses Modell soll in der Lage sein, realistische Ausbreitungsszenarien abzubilden, indem es individuelle Verhaltensweisen, räumliche Heterogenität und dynamische Interaktionen berücksichtigt, performant sein und eine effiziente Kalibrierung bieten. Die zentrale Fragestellung dieser Arbeit lautet: Wie können Tensoren und neuronale Netze in die Simulation von Infektionskrankheiten integriert werden, um realistischere Ergebnisse zu erzielen? Insbesondere wird untersucht, wie sie zur Optimierung von Modellparametern und zur Modellierung individueller Verhaltensweisen eingesetzt werden können.

1.4 Struktur der Arbeit

Die Bachelorarbeit gliedert sich in mehrere Abschnitte: Zunächst werden die theoretischen Grundlagen erläutert, darunter epidemiologische Modelle, Tensoren, neuronale Netze und Multi-Agenten-Systeme. Es folgt eine Übersicht über den aktuellen Forschungsstand mit Schwerpunkt auf dem GRADABM-Modell (vgl. Chopra u. a. 2023). Anschließend wird die Methodik detailliert beschrieben, einschließlich des verwendeten Multi-Agenten-Frameworks MARS, des zentralen Zustandstensors und der Integration des neuronalen Netzes. Die Implementierung des Modells wird im nächsten Abschnitt behandelt, wobei eine ausführlichere Darstellung im Anhang zu finden ist. Es folgt eine

Präsentation und Diskussion der Simulationsergebnisse. Abschließend werden die wichtigsten Ergebnisse zusammengefasst, Schlussfolgerungen gezogen und ein Ausblick auf zukünftige Forschungsrichtungen gegeben.

2 Theoretische Grundlagen

2.1 Multiagentensysteme

Multiagentensysteme (MAS) sind eine Klasse von Systemen, die aus mehreren interagierenden Agenten bestehen. Diese Agenten arbeiten zusammen oder konkurrieren miteinander, um individuelle oder gemeinsame Ziele zu erreichen.

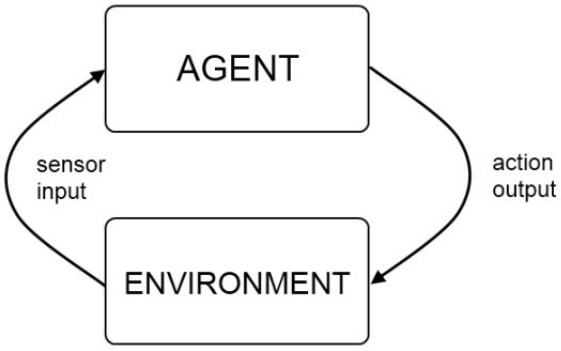

Abbildung 2.1: Darstellung der Interaktion zwischen einem Agenten und der Umgebung (Constandache und Leon 2012)

2.1.1 Grundlegende Konzepte

- **Agenten:** Ein Agent ist eine autonome Einheit, die in der Lage ist, ihre Umgebung wahrzunehmen, Entscheidungen zu treffen und Aktionen auszuführen, um ihre Ziele zu erreichen. Agenten können unterschiedliche Fähigkeiten, Wissen und Ziele haben.
- **Umgebung:** Die Umgebung ist der Raum, in dem die Agenten agieren. Sie kann physisch sein, ein Beispiel hierfür könnten Roboter in einer Fabrikhalle sein, oder

virtuell, zum Beispiel bei Nutzern von sozialen Netzwerken. Die Umgebung kann sich im Laufe der Zeit ändern.

- **Interaktionen:** Agenten interagieren miteinander, indem sie Informationen austauschen, kooperieren, konkurrieren oder verhandeln. Diese Interaktionen können direkt, durch Kommunikation, oder indirekt zum Beispiel durch Veränderungen in der Umgebung erfolgen.
- **Kommunikation:** Kommunikation ist ein zentraler Aspekt der Interaktion zwischen Agenten. Agenten können verschiedene Kommunikationsformen nutzen, wie den Nachrichtenaustausch.

2.1.2 Implementierung von MAS mit Agentenbasierten Modellen

Agentenbasierte Modelle (ABM) sind eine spezielle Methode der Computersimulation, bei der das Verhalten eines Systems durch die Interaktionen vieler einzelner Agenten modelliert wird (vgl. Norton u. a. 2019; Zheng u. a. 2021; Bonabeau 2002). ABMs basieren auf der Theorie von Multiagentensystemen und werden häufig verwendet, um komplexe Systeme zu untersuchen, deren Verhalten sich aus dem Zusammenspiel vieler individueller Entscheidungen ergibt (Emergentes Verhalten).

ABMs bieten gegenüber herkömmlichen Modellen den Vorteil, dass sie die individuellen Unterschiede innerhalb einer Population besser darstellen können (Heterogenität). Durch die Berücksichtigung der einzigartigen Eigenschaften und Verhaltensweisen jedes Agenten können realistischere Simulationen von komplexen Systemen erstellt werden, in denen individuelle Unterschiede eine wichtige Rolle spielen. ABMs können auch das sich anpassende Verhalten von Individuen modellieren (Adaptivität), was die Simulation von Systemen ermöglicht, in denen das Verhalten der Agenten auf Veränderungen in ihrer Umgebung oder Interaktionen mit anderen Agenten reagiert. Darüber hinaus können ABMs auf einer sehr detaillierten Ebene simuliert werden(Granularität), was eine präzise Analyse der Auswirkungen von Entscheidungen und Maßnahmen auf das gesamte System ermöglicht. Dank ihrer flexiblen Modellierungssprache können ABMs an eine Vielzahl von Anwendungsfällen angepasst werden (Flexibilität), was die Entwicklung von Modellen ermöglicht, die auf die spezifischen Bedürfnisse der jeweiligen Anwendung zugeschnitten sind.

ABMs haben jedoch auch einige Nachteile. Sie können aufgrund der großen Anzahl von Agenten und ihren Interaktionen sowie der Notwendigkeit, individuelle Eigenschaften und Verhaltensweisen zu berücksichtigen, komplexer in der Entwicklung und Implementierung sein als herkömmliche Modelle (Komplexität). Die Simulation von ABMs kann insbesondere bei großen Systemen mit vielen Agenten rechenintensiv sein (Rechenintensivität), da viele Berechnungen erforderlich sind, um die Interaktionen zwischen den Agenten und die Auswirkungen der Umgebung zu simulieren. Die Kalibrierung von ABMs kann schwierig sein, da es aufwendig sein kann, die Modellparameter so einzustellen, dass sie die Realität genau widerspiegeln, was zu ungenauen oder unrealistischen Simulationsergebnissen führen kann. Außerdem ist es möglich dass die Validierung von ABMs wegen der Komplexität problematisch werden kann, es ist nicht offensichtlich ob das Modell das reale System korrekt abbildet, was zu Fehlern in den Simulationsergebnissen führen kann.

Ein gutes Beispiel der Vor- und Nachteile von ABMs bietet Pellis u. a. (2015), in dem die Agenten, die Wirte sind und über ihre Kontaktnetzwerke Viren übertragen. Trotz der Fähigkeit mit heterogenen Daten arbeiten zu können, hatte man wegen der schlechten Skalierung von ABMs Probleme.

2.1.3 Das MARS Framework

Das MARS Framework ist ein Framework, das speziell für die Entwicklung und Ausführung von agentenbasierten Modellen entwickelt wurde (vgl. MARS-Group 2024a). MARS bietet eine Vielzahl von Funktionen, die die Implementierung von MAS erleichtern, darunter:

- **Agentenmodellierung:** MARS ermöglicht die Modellierung von Agenten und deren Verhalten.
- **Umgebungsmodellierung:** MARS ermöglicht die Erstellung komplexer 2D- und 3D-Umgebungen, in denen Agenten interagieren können.
- **Simulationssteuerung:** MARS bietet umfangreiche Funktionen zur Steuerung und Analyse von Simulationen.

Die Simulationen der Modelle sind Tick-basiert. Tick-basierte Simulationen laufen im Gegensatz zu kontinuierlichen Simulationen in festen Zeitschritten ab. Jeder Zeitschritt, auch Tick genannt, repräsentiert einen gleich großen Zeitabschnitt Δt. Diese Zeitabschnitte sind für die gesamte Simulation konstant. Zum Beispiel kann eine Simulation in

Schritten von 1 Minute ablaufen, in der nach einem Tick eine Minute in der Simulation vergangen ist.

2.2 Epidemiologische Modelle

Die Modellierung in der Epidemiologie spielt eine entscheidende Rolle beim Verständnis der Ausbreitungsdynamik von Infektionskrankheiten, der Vorhersage zukünftiger Entwicklungen und der Bewertung der Wirksamkeit von Interventionen. Durch die Simulation können Epidemiologen die Auswirkungen von Variablen wie Übertragungsraten, Inkubationszeiten und Immunität auf die Krankheitsausbreitung analysieren. Diese Erkenntnisse sind wichtig, um gezielte Maßnahmen und Strategien zur Eindämmung von Epidemien und Pandemien zu entwickeln.

2.2.1 Basisreproduktionszahl R_0

Die Basisreproduktionszahl R_0, eine wichtige Kennzahl in der Epidemiologie, gibt an, wie viele Menschen im Durchschnitt von einer infizierten Person angesteckt werden (vgl. Delamater u. a. 2019). Er wird unter der Annahme berechnet, dass jeder in der Bevölkerung anfällig für die Krankheit ist, also das niemand immun ist und es keine Maßnahmen zur Eindämmung der Krankheit gibt. Durch diese Annahme sehen wir, wie schnell sich eine Krankheit ausbreiten könnte, wenn nichts unternommen wird. In der Realität ist R_0 generell kleiner. Bei der Interpretation von R_0 ist es wichtig zu beachten, dass R_0 ein Durchschnittswert ist. Nicht jede infizierte Person wird genau R_0 andere Personen anstecken. Einige werden mehr anstecken, andere weniger. Durch den R_0 Wert haben wir jedoch eine Vorstellung von der allgemeinen Übertragbarkeit der Krankheit.

- $R_0 > 1$**:** Wenn R_0 größer als 1 ist, steckt jede infizierte Person im Durchschnitt mehr als eine weitere Person an. In diesem Fall kann sich die Krankheit exponentiell ausbreiten und zu einer Epidemie oder sogar Pandemie führen. Je höher R_0 ist, desto schneller verbreitet sich die Krankheit.
- $R_0 = 1$**:** Wenn R_0 gleich 1 ist, steckt jede infizierte Person im Durchschnitt genau eine weitere Person an. Die Krankheit bleibt in diesem Fall stabil und breitet sich nicht weiter aus, stirbt aber auch nicht aus.

- $R_0 < 1$**:** Wenn R_0 kleiner als 1 ist, steckt jede infizierte Person im Durchschnitt weniger als eine weitere Person an. In diesem Fall nimmt die Anzahl der Neuinfektionen ab, und die Krankheit wird wahrscheinlich aussterben.

Eine Herausforderung der Bedeutung von R_0 für epidemiologische Modelle ist, dass er schwer zu messen ist, da es kein statischer Wert ist und sich im Laufe einer Epidemie dynamisch verändern kann. Diese Veränderungen hängen von Faktoren wie der Immunität in der Bevölkerung und den ergriffenen Maßnahmen zur Eindämmung der Krankheit ab. Zudem ist die Übertragung einer Krankheit nicht gleichmäßig über die Bevölkerung verteilt, da einige Menschen mehr Kontakte haben als andere und daher anfälliger für Infektionen sind. Weiterhin sind die Daten über Infektionen in der Realität oft unvollständig, was die genaue Bestimmung von R_0 zusätzlich erschwert.

2.2.2 Modellierungsansätze

Es gibt mehrere Modellierungsansätze, die jeweils unterschiedliche Stärken und Schwächen aufweisen und für verschiedene Problemstellungen geeignet sind.

Kompartmentmodelle, die die Bevölkerung in verschiedene Gruppen oder Krankheitsstadien (Kompartments) wie anfällig, infiziert und genesen einteilen, sind einfach und gut zu berechnen (vgl. Silva und Torres 2017). Diese Modelle basieren jedoch oft auf vereinfachten Annahmen, die die Komplexität realer Ausbreitungsprozesse nicht vollständig erfassen können. Kompartment-Modelle sind typischerweise differenzierbar und verwenden Differentialgleichungen, um den Übergang bei Individuen von einem Kompartment in einen anderen zu modellieren. Sie ermöglichen eine Modellierung des Verlaufs von Infektionskrankheiten, indem sie die Anzahl der Individuen in den verschiedenen Krankheitsstadien im Laufe der Zeit beschreiben. Epidemiologen müssen zur Untersuchung der Auswirkungen von Interventionen oder Verhaltungsänderungen nur die Parameter des Modells anpassen. Da die Modelle differenzierbar sind können Simulationsparameter durch effiziente Gradienten-basierte Verfahren optimiert werden. Was zu einer schnellen Anpassung des Modells an reale Daten führt, um eine bessere Übereinstimmung zwischen Modell und Beobachtungen zu erreichen.

ABMs hingegen modellieren Individuen als Agenten, die miteinander interagieren und sich in einer Umgebung bewegen. Diese Modelle ermöglichen die Berücksichtigung individueller Verhaltensweisen, räumlicher Heterogenität und sozialer Netzwerke. Somit sind

ABMs flexibler als Kompartmentmodelle, allerdings auch komplexer und benötigen eine höhere Rechenleistung.

Netzwerkmodelle stellen die Bevölkerung als Netzwerk dar, wobei die Knoten Individuen und die Kanten die Kontakte zwischen den Individuen repräsentieren (vgl. Keeling und Eames 2005). Weil die Kontakte zwischen Individuen explizit abgebildet werden, kann dieser Ansatz zu einer genaueren Analyse von Übertragungsmustern führen. Allerdings kann die Simulation rechenintensiv sein. Dazu kommt dass die Erstellung eines solchen Modells detaillierte Daten über soziale Kontakte benötigt, welche nicht immer verfügbar sind.

2.2.3 Gradientenverfahren

Das Gradientenverfahren wird in der epidemiologischen Modellierung verwendet, um die Parameter eines Modells zu kalibrieren (vgl. Andrychowicz u. a. 2016; Ruder 2017). Es nutzt die Eigenschaft aus, dass wenn man bei einer differenzierbaren Funktion sich in Richtung des negativen Gradienten bewegt, man sich zu einem Minimum bewegt. Formal ausgedrückt: $\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$ wobei γ die Lernrate ist, die bestimmt um welchen Faktor man sich zum negativen Gradienten bewegt. Um die Parameter eines Modells zu bestimmen, kann man sie zufällig initialisieren und anschließend die Simulation durchlaufen lassen. Danach vergleicht man die Ergebnisse mit den realen Datensätzen und berechnet die Kostenfunktion zwischen realen Daten und dem Modellergebnis. Anschließend wird der Gradient der Kostenfunktion in Bezug auf die Parameter gebildet und die Parameter werden je nach Lernrate geändert indem $\theta = \theta - \gamma \nabla F(\theta)$ berechnet wird. θ ist der Parameter. Ein visuelles Beispiel für das Gradientenverfahren sieht man in Abbildung 2.2 dass das Gradientenverfahren bei einer Funktion mit 2 Variablen demonstriert. Das Gradientenverfahren ist durch die Eigenschaft, dass die Steigung größer wird wenn man weiter vom Minimum entfernt ist, ein viel schnelleres Verfahren um Parameter zu kalibrieren.

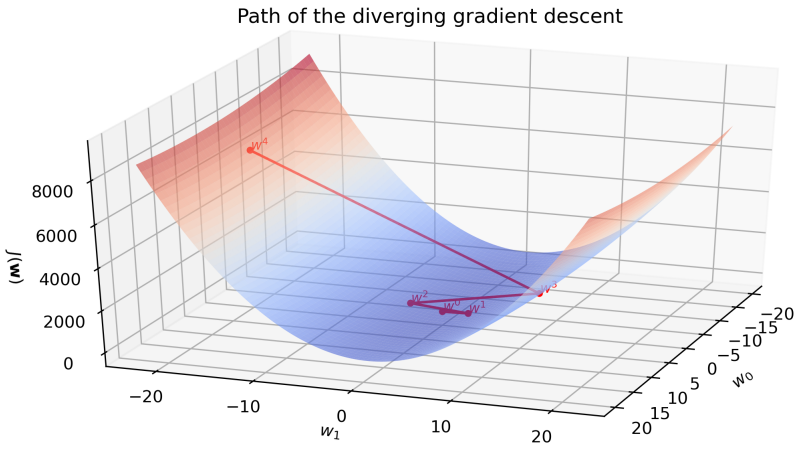

Abbildung 2.2: Schrittweises Nähern zum Minimum durch das Gradientenverfahren (Kass 2024)

2.2.4 SIR-ähnliche Modelle

Das bekannteste Beispiel eines Kompartmentmodells, das als Grundlage vieler Kompartment-Modelle dient, ist das SIR-Modell (vgl. Hethcote 2000). Es modelliert Übergänge zwischen den Kompartments Susceptible(Eine Person die gesund und anfällig für eine Infektion ist), Infectious(Eine Person die infiziert ist und andere anstecken kann) und Recovered(Eine Person die die Infektion hinter sich hat und eine Immunität entwickelt hat). Die Modelle die auf dem SIR Modell basieren werden als SIR-ähnliche Modelle bezeichnet. Sie sind einfach gestaltet, gut zu verstehen und leicht zu implementieren.

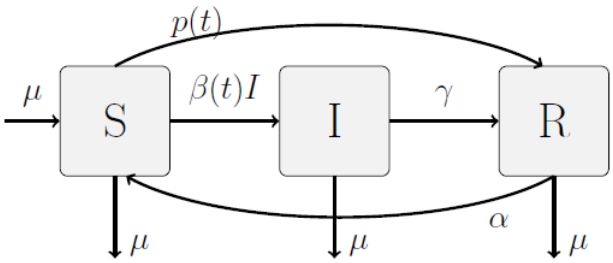

Abbildung 2.3: Diagramm des SIRS-Modells (Doutor u. a. 2016)

Ein Beispiel für ein SIR-ähnliches Modell ist das SIRS-Modell (vgl. Doutor u. a. 2016). Im Gegensatz zum SIR-Modell, bei dem Individuen nach der Genesung immun bleiben, können im SIRS-Modell genesene Individuen ihre Immunität verlieren und wieder in die Gruppe der Anfälligen zurückkehren. Die Übergänge der verschiedenen Kompartments sind in Abbildung 2.3 zu sehen. Die Knoten stellen die Kompartments und die Kanten die Übergänge dar. μ ist die Geburts- sowie Sterberate. α stellt die zeitliche Immunität dar und und γ die Erholungsrate. Die Funktionen $\beta(t), p(t)$ sind die Übergangsfunktion und die Impfrate. Die Differentialgleichungen beschreiben die zeitliche Veränderung der Anzahl der Individuen in den einzelnen Kompartments mit den Formeln:

$$\begin{aligned} S' &= \mu + \alpha R - \beta(t)IS - p(t)S - \mu S \\ &= \mu + \alpha - \alpha|I - \beta(t)IS - p(t)S - (\mu + \alpha)S \\ I' &= \beta(t)IS - \gamma I - \mu I \\ R' &= \gamma I + p(t)S - \mu R - \alpha R \end{aligned} \tag{2.1}$$

S' beschreibt die Veränderung der Anfälligen. Sie nimmt durch die Geburten μ und den Genesenen die ihre Immunität verlieren zu αR. Sie nimmt durch die Tode μS, Infektionen $\beta(t)IS$ und Impfungen $p(t)S$ ab.
I' beschreibt die Veränderung der Infizierten. Sie nimmt durch die Infektionen $\beta(t)IS$ zu. Sie nimmt durch die Tode μI und Genesung γI ab.
R' beschreibt die Veränderung der Genesenen. Sie nimmt durch die Genesung γI und den Impfungen $p(t)S$ zu. Sie nimmt durch die Tode μR und den Verlust der Immunität αR ab.

Das SIRS-Modell bietet eine gute Veranschaulichung der Nachteile von SIR-ähnlichen Modellen. Die Interaktionen und Übertragungen der Bevölkerung werden vereinfacht und homogenisiert dargestellt. Jedes Individuum hat die gleiche Wahrscheinlichkeit, mit jedem anderen in Kontakt zu kommen, was die komplexen und ungleich verteilten sozialen Interaktionen in der Realität nicht widerspiegelt. Zudem vernachlässigen sie räumliche Heterogenität, die in der realen Welt zu unterschiedlichen Ausbreitungsgeschwindigkeiten in verschiedenen Regionen führen kann. Die Annahme konstanter Geburten- und Sterberaten ist eine weitere Vereinfachung, da diese Raten in Wirklichkeit je nach Altersgruppe und anderen Faktoren variieren können. Individuelle Unterschiede in Anfälligkeit, Immunität und Verhalten, die die Ausbreitung von Krankheiten beeinflussen können, werden ebenfalls nicht berücksichtigt. Um realistischere Vorhersagen zu treffen, sind komplexere Modelle erforderlich, die diese Aspekte berücksichtigen und die Heterogenität der Be-

völkerung, räumliche Faktoren und individuelle Unterschiede einbeziehen. Obwohl einige Arbeiten separate Kompartment-Modelle für verschiedene Subpopulationen entwickelt haben, wie zum Beispiel das Age-SEIR-Modell (vgl. Mossong u. a. 2008) und das Geo-SEIR-Modell (vgl. Balcan u. a. 2009), ist es aufgrund weiterer vereinfachter Annahmen oft schwierig, die Vielfalt der individuellen Eigenschaften und Verhaltensweisen vollständig abzubilden.

2.3 Tensoren

Tensoren, mehrdimensionale Arrays, die Daten strukturiert und effizient darstellen, sind ein zentrales Konzept in der modernen Datenanalyse und im maschinellen Lernen (vgl. Kossaifi u. a. 2019; Sidiropoulos u. a. 2016). Moderne Hardware, insbesondere GPUs (Graphics Processing Units), sind darauf spezialisiert, mathematische Operationen auf Tensoren parallel auszuführen. Dadurch können Berechnungen, die normalerweise sequentiell und zeitaufwändig wären, erheblich beschleunigt werden. Beliebte Frameworks zur Berechnung mit Tensoren sind TensorFlow (vgl. TensorFlow 2024) und PyTorch (vgl. PyTorch 2024).

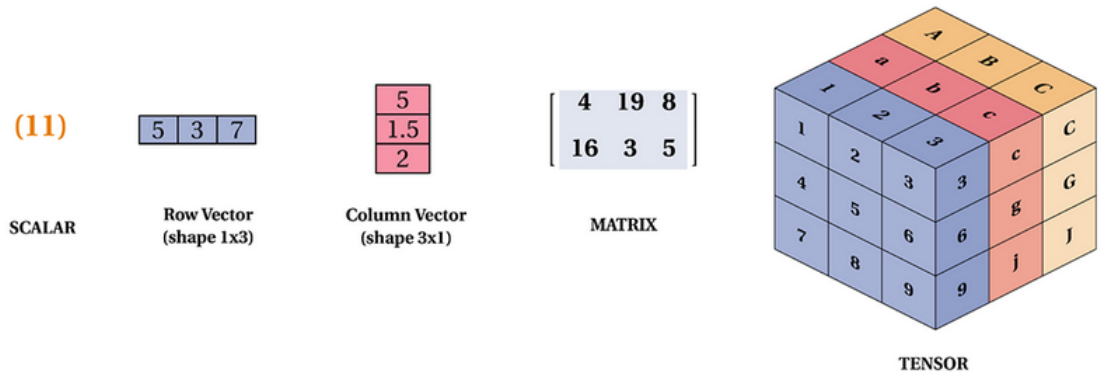

Abbildung 2.4: Veranschaulichung der Beziehung zwischen Skalaren, Vektoren, Matrizen und Tensoren in der linearen Algebra (SciSharp 2024)

In der Abbildung 2.4 kann man sehen wie die verschiedenen Strukturen der linearen Algebra zu Tensoren stehen. Ein Skalar ist ein einzelner numerischer Wert, in diesem Fall 11. Ein Vektor ist eine geordnete Liste von Zahlen, die entweder als Zeilenvektor oder als Spaltenvektor dargestellt werden kann. Ein Zeilenvektor ordnet die Zahlen in einer horizontalen Reihe an, hier [5 3 7], während ein Spaltenvektor die Zahlen in einer

vertikalen Spalte anordnet. Eine Matrix ist eine Anordnung von Zahlen in Zeilen und Spalten, hier eine 2x3-Matrix mit den Elementen 4, 19, 8, 16, 3, 5. Ein Tensor ist eine Verallgemeinerung von Skalaren, Vektoren und Matrizen auf höhere Dimensionen und kann als mehrdimensionales Array von Zahlen dargestellt werden, wie hier ein 3x3x3 Tensor, der als Würfel visualisiert wird. In der Programmierung werden diese Strukturen oft zu Tensoren zusammengefasst. Ein Skalar kann als Tensor der Ordnung 0, ein Vektor als Tensor der Ordnung 1 und eine Matrix als Tensor der Ordnung 2 betrachtet werden. Ein Tensor kann eine beliebige Ordnung haben, sei es 0, 1, 2, 3 oder höher.

2.3.1 Tensoren in der Modellierung

In der modernen Datenverarbeitung und Modellierung spielen Tensoren eine entscheidende Rolle. Ihre Verwendung hat sich in zahlreichen Bereichen etabliert, darunter dem maschinelles Lernen und der Datenanalyse. Sie ermöglichen eine effiziente Darstellung komplexer Daten, da sie Daten mit mehreren Dimensionen darstellen können, wie zum Beispiel Bilder oder Zeitreihen. Das wiederum ermöglicht es ihnen in verschiedenen Bereichen eingesetzt zu werden, wie etwa in der Bildverarbeitung, der Verarbeitung natürlicher Sprache und der Zeitreihenanalyse. Wie bereits erwähnt bieten Tensoren die Möglichkeit der Hardwarebeschleunigung. Moderne Hardware wie GPUs und TPUs (Tensor Processing Units) sind speziell für die Verarbeitung von Tensoren optimiert. Durch die Nutzung dieser Hardware können Berechnungen mit Tensoren erheblich beschleunigt werden, was insbesondere bei großen Modellen und Datensätzen von Vorteil ist. Ein weiterer wichtiger Aspekt ist die automatische Differenzierung. Frameworks wie TensorFlow und PyTorch nutzen Tensoren als zentrale Datenstruktur und bieten automatische Differenzierung. Das bedeutet, dass sie die Ableitungen von Funktionen, die auf Tensoren operieren, automatisch berechnen können. Dies ist für das Training von Modellen mittels Gradientenabstieg von entscheidender Bedeutung, da die Gradienteninformationen verwendet werden, um die Modellparameter iterativ zu aktualisieren und die Leistung des Modells zu verbessern.

2.3.2 Automatische Differenzierung

Die automatische Differenzierung erlaubt es, die Ableitung einer Funktion effizient zu berechnen und findet in der Forschung in den letzten Jahren viel Anwendung, zum Beispiel beim differenzierbaren Sortieren oder Optimieren (vgl. Baydin u. a. 2018; Blondel u. a.

2020). Durch die schnellere Berechnung der Ableitung folgt dass das Gradientenverfahren, welches bei den genannten Problemen verwendet wird, um Einiges beschleunigt wird. Dies ist besonders vorteilhaft bei komplexen Modellen mit vielen Variablen, da das manuelle Ableiten zeitaufwendig, fehleranfällig und schwierig sein kann. Bei der automatischen Differenzierung gibt es den Vorwärtsmodus und den Rückwärtsmodus. Der Vorwärtsmodus wird bei Funktionen mit vielen Eingaben und wenigen Ausgaben verwendet. Komplementär dazu wird der Rückwärtsmodus, im englischen auch Backpropagation genannt, bei Funktionen mit wenigen Eingaben und vielen Ausgaben verwendet. Die Überlegenheit der automatischen Differenzierung gegenüber zu anderen Ableitungsmethoden kann anhand eines Beispiels verdeutlicht werden:

Wir haben die Gleichung $z = x^2 + 2x + sin(y)$ mit den Werten $x = 3, y = \pi$ Zuerst wird ein Berechnungsgraph aufgestellt, in dem jede Teilfunktion von unten nach oben berechnet wird, anschließend hat man einen Graphen wie in Abbildung 2.5 zu sehen ist:

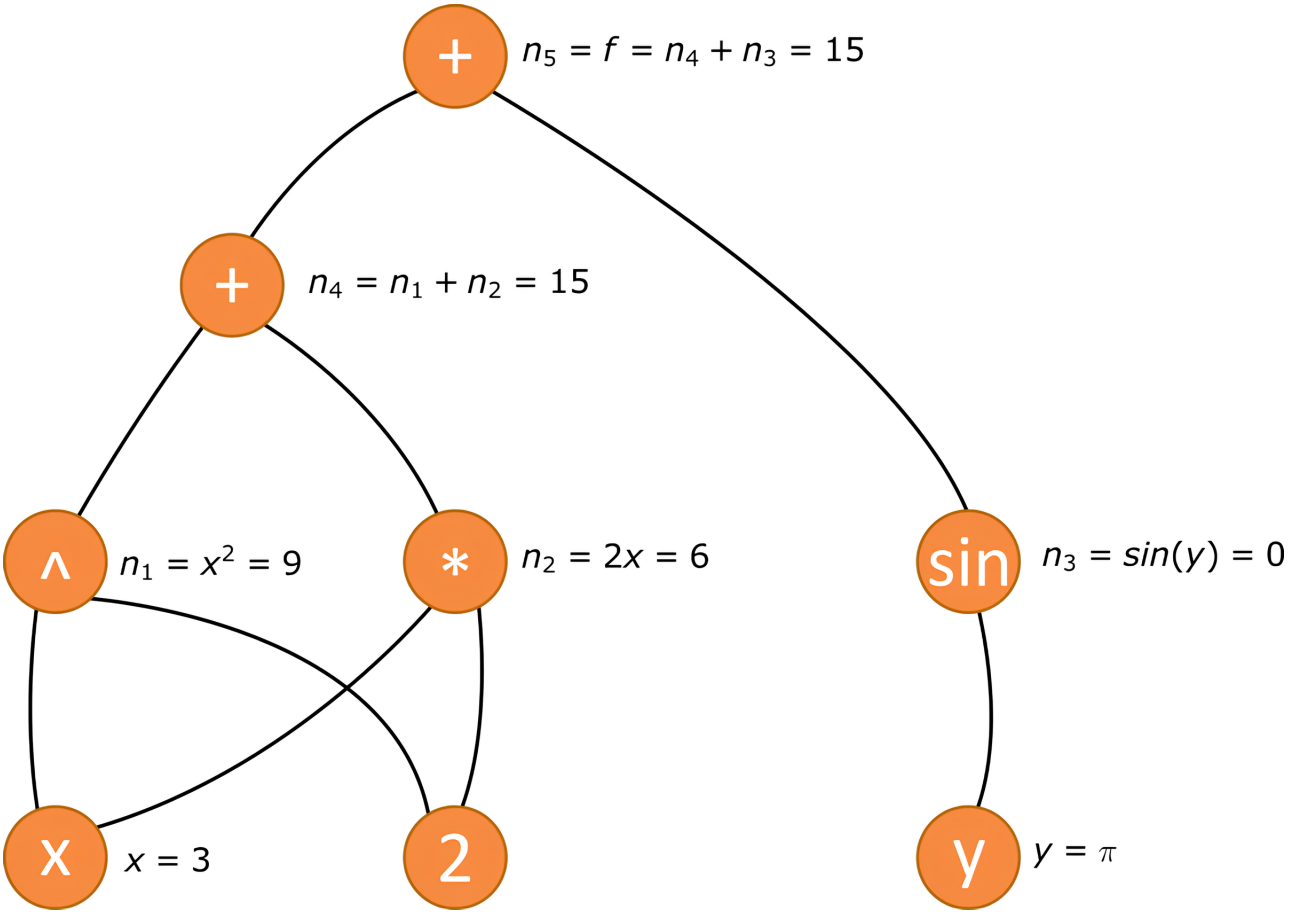

Abbildung 2.5: Berechnung des Funktionswertes $f(x, y) = x^2 + 2x + \sin(y)$ an der Stelle $(x, y) = (3, \pi)$ mittels Berechnungsbaum (Buchner 2024)

Nachdem man alle Funktionswerte hat wird von oben nach unten die partielle Ableitung durch Anwendung der Kettenregel durchgeführt. Wenn ein Knoten mehrere Ausgänge hat werden die Ableitungen summiert, wie man beim Knoten x in Abbildung 2.6 sehen kann.

Da beim Rückwärtsmodus des Berechnungsgraphen nur einmal durchgelaufen wird und die Effizienz unabhängig von der Anzahl der Eingabevariablen bleibt, ist der Rückwärtsmodus bei vielen Eingabevariablen, wie sie in der Optimierung von Parametern im maschinellen Lernen vorkommen können (mit bis zu Millionen von Parametern), der Standard (vgl. Elliott 2018).

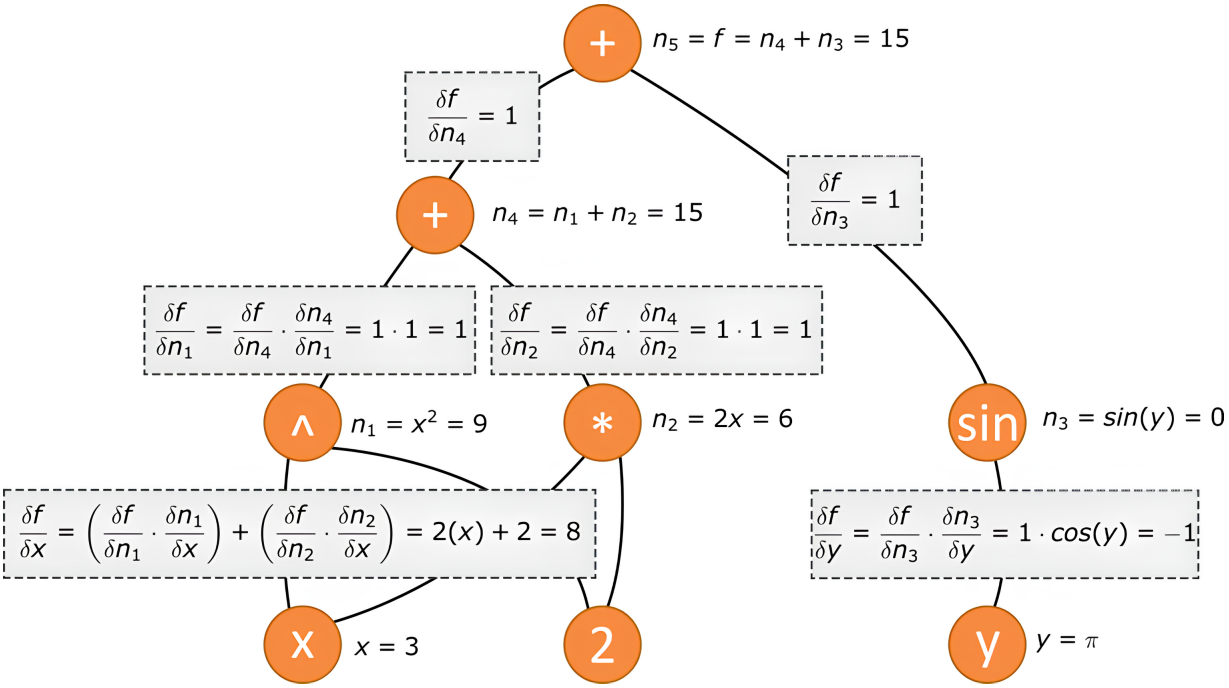

Abbildung 2.6: Berechnung partieller Ableitungen mittels Rückwärtsmodus der automatischen Differenzierung (Buchner 2024)

2.3.3 Anwendungsbeispiel

Zur Veranschaulichung des Unterschieds in der Arbeit mit Tensoren anstelle von anderen Datenstrukturen betrachten wir den Code 2.1:

```
float[,] arrayA = {{1, 2}, {3,4} };
float[,] arrayB = {{5, 6}, {7,8} };

float[,] arraySum = new float[2, 2];
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 2; j++)
    {
        arraySum[i, j] = arrayA[i, j] + arrayB[i, j];
    }
}

var tensorA = new Tensor (arrayA);
var tensorB = new Tensor (arrayB);

var tensorSum = tf.add(tensorA, tensorB);
```

Listing 2.1: Elementweise Addition beim Array und beim Tensor

In diesem wird die folgende Addition durchgeführt:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} \tag{2.2}$$

Die elementweise Addition der Arrays arrayA und arrayB erfolgt durch Iteration über die jeweiligen Elemente mittels verschachtelter Schleifen, wobei die Summe in einem neuen Array arraySum gespeichert wird. Bei der Verwendung von Tensoren ist die Nutzung von Schleifen nicht erforderlich. Die elementweise Addition wird durch die Funktion tf.add() durchgeführt, die für eine effiziente Berechnung optimiert ist. TensorFlow ermöglicht dabei die Nutzung von Hardwarebeschleunigung, wie zum Beispiel durch GPUs oder TPUs, sofern diese verfügbar sind. Die Berechnungen werden automatisch für die verfügbare Hardware optimiert, beispielsweise durch Parallelisierung und geschickte Speicherverwaltung. Beide Ansätze liefern das korrekte Ergebnis, jedoch kann sich die Laufzeit, insbesondere bei größeren Matrizen, erheblich unterscheiden.

2.4 Neuronale Netze

Neuronale Netze (NN) sind Verfahren zur Analyse komplexer Zusammenhänge, die von der Funktionsweise biologischer neuronaler Netze inspiriert sind. Sie bestehen aus miteinander verbundenen einfachen Recheneinheiten (Neuronen), die Informationen gewichtet zusammenfassen und nichtlinear verarbeiten (vgl. Dayhoff und DeLeo 2001; Stern 1996). Durch einen Trainingsprozess, bei dem die Gewichte der Verbindungen anhand von Daten angepasst werden, können NN lernen, vielfältige Aufgaben zu bewältigen. Dazu gehören beispielsweise die Vorhersage von Werten, die Klassifizierung von Objekten, die Annäherung an Funktionen, die Erkennung von Mustern in komplexen Daten oder die Vervollständigung bekannter Muster.

Neuronale Netze finden vielfältige Anwendung, beispielsweise im militärischen Bereich, wo sie zur automatischen Zielerkennung, Flugzeugsteuerung und Optimierung der Triebwerksverbrennung beitragen. In der Medizin spielen sie eine entscheidende Rolle bei der Erkennung von Tumoren in medizinischen Bildern und der Klassifizierung von Zellen

in zytologischen Untersuchungen als bösartig oder normal. In dieser Arbeit konzentrieren wir uns auf die neuronalen Netze die die Regression durchführen, also den Prozess der Vorhersage eines kontinuierlichen Ausgabewertes. Im Kontext der epidemiologischen Modellierung könnte das die Generierung von Parametern für das Modell sein.

Jedes Neuron in einer Schicht ist mit den Neuronen der vorherigen Schicht verbunden, und diese Verbindungen haben Gewichte, die die Stärke der Verbindung zwischen zwei Neuronen repräsentieren. Während des Trainings eines neuronalen Netzes werden diese Gewichte angepasst, um die Leistung des Modells zu verbessern. Die Daten werden von der Eingabeschicht durch die verborgenen Schichten zur Ausgabeschicht weitergeleitet. Jedes Neuron berechnet eine gewichtete Summe seiner Eingaben und wendet dann eine Aktivierungsfunktion an, um die Ausgabe des Neurons zu bestimmen. Die Aktivierungsfunktion führt eine nichtlineare Transformation durch, die es dem neuronalen Netz ermöglicht, komplexe Muster in den Daten zu lernen.

2.4.1 Aufbau und Funktion eines Neurons

In der Abbildung 2.7 wird die Funktionsweise eines einzelnen Neurons dargestellt. Dieses Neuron empfängt Eingangssignale von mehreren Eingabe-Neuronen $a_1, a_2, ..., a_n$, deren jeweilige Signalstärke durch ihre Aktivierungswerte a repräsentiert wird. Jedes Eingabeneuron ist über eine gewichtete Verbindung $w_{j1}, w_{j2}, ..., w_{jn}$ mit dem Neuron verbunden. Diese Gewichte w bestimmen den Einfluss jedes Eingangssignals auf das Neuron. Das Neuron berechnet die gewichtete Summe S_j seiner Eingaben nach der Formel: $\mathbf{S_j} = \sum_{i=0}^{n} w_{ji} a_i$ wobei i von 1 bis n läuft (n ist die Anzahl der Eingabeneuronen). Diese gewichtete Summe S_j wird dann durch eine Aktivierungsfunktion transformiert, um den Aktivierungswert a_j des Neurons zu erzeugen. Im Bild ist dies die logistische Sigmoidfunktion, welche die gewichtete Summe nichtlinear transformiert, damit das Neuron komplexe Beziehungen zwischen den Eingaben lernen kann. Die Formel für die logistische Sigmoidfunktion lautet: $a_j = \frac{1}{1+\exp(-S_j)}$ Obwohl dies im Bild nicht dargestellt ist, kann ein Bias-Term b zur gewichteten Summe hinzugefügt werden, um die Aktivierung des Neurons zu verschieben. Die Formel von S_j ist in diesem Fall: $\mathbf{S_j} = \sum_{i=0}^{n} w_{ji} a_i + b$. Der resultierende Aktivierungswert a_j wird dann an das nächste Neuron oder die Ausgabeschicht des Netzwerks weitergeleitet.

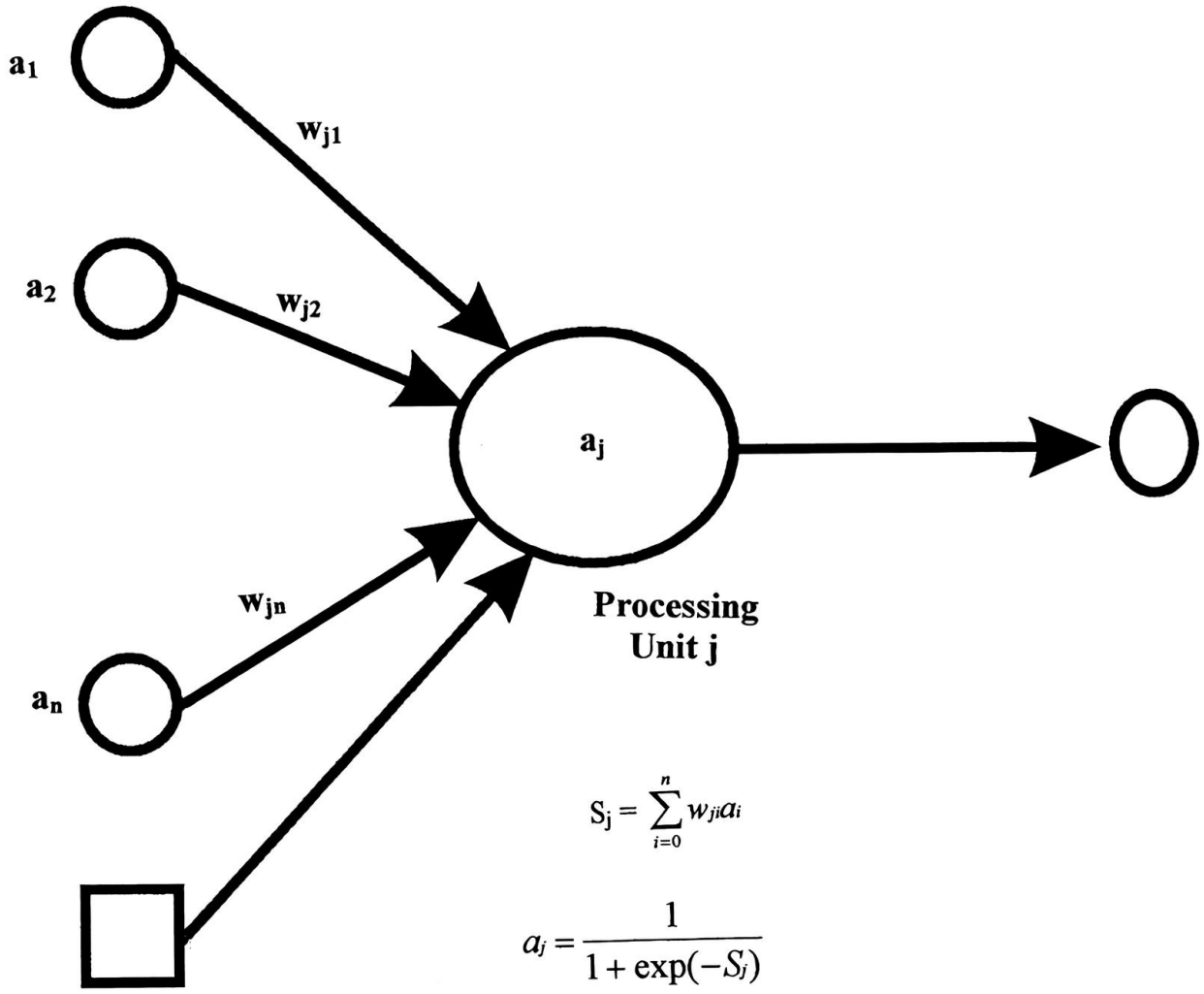

Abbildung 2.7: Berechnungen bei einem Neuron (Dayhoff und DeLeo 2001)

2.4.2 Gradientenbasierte Optimierung in neuronalen Netzen

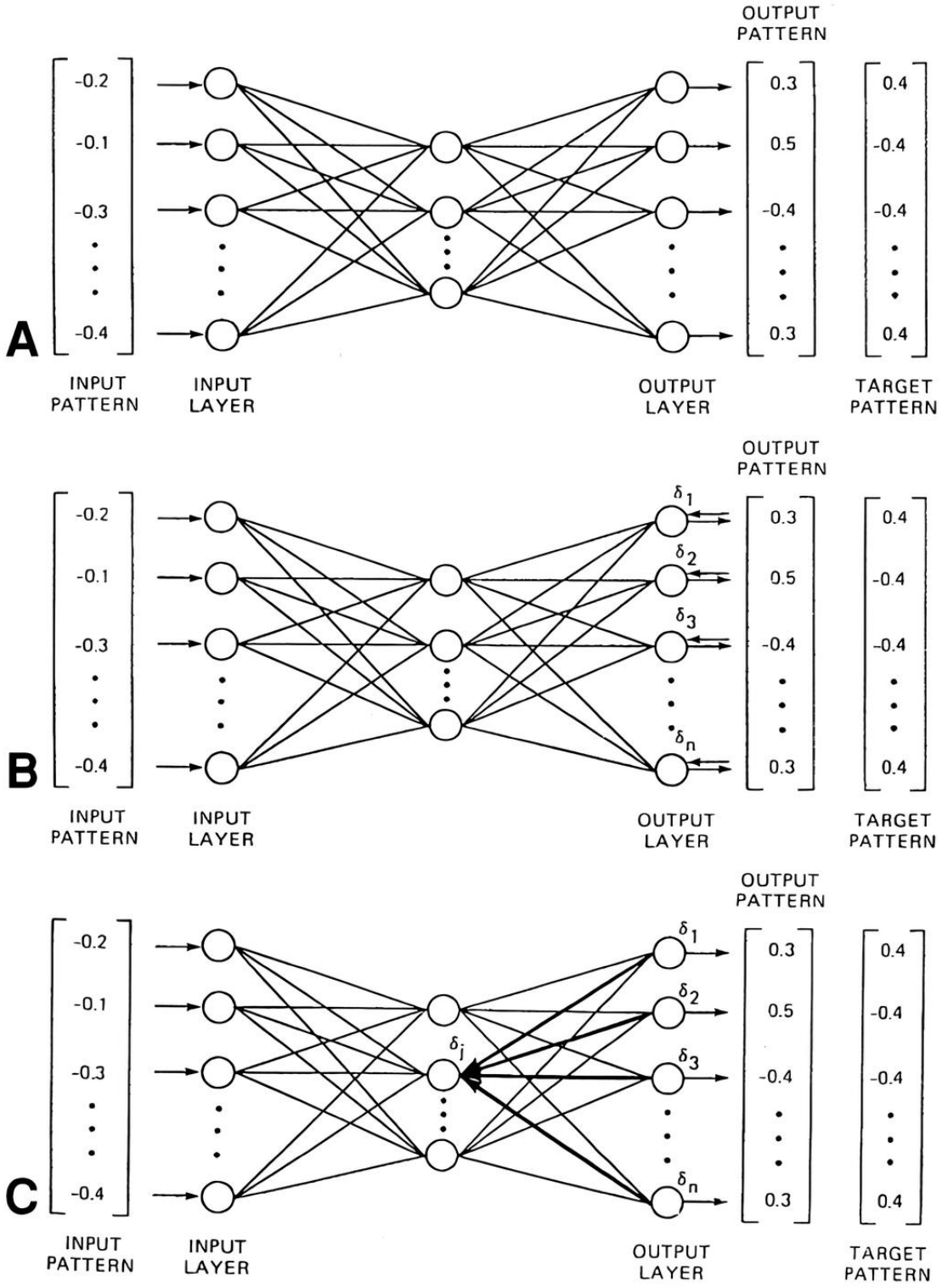

Abbildung 2.8: Phasen der Backpropagation in einem mehrschichtigen neuronalen Netz (A: Vorwärtspropagation, B: Fehlerberechnung, C: Rückwärtspropagation) (Dayhoff und DeLeo 2001)

Neuronale Netze sind zu Beginn nicht in der Lage, spezifische Aufgaben zu lösen. Um ihre volle Leistungsfähigkeit zu erreichen, durchlaufen sie einen Trainingsprozess, der aus drei Phasen besteht. Die drei Phasen werden in der Abbildung 2.8 als A, B und C bezeichnet.

In Phase A, der Vorwärtspropagation, erhält das Netzwerk ein Eingabemuster, dargestellt durch eine Reihe von Zahlen. Diese Zahlen repräsentieren beispielsweise Pixelwerte eines Bildes oder andere Merkmale, die das Netzwerk analysieren soll. Die Eingabeschicht empfängt dieses Muster und leitet es an die nächste Schicht weiter. Die Ausgabeschicht berechnet basierend auf den Eingaben und den aktuellen Gewichten der Verbindungen zwischen den Neuronen eine Ausgabe. Das berechnete Ausgabemuster wird anschließend mit dem Zielmuster, auch Label genannt, verglichen, um den Fehler zu bestimmen.

In Phase B, der Fehlerberechnung, dient das Zielmuster als gewünschte Ausgabe für das gegebene Eingabemuster. Der Fehler zwischen dem tatsächlichen Ausgabemuster und dem Zielmuster wird berechnet und in der Regel durch eine Verlustfunktion, wie den mittleren quadratischen Fehler, auch MSE genannt, bestimmt. Diese Fehlerberechnung wird verwendet, um Anpassungen der Gewichte der Verbindungen zwischen den Neuronen zu berechnen. Die Anpassungen zielen darauf ab, den Fehler zu reduzieren und die Leistung des Netzwerks zu verbessern.

In Phase C, der Rückwärtspropagation, werden die in Phase B berechneten Fehler genutzt, um die Gewichte des neuronalen Netzwerks anzupassen. Mittels automatischer Differenzierung wird für jedes Gewicht die Ableitung der Fehlerfunktion, der sogenannte Gradient, berechnet. Diese Gradienten dienen als Grundlage für den Gradientenabstiegsalgorithmus, welcher dem allgemeinen Gradientenverfahren ähnelt und die Gewichte iterativ optimiert, um den Fehler zu minimieren. Der Gradientenabstieg ist ein Optimierungsverfahren, das die Richtung des steilsten Abstiegs der Fehlerfunktion bestimmt und die Gewichte entsprechend anpasst. Durch wiederholte Anwendung dieses Verfahrens in den Phasen A, B und C, wobei das Netzwerk jedes Mal ein neues Eingabemuster erhält und seine Gewichte basierend auf den berechneten Gradienten anpasst, lernt das Netzwerk, die Beziehung zwischen Eingabemuster und Zielmuster zu erkennen und seine Ausgabe im Laufe der Zeit zu verbessern.

2.4.3 GNN

Ein Graph Neural Network (GNN) ist ein neuronales Netzwerk, das speziell für die Verarbeitung von Graphen entwickelt wurde. Ein Graph ist eine Struktur, die aus Knoten (Objekten) und Kanten (Beziehungen zwischen Objekten) besteht. Ein GNN besteht aus mehreren Schichten, die jeweils die Knoten und Kanten des Graphen verarbeiten.

Der grundlegende Ablauf eines GNN besteht aus einem iterativen Prozess, der als Message Passing bezeichnet wird. In jeder Schicht eines GNN finden folgende Schritte statt:

- **Message Passing:** Jeder Knoten sendet Informationen (Nachrichten) an seine direkten Nachbarn über die Kanten des Graphen.
- **Aggregation:** Jeder Knoten sammelt die Nachrichten seiner Nachbarn und fasst diese Informationen zusammen (aggregiert). Dies kann beispielsweise durch Summieren, Mitteln oder andere Funktionen geschehen.
- **Node Update:** Basierend auf den aggregierten Informationen und seinem eigenen Zustand aktualisiert jeder Knoten seinen Zustand. Dieser neue Zustand dient als Eingabe für die nächste Schicht.

Dieser Prozess wird wiederholt, bis alle Schichten des GNN durchlaufen wurden. Die Ausgabe des GNN kann dann für verschiedene Aufgaben verwendet werden, wie z.B. die Klassifizierung von Knoten oder die Vorhersage von Kanten. In der Epidemiologie können sie helfen Superspreader zu identifizieren oder überprüfen ob Interventionsmaßnahmen wie die Kontaktbeschränkung Auswirkungen auf die Eindämmung von Ausbrüchen hat.

Die Abbildung 2.9 veranschaulicht diesen Prozess für ein GNN mit zwei verborgenen Schichten. Jeder Knoten im Eingabegraphen wird in jeder Schicht aktualisiert, wobei die Farbe der Knoten ihren Zustand repräsentiert. Die ReLU-Aktivierungsfunktion wird nach jeder Schicht angewendet, um Nichtlinearität in das Modell einzuführen.

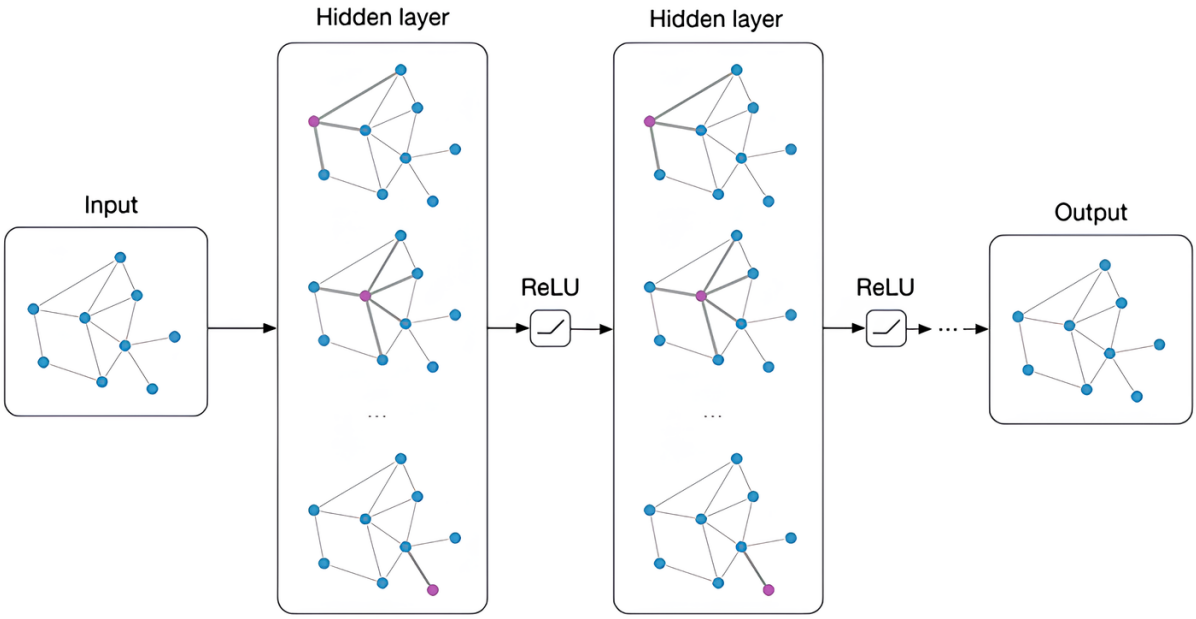

Abbildung 2.9: Darstellung eines Graph Neural Networks mit ReLU-Aktivierung (Kipf 2024)

3 Stand der Forschung

Die Modellierung von Infektionskrankheiten ist seit langem ein etabliertes Forschungsfeld, doch tranditionelle Ansätze wie SIR- und SEIR-Modelle stoßen bei der Abbildung der realen Komplexität von Ausbreitungsprozessen an ihre Grenzen. Angesichts aktueller Ereignisse wie der COVID-19-Pandemie wurden deswegen neue Modelle entwickelt, um ein tieferes Verständnis der Infektionsdynamik und der Wirksamkeit von Interventionsmaßnahmen zu erlangen. In den folgenden Unterkapiteln werden zwei dieser Modelle, GRADABM (vgl. Chopra u. a. 2023) und θ-SEIRHD (vgl. Ivorra u. a. 2020), vorgestellt sowie ihre Funktionsweise und Ergebnisse analysiert. Der Schwerpunkt liegt dabei auf GRADABM, da dieses Modell einen neuen Ansatz für die Epidemiologie bietet.

3.1 GRADABM

Agentenbasierte Modelle haben sich als vielversprechendes Werkzeug zur Modellierung komplexer epidemiologischer Phänomene erwiesen, da sie die Heterogenität von Individuen und deren Interaktionen berücksichtigen können. Allerdings stellt die Kalibrierung solcher Modelle eine große Herausforderung dar, da sie oft eine manuelle Anpassung zahlreicher Parameter erfordert, um das Modell an reale Daten anzupassen. Um diese Herausforderung zu bewältigen, schlagen Chopra u. a. (2023) in ihrer Arbeit Differentiable Agent-based Epidemiology einen neuartigen Ansatz namens GRADABM vor. GRADABM nutzt gradientenbasierte Optimierungstechniken, um die Parameter von ABMs automatisch zu lernen und so die Übereinstimmung zwischen simulierten und beobachteten Daten zu maximieren. Dieser Ansatz ermöglicht eine effiziente und datengetriebene Kalibrierung von ABMs, die insbesondere bei großen und komplexen Modellen von Vorteil ist. Durch die Verwendung von Gradienteninformationen kann GRADABM die Parameter des Modells gezielt anpassen, um die Modellvorhersagen an die beobachteten Daten anzupassen.

Die Implementierung der Agenten in GRADABM unterscheidet sich vom typischen Ansatz. Anstatt jeden Agenten als individuelles Objekt zu implementieren, wird ein Tensor erstellt, der die Zustände aller Agenten des Modells beinhaltet. Jeder Agent wird durch einen Vektor repräsentiert, der seine Eigenschaften enthält: Alter a_j, Krankheitszustand d_j^t und Zeitpunkt der letzten Infektion e_j^t. Der Tensor kombiniert diese Vektoren für alle Agenten und ermöglicht effiziente gradientenbasierte Operationen durch die Verwendung der Sparse Tensor API von PyTorch.

Der Krankheitszustand d_j^t eines Agenten j zum Zeitpunkt t wird durch das SIR-ähnliche Modell SEIRM dargestellt, das die Zustände Susceptible (S, nicht infiziert, aber anfällig), Exposed (E, infiziert, aber noch nicht infektiös), Infectious (I, infiziert und infektiös), Recovered (R, genesen und immun) und Mortality (M, verstorben) umfasst. Das Alter a_j kann einen von neun Werten annehmen, die verschiedene Altersgruppen repräsentieren (0-10, 11-20, ..., 80+). Der Zeitpunkt der letzten Infektion e_j^t gibt an, wann der Agent zuletzt infiziert wurde.

Um die Übertragung einer Infektionskrankheit zu modellieren, verwendet GRADABM einen Kontaktgraphen, der Personen als Knoten und Kontakte als Kanten darstellt. Wie im Artikel von Abueg u. a. (2021) wird der Kontaktgraph für jedes Bezirk separat modelliert und Agenteninteraktionen in mehreren Szenarien dargestellt: Haushalts-, zufällige und arbeitsbezogene Interaktionen. Dieser Graph wird als GNN implementiert und unter Verwendung demografischer Daten erstellt. Er ändert sich in jedem Zeitschritt durch die Interaktionen der Agenten.

Die Übertragungsrate λ, die bestimmt, ob eine Infektion stattfindet, hängt von mehreren Faktoren ab: Der Infektiosität des Erregers zum Zeitpunkt t R^t, der Empfänglichkeit des potenziell Infizierten i S_i, der Übertragbarkeit des Infizierers j T_j und der Zeit seit der letzten Infektion des Infizierers ($\Delta E_j^t = t - e_j^t$). Die Infektiosität, die im Laufe der Zeit variiert, wird durch eine Gamma-Verteilung modelliert (vgl. Abueg u. a. 2021). Die Empfänglichkeit ist altersabhängig (vgl. Hinch u. a. 2021).

Die Wahrscheinlichkeit einer Übertragung q wird wie folgt berechnet:

$$q(d_i^t, d_j^t) = 1 - e^{-\lambda(R, S_i, T_j, \Delta E_j^t)}$$

wobei

$$\lambda(R, S_i, T_j, \Delta E_j^t) = \frac{R S_i T_j}{\hat{I}_i} \int_{\Delta E_j^t - 1}^{\Delta E_j^t} G_\Gamma(u; \mu_j, \sigma_j^2) du$$

die Übertragungsrate in der Interaktion darstellt. $\hat{I}_i$ steht für die erwartete Anzahl von Interaktionen für den Agenten i. Die aggregierte Übertragungswahrscheinlichkeit wird verwendet, um eine Bernoulli-Verteilung zu parametrisieren, die dann entscheidet, ob eine Infektion tatsächlich stattfindet. Eine erfolgreiche Infektion führt zu einer Aktualisierung des Zustands des Agenten i, indem d_i^t und e_j^t aktualisiert werden.

Das Verlaufsmodell beschreibt den Krankheitsverlauf einer infizierten Person. Sobald ein Agent infiziert ist, durchläuft er eine Folge von Krankheitsstadien (SEIRM), die zu Änderungen in seinem Zustand führen. GRADABM folgt dabei einem leicht modifizierten SEIRM-Verlaufsmodell (vgl. Wu u. a. 2020). Die Parameter für das Verlaufsmodell sind die Übergangszeiten zwischen den Stadien und die Todesrate.

3.1.1 Gradientenbasiertes Lernen von ABM-Parametern mit CALIBNN

Für das Training von GRADABM wird ein neuronales Netzwerk namens CALIBNN verwendet, das die Parameter für GRADABM generiert. CALIBNN besteht aus mehreren Schichten die in einen Encoding- und einen Decoding-Prozess unterteilt sind, dabei nutzt es verschiedene Techniken, um die Parameter effektiv zu bestimmen. Der Encoder ist dafür zuständig, die relevanten Informationen aus den Hilfsdaten zu extrahieren und in eine kompakte Darstellung, den Kontextvektor oder Embedding, zu komprimieren. Der Decoder nutzt den Kontextvektor, um Vorhersagen für zukünftige Parameter zu treffen. Die einzelnen Schichten des Encoders und Decoders werden im Folgenden genauer erläutert:

Die Hilfsdaten x_t bis zum Zeitpunkt t_N werden in Vektordaten umgewandelt. Dies geschieht mithilfe einer Gated Recurrent Unit (GRU) (vgl. Zaghloul und Elsayed 2021), einer Art von rekurrentem neuronalen Netzwerk (vgl. DiPietro und Hager 2020). Embeddings ermöglichen es neuronalen Netzen, mit diskreten Kategorien wie Wörtern oder IDs umzugehen. Sie wandeln diese Kategorien in kontinuierliche Vektoren um, die semantische Beziehungen erfassen und vom Netz leichter verarbeitet werden können. Dadurch können neuronale Netze Muster in den Daten erkennen. Die GRU verarbeitet die Eingabesequenz $\left\{\mathbf{x}^t\right\}_{t=t_0}^{t_N}$ und erzeugt eine Sequenz von versteckten Zuständen (Embeddings) $\{\mathbf{h}^t\}_{t=t_0}^{t_N}$: $\{\mathbf{h}^t\}_{t=t_0}^{t_N} = \mathrm{GRU}(\left\{\mathbf{x}^t\right\}_{t=t_0}^{t_N})$. Die Gated Recurrent Unit (GRU), deren Architektur in Abbildung 3.1 dargestellt ist, stellt eine Weiterentwicklung rekurrenter neuronaler Netzwerke (RNNs) dar (vgl. DiPietro und Hager 2020). Ein RNN ist ein neuronales

Netzwerk, das speziell für die Verarbeitung von Sequenzdaten entwickelt wurde. Im Gegensatz zu einfachen neuronalen Netzwerken, die Eingaben unabhängig voneinander behandeln, können RNNs Informationen über vorherige Eingaben beachten. Diese Fähigkeit ermöglicht es RNNs, Abhängigkeiten in Sequenzen zu erkennen und zu nutzen, was sie besonders nützlich für Aufgaben wie Sprachverarbeitung, Zeitreihenanalyse und maschinelles Übersetzen macht. Allerdings haben RNNs Schwierigkeiten, Langzeitabhängigkeiten zu lernen, weil der Gradientenfluss während des Trainings exponentiell abnehmen kann. Die GRU löst dieses Problem durch die Einführung von Gating-Mechanismen, die den Informationsfluss innerhalb des Netzwerks steuern. Eine GRU besteht aus mehreren Komponenten, darunter das Reset-Gate und das Update-Gate, die die Informationsweitergabe über verschiedene Zeitschritte hinweg regulieren. Das Reset-Gate entscheidet,

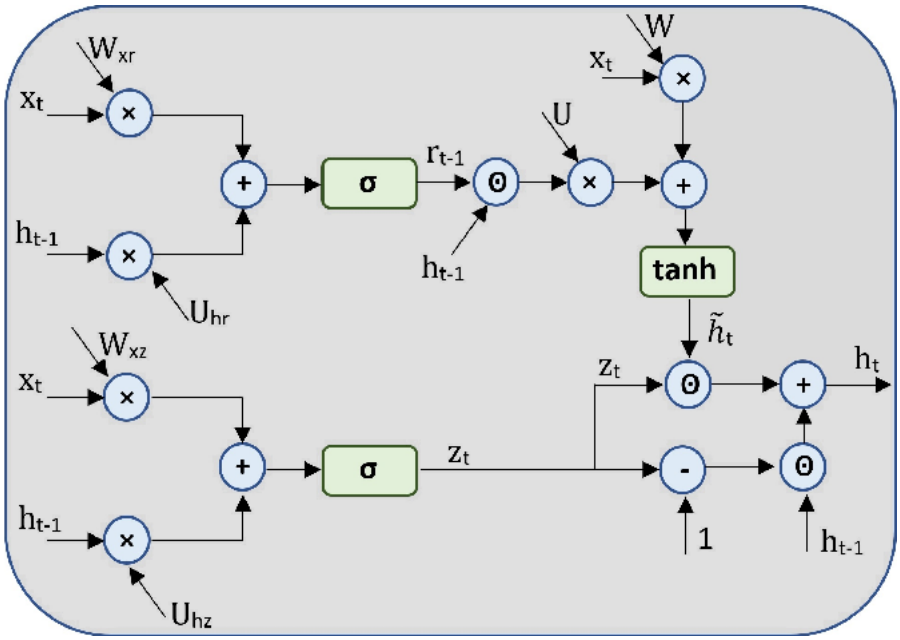

Abbildung 3.1: Gated Recurrent Unit (Zaghloul und Elsayed 2021)

wie viel vom vorherigen Zustand h_{t-1} ignoriert werden soll. Es wird berechnet durch $r_t = \sigma(W_{xr} \cdot x_t + U_{hr} \cdot h_{t-1})$. Hierbei ist r_t das Reset-Gate, x_t die aktuelle Eingabe, $h_t - 1$ der vorherige Zustand W_{xr} die Gewichtsmatrix für die Eingabe und U_{hr} die Gewichtsmatrix für den vorherigen Zustand. Die Sigmoid-Funktion σ sorgt dafür, dass der Wert des Gates zwischen 0 und 1 liegt. Wenn das Reset-Gate nahe bei 0 ist, wird der vorherige Zustand weitgehend ignoriert, was es dem Netzwerk ermöglicht, neue Informationen effizient zu integrieren. Das Update-Gate steuert, wie viel des neuen Zustands in den endgültigen Zustand übernommen wird. $z_t = \sigma(W_{xz} \cdot x_t + U_{hz} \cdot h_{t-1})$ Hierbei ist z_t

das Update-Gate, mit ähnlichen Gewichtsmatrizen wie beim Reset-Gate. Ein hoher Wert des Update-Gates bedeutet, dass der neue Zustand den vorherigen Zustand ersetzt. Dies erlaubt dem Modell, vergangene Informationen beizubehalten oder zu überschreiben. Die Berechnung des neuen Zustands erfolgt durch eine Kombination der aktuellen Eingabe und des durch das Reset-Gate modulierten vorherigen Zustands, die durch eine hyperbolische Tangens-Funktion transformiert wird. $\tilde{h_t} = \tanh(W \cdot x_t + U \cdot (r_t \circ h_{t-1}))$ Hierbei ist $\tilde{h_t}$ der neue Kandidaten-Zustand, $\circ$ steht für das elementweise Produkt und tanh ist die hyperbolische Tangens-Funktion, die den Wertebereich auf [-1,1] beschränkt. Der endgültige Zustand der GRU wird dann wie folgt berechnet: $h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$. GRUs können sowohl für kurze als auch lange Sequenzen verwendet werden, da sie flexibel an die jeweilige Länge der Eingabedaten angepasst werden können. Ihre Stärke liegt in der Balance zwischen Effizienz und Leistungsfähigkeit.

Auf das Embedding folgt ein Self-Attention-Layer, um die relevantesten Informationen zu extrahieren und langfristige Abhängigkeiten zwischen den Datenpunkten zu berücksichtigen (vgl. Vaswani u. a. 2017). Dieser Ansatz ermöglicht es CALIBNN, sich auf die für die Generierung der Parameter wesentlichen Teile der eingebetteten Daten zu konzentrieren, was die Genauigkeit der Simulation verbessert und Überanpassung reduziert. Außerdem kann Self-Attention langfristige Abhängigkeiten zwischen den Datenpunkten erfassen, die für die Modellierung der Ausbreitung von Infektionskrankheiten wie COVID-19 und Grippe von Bedeutung sind, da diese über längere Zeiträume übertragen werden können. Der Self-Attention-Layer berechnet die Aufmerksamkeit λ^t für jeden Zeitpunkt t: $\{\lambda^t\}_{t=t_0}^{t_N}$ =Self-Atten($\{h^t\}_t = t_0^{t_N}$) Den Ablauf zur Berechnung des Attentions wird in Abbildung 3.2 dargestellt. Die versteckten Zustände h_t werden verwendet um die Queries, Keys und Values zu berechnen. Die Queries Q werden durch eine lineare Transformation der Eingabesequenzen h_t berechnet. Dies erfolgt durch Multiplikation mit einer Gewichtsmatrix W_Q: $Q = W_Q \cdot H$, wobei H die Matrix der verborgenen Zustände h_t über alle Zeitpunkte t darstellt. Die Queries sind dafür da um zu bestimmen welche anderen Zustände in der Sequenz für das Verständnis dieses Zustands am wichtigsten sind. Die Keys K werden durch eine lineare Transformation von h_t mit der Gewichtsmatrix W_K berechnet: $K = W_K \cdot H$. Der Key-Vektor, fasst die enhaltenen Informationen im Zustand zusammen. Die Values V werden durch eine lineare Transformation von h_t mit der Gewichtsmatrix W_V berechnet: $V = W_V \cdot H$. Der Value enthält die tatsächlichen Informationen, die für jeden Teil der Eingabe gespeichert sind. Der Self-Attention-Mechanismus berechnet die Ähnlichkeit zwischen dem Query-Vektor eines Zustands und den Key-Vektoren aller Zustände. Diese Ähnlichkeit gibt an, wie relevant ein Zustand für einen

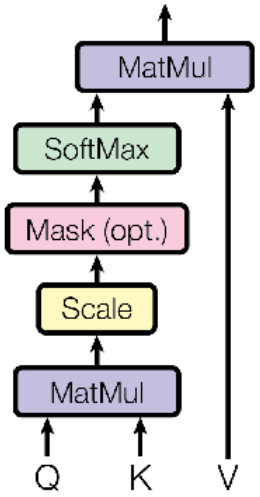

Abbildung 3.2: Self-Attention-Mechanismus (Vaswani u. a. 2017)

anderen Zustand ist. Die Value-Vektoren der relevanten Zustände werden dann kombiniert, um einen neuen Vektor zu erzeugen, der den Kontext des ursprünglichen Zustands repräsentiert: $U = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$. Der erste Schritt im Prozess ist die Berechnung des Produkts der Queries Q und der Transponierten der Keys K^T. Um Stabilität zu gewährleisten und extrem große Werte zu vermeiden, wird das Ergebnis durch die Quadratwurzel der Dimension der Keys $\sqrt{d_k}$ geteilt. Der Quotient wird durch die Softmax-Funktion geleitet, die diesen in Wahrscheinlichkeiten umwandelt. Die Softmax-Funktion sorgt dafür, dass die Summen der Wahrscheinlichkeiten gleich 1 ist. Schließlich werden die berechneten Attention-Wahrscheinlichkeiten mit den Values V multipliziert. Diese Matrixmultiplikation erzeugt die gewichtete Summe der Values, wobei die Gewichte den berechneten Attention-Wahrscheinlichkeiten entsprechen. Das Ergebnis dieser Berechnung ist die Aufmerksamkeitsmatrix U, die die Beziehungen zwischen allen Paaren von Zeitschritten in der Sequenz erfasst. Jede Zeile repräsentiert die Aufmerksamkeitsgewichte λ für einen bestimmten Zeitschritt. Diese Gewichte geben an, wie viel Relevanz jeder andere Zeitschritt bei der Berechnung des endgültigen Embeddings für diesen bestimmten Zeitschritt erhält. Der Kontextvektor u wird berechnet, indem die Aufmerksamkeitsgewichte λ mit den entsprechenden versteckten Zuständen h multipliziert und über alle Zeitschritte summiert werden: $u^{t_0:t_N} = \sum_{t=t_0}^{t_N} \lambda_t h_t$. Nachdem das Modell den Kontextvektor $u^{t_0:t_N}$ aus den Eingabedaten extrahiert hat, wird dieser als Eingabe für einen weiteren GRU verwendet, der als Decoder arbeitet. Der Decoder erhält zusätzlich eine

Positionskodierung τ^k, die angibt, wie viele Tage in der Zukunft vorhergesagt werden sollen. τ^k ist ein einfacher Gleitkommawert zwischen 0 und 1. Die Decoder-GRU verarbeitet den Kontextvektor und die Positionskodierung, um eine Folge von versteckten Zuständen $\{o^t\}_{t=t_1}^{t_K}$ zu erhalten. Diese Zustände repräsentieren die dekodierte Information zu jedem zukünftigen Zeitpunkt t: $\{\mathbf{o}^t\}_{t=t_1}^{t_K} = \text{GRU}(\{\tau^k\}_{k=1}^{K}; u^{t_0:t_N})$. Damit CALIBNN nicht nur Parameter zum aktuellen Zeitpunkt t sondern auch für zukünftige Zeitschritte diese generieren kann wird die Ausgabe des Decoders durch ein Feedforward-Netzwerk (FFN) geleitet, um die endgültigen Vorhersagen für die zukünftigen Werte zu erhalten. Chopra u. a. (2023) stellten fest, dass die direkte Verwendung der Ausgabe des neuronalen Netzes $\text{FFN}(o^t)$ als Parameter für den Simulator zu Optimierungsproblemen führen kann. Daher wird die Ausgabe des FFN mithilfe einer Sigmoid-Funktion und der oberen und unteren Grenzen der Parameterwerte begrenzt: $\theta^t = \theta_L + (\theta_U - \theta_L) \cdot \sigma(\text{FFN}(o^t))$, wobei θ_L die untere Grenze des Parameters und θ_U die obere Grenze des Parameters ist.

In jeder Trainingsiteration wird CALIBNN einmal durchlaufen, während GRADABM K Schritte simuliert. Abbildung 3.3 zeigt eine visuelle Darstellung des Trainingsprozesses, der aus vier Phasen besteht:

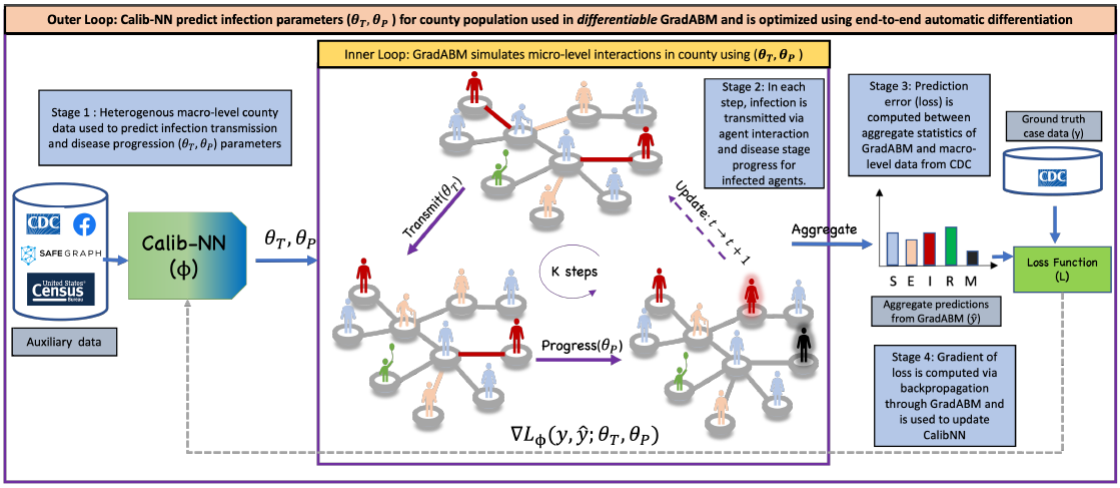

Abbildung 3.3: Traingsdurchlauf von GRADABM (Chopra u. a. 2023)

Phase 1: CALIBNN erhält im Schritt w die Netzwerkparameter ϕ^w und Hilfsdaten (D) als Eingabe. Die Hilfsdaten können demografische Informationen, Kontaktdaten oder andere relevante Informationen sein, die CALIBNN dabei helfen, realistische Parameter zu generieren. Aus diesen Eingaben erzeugt CALIBNN die epidemiologischen Parameter (θ_T^t, θ_P^t), die an GRADABM weitergegeben werden.

Phase 2: GRADABM führt mit den empfangenen Parametern (θ_T^t, θ_P^t) eine K-Schritte-Simulation durch, die die Ausbreitung der Krankheit in der Bevölkerung modelliert.

Phase 3: Die Ausgabe von GRADABM wird aggregiert, um die Gesamtzahl an Todesfällen zu berechnen. Dieser Wert wird mit den realen Daten verglichen und der Verlust wird berechnet indem eine Kostenfunktion, in diesem Fall Mean-Squared Error verwendet wird. $\mathcal{L}\left(\hat{y}^w, y; \left(\theta_T^t, \theta_P^t\right)^w\right) = \text{MSE}\left(\hat{y}^w, y\right)$, wobei y die realen Daten darstellt, die die Anzahl an Todesfällen darstellt , $\hat{y}^w$ die von dem epidemiologischen Modell bei dem w-ten Trainingsschritt vorhergesagte Anzahl an Todesfällen mit den Eingangsparametern $\left(\theta_T^t, \theta_P^t\right)^w\right)$ darstellt und $\mathcal{L}$ die Notation für die Kostenfunktion ist

Phase 4: Nachdem der Verlust berechnet wurde, wird die automatische Differenzierung verwendet, um den Gradienten zur Minimierung des Verlustes abzuschätzen. Der Gradient wird über die K Schritte von GRADABM und CALIBNN berechnet. Dann wird das Backpropagation-Verfahren über die Parameter des neuronalen Netzwerks CALIBNN (ϕ) ausgeführt. Sobald der Gradient bestimmt wurde, werden die Parameter von CALIBNN wie folgt aktualisiert:

$$\phi^{w+1} = \phi^w - \alpha \frac{\partial \mathcal{L}\left(\hat{y}^w, y; \left(\theta_T^t, \theta_P^t\right)^w\right)}{\partial \phi}$$

Hierbei ist α die Lernrate, $\left(\theta_T^t, \theta_P^t\right)^w = f\left(D; \phi^w\right)$ mit CALIBNN als f. $\hat{y}^w$ wird durch Aufrufen von GRADABM mit K Schritten mit den unveränderten Parametern $\left(\theta_T^t, \theta_P^t\right)^w$ berechnet.

Diese vier Phasen werden wiederholt, bis ein akzeptabler Verlust erreicht ist, also bis die vorhergesagte Anzahl an Todesfällen gut mit den realen Daten übereinstimmt.

Um das CALIBNN-Modell trainieren zu können, ist es notwendig, dass die gesamte Simulation differenzierbar ist, also dass Gradienten für alle Komponenten berechnet werden können. Die Bestimmung einer Infektion stellt jedoch eine Herausforderung dar, da eine Infektion nur die diskreten Werte 1 (infiziert) oder 0 (nicht infiziert) annehmen kann. Um dieses Problem zu lösen, wird die Gumbel-Softmax Reparametrisierung verwendet (vgl. Jang u. a. 2016). Die Gumbel-Softmax-Reparametrisierung ist eine Technik, die verwendet wird, wenn diskrete Entscheidungen stochastisch sind und wenn eine differenzierbare Approximation benötigt wird, um den Gradienten Fluss in neuronalen Netzen zu erhalten. Die Abbildung 3.4 zeigt den Prozess des GSR.

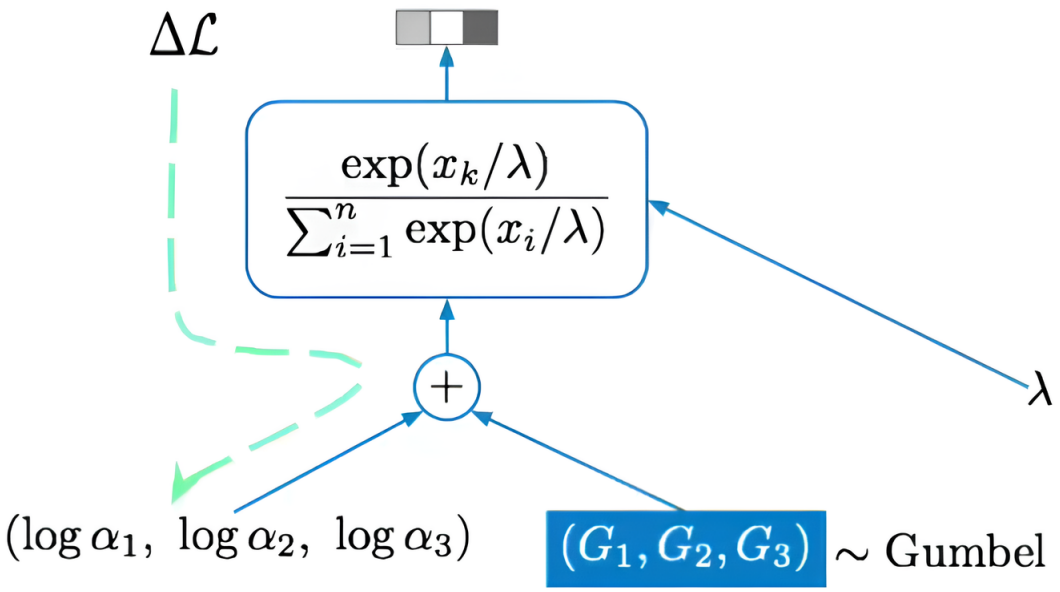

Abbildung 3.4: Gumbel Softmax Reparametrisierung (Jang u. a. 2024)

- **Berechnung der Logits:** Zuerst werden die natürlichen Logarithmen (Logits) der Wahrscheinlichkeiten für jedes mögliche Ereignis im Ereignisraum berechnet.
- **Generierung des Gumbel-Rauschens:** Für jedes Logit wird Gumbel-Rauschen erzeugt, indem ein Wert x_i aus einer Gleichverteilung zwischen 0 und 1 gezogen wird. Dieser Wert wird dann auf die Weise $-\log(-\log(x_i))$ transformiert. Das Ergebnis ist ein Rauschwert G_i für jedes Logit.
- **Hinzufügen des Gumbel-Rauschens zu den Logits:** Das Gumbel-Rauschen G_i wird zu den Logits hinzugefügt. Dies führt dazu, dass die Logits durch das Rauschen gestört werden und die endgültige Verteilung stochastisch beeinflusst wird.
- **Anwendung der Softmax-Funktion:** Auf die verrauschten Logits wird die Softmax-Funktion angewendet. Diese Funktion wandelt die Logits in eine Wahrscheinlichkeitsverteilung um, bei der jedes Ereignis eine Wahrscheinlichkeit zwischen 0 und 1 hat. Der Parameter Lambda (auch als Temperatur bezeichnet) beeinflusst, wie stark sich das größte Logit an 1 und alle anderen an 0 annähern sollen.

3.1.2 Testergebnisse

Um zu beweisen, dass GRADABM nicht nur schneller als nicht differenzierbare ABMs arbeitet, sondern auch bessere Ergebnisse liefert, wurde die Genauigkeit von GRADABM mit der von PC-ABM (vgl. Abueg u. a. 2021) und herkömmlichen ABMs (vgl. Romero-Brufau u. a. 2021), in Abbildung 3.5 als Vanilla-ABM bezeichnet, verglichen. Zum Vergleich werden die Standardmetriken von Tabataba u. a. (2017) verwendet. Nach den Vorgaben von For the Influenza Forecasting Contest Working Group u. a. (2016), einer Behörde in der USA, die für die Überwachung von Krankheiten und die Prävention von Krankheitsausbrüchen zuständig ist werden Vorhersagen nur im Zeitraum 1 bis 4 Wochen gemacht. Die Auswertung beider Krankheiten geht in 10 Landkreisen über 4 Monate.

	COVID-19			Influenza		
Model	ND	RMSE	MAE	ND	RMSE	MAE
Vanilla-ABM [48]	8.75	689.92	270.13	0.57	2.03	1.72
PC-ABM [5]	2.21 ± 1.36	121.87 ± 63.97	68.20 ± 41.84	0.59 ± 0.02	2.17 ± 0.05	1.77 ± 0.05
GRADABM	$\mathbf{0.97 \pm 0.18}$	$\mathbf{50.99 \pm 12.12}$	$\mathbf{30.02 \pm 5.60}$	$\mathbf{0.41 \pm 0.02}$	$\mathbf{1.47 \pm 0.06}$	$\mathbf{1.22 \pm 0.06}$

Abbildung 3.5: Qualität der Vorhersagen bei 5 Durchläufen mit COVID-19 und Influenza (Chopra u. a. 2023)

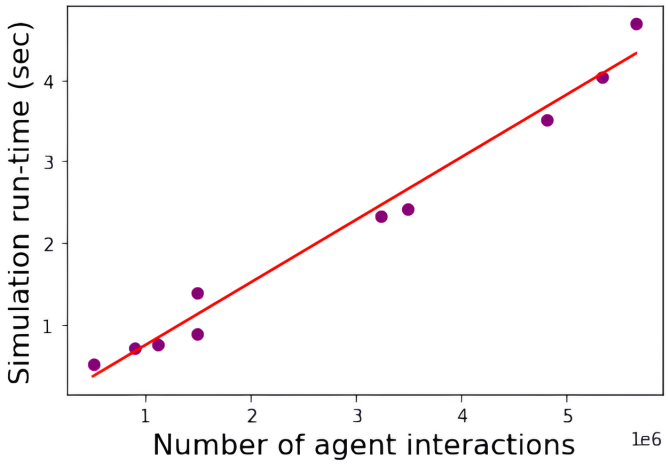

Abbildung 3.6: Darstellung der Steigung der Zeit abhängig von den Interaktionen der Agenten (Chopra u. a. 2023)

In Abbildung 3.6 ist zu sehen dass GRADABM sowohl bei COVID-19 als auch bei Influenza bei jeder Kostenmetrik die anderen ABMs übertrifft. GRADABM ist immer von

8 bis 12 mal so gut wie Vanilla-ABM. Der Grund dafür sind die 2 wichtigen Features von GRADABM. Gradientenbasierte Kalibrierung und Verwendung von heterogenen Datenquellen über CALIBNN. Nicht nur bei der Qualität der Ergebnisse hat GRADABM gepunktet sondern auch bei der Skalierung. Wie man in Abbildung 3.6 sehen kann skaliert die Laufzeit von GRADABM linear zu den Interaktionen der Agenten. Dieses Verhalten ist sehr beeindruckend wenn man ihn mit dem exponentiellem Wachstum von nicht differenzierbaren Agenten basierten Modellen vergleicht.

3.2 θ-SEIRHD

Nachdem die COVID-19-Pandemie ausbrach, hat sich das Virus rasch über die ganze Welt ausgebreitet, und die Notwendigkeit, seine Dynamik zu verstehen und vorherzusagen, wurde immer dringlicher. In diesem Kontext wurde das SEIRHD-Modell (Susceptible-Exposed-Infected-Recovered-Hospitalized-Dead) entwickelt (vgl. Ivorra u. a. 2020). Es baut auf der grundlegenden SIR Struktur auf, berücksichtigt aber zusätzlich die Inkubationszeit (Exposed), die Hospitalisierung (Hospitalized) und die Sterblichkeit (Dead) der Krankheit. Außerdem wurde das Modell erweitert, um unerkannte Infektionen zu berücksichtigen, da diese einen erheblichen Einfluss auf die Ausbreitung des Virus haben können.

3.2.1 Kompartments

Das θ-SEIRHD-Modell berücksichtigt mehrere epidemiologische Zustände. Susceptible (S) bezeichnet anfällige Personen, die noch nicht mit dem Virus infiziert sind. Exposed (E) bezieht sich auf exponierte Personen, die sich in der Inkubationszeit befinden. Sie zeigen noch keine Anzeichen, können aber andere Menschen mit geringerer Wahrscheinlichkeit infizieren. Infectious (I) umfasst infizierte Personen, die Symptome zeigen und das Virus auf andere übertragen können. Diese Personen können von den Behörden entdeckt und hospitalisiert (H) oder weiterhin unentdeckt bleiben. Infectious undetected (I_u) beschreibt unerkannt infizierte Personen, die Symptome zeigen, aber noch nicht erkannt und isoliert wurden. Hospitalized recovered (H_R) umfasst Personen, die ins Krankenhaus eingeliefert wurden und sich erholen. Hospitalized dead (H_D) bezeichnet Personen, die ins Krankenhaus eingeliefert wurden und versterben. Recovered (R) umfasst genesene Personen, die nicht mehr infektiös sind und eine natürliche Immunität gegen das Virus

entwickelt haben. Innerhalb dieser Kategorie gibt es Personen, die zuvor als infektiös erkannt wurden (R_d). Wenn eine Person in dieses Kompartment eintritt, bleibt sie für eine Genesungszeit von durchschnittlich C_o Tagen im Krankenhaus. Es gibt auch Personen, die zuvor infektiös, aber unerkannt waren (R_u). Dead (D) beschreibt Personen, die an COVID-19 gestorben sind.

3.2.2 Maßnahmen zur Kontrolle der Ausbreitung

Um die Pandemie einzudämmen, können Behörden verschiedene Kontrollmaßnahmen ergreifen (vgl. Chen u. a. 2020):

Infizierte Personen werden isoliert, um die Übertragung auf andere zu verhindern, wobei nur medizinisches Personal Kontakt zu ihnen hat, allerdings mit dem Risiko einer Ansteckung. Isolierte Patienten erhalten eine angemessene medizinische Behandlung, die die COVID-19-Sterblichkeitsrate senkt.

Die Bewegungsfreiheit von Personen im Umfeld einer infizierten Person wird eingeschränkt und kontrolliert zum Beispiel durch Gesundheitskontrollen an Flughäfen, um die Verbreitung der Krankheit durch potenziell Infizierte zu verhindern.

Eine Kontaktverfolgung, potenziell infektiöse Kontakte werden identifiziert, also diejenigen die eine Person infiziert haben oder COVID-19 auf andere übertragen haben könnten. Eine Erhöhung der Anzahl der Tests kann den Prozentsatz der erkannten Infektionen steigern.

Die Anzahl der verfügbaren Krankenhausbetten und des medizinischen Personals zur Erkennung und Behandlung von Betroffenen wird erhöht, was zu einer Verkürzung der infektiösen Phase führt.

Das θ-SEIRHD-Modell kann die Auswirkungen dieser Maßnahmen berücksichtigen, indem die Raten, die die Übergänge zwischen den Kompartments beschreiben, angepasst werden, um den Effekt der Maßnahmen widerzuspiegeln.

3.2.3 Übertragungs- und Verlaufsmodell

Das Übertragungs- und Verlaufsmodell ist ein deterministisches Kompartmentmodell, das die zeitliche Veränderung der Anzahl von Individuen in verschiedenen Kompartments während einer COVID-19-Ausbreitung beschreibt. Es geht davon aus, dass sich

alle Individuen einer Population in einem der definierten Kompartments befinden und die Übergänge zwischen diesen Kompartments durch ein System gewöhnlicher Differentialgleichungen modelliert werden können. Diese Gleichungen berücksichtigen die Anzahl der Individuen in relevanten Kompartments sowie die spezifischen Übergangsraten zwischen ihnen. Die Modellparameter können an spezifische Daten angepasst werden, um die Ausbreitung von COVID-19 unter verschiedenen Bedingungen und in verschiedenen Regionen zu simulieren. Dabei wird angenommen, dass die Bevölkerung innerhalb eines Gebiets homogen verteilt ist.

Eine visuelle Darstellung des Modells ist in der Abbildung 3.7 zu sehen. Durch den deterministischen Ansatz um die Ausbreitung von COVID-19 innerhalb eines bestimmten Territoriums zu simulieren erhält man Vorteile wie geringere Rechenkomplexität und bessere Kalibrierung der Modellparameter. Stochastische Modelle sind oft schwieriger zu analysieren und erfordern mehr Daten für die Kalibrierung. Die Simulationen beginnen typischerweise mit einer Bevölkerung, die nur anfällige Personen umfasst. Infizierte Personen werden zu Beginn der Simulation in die entsprechenden Kompartments eingeteilt. Die Simulation wird gestoppt, wenn keine infizierten Personen mehr vorhanden sind oder das Ende des Simulationszeitraums erreicht ist. Durch die Anpassung des Modells an reale Daten können die Modellparameter kalibriert und validiert werden. Dies ermöglicht es, die Wirksamkeit von Kontrollmaßnahmen zu berechnen und die Ausbreitung der Krankheit vorherzusagen. Insbesondere für politische Entscheidungsträger können solche Modelle wertvolle Einblicke in die Auswirkungen verschiedener Interventionsstrategien bieten.

Die in den Tabellen 3.1, 3.2 und 3.3 aufgeführten Variablen und Parameter quantifizieren die demografische, epidemiologische und verhaltensbezogenen Faktoren des Modells.

Die darauf folgenden Differentialgleichungen, die auf diesen basieren, beschreiben die zeitliche Entwicklung der verschiedenen Kompartments des θ-SEIRHD-Modells. Sie erfassen die Übergänge zwischen den verschiedenen Kompartments und berücksichtigen dabei Faktoren wie Übertragungsraten, Genesungsraten, Sterblichkeitsraten und den Einfluss von Kontrollmaßnahmen.

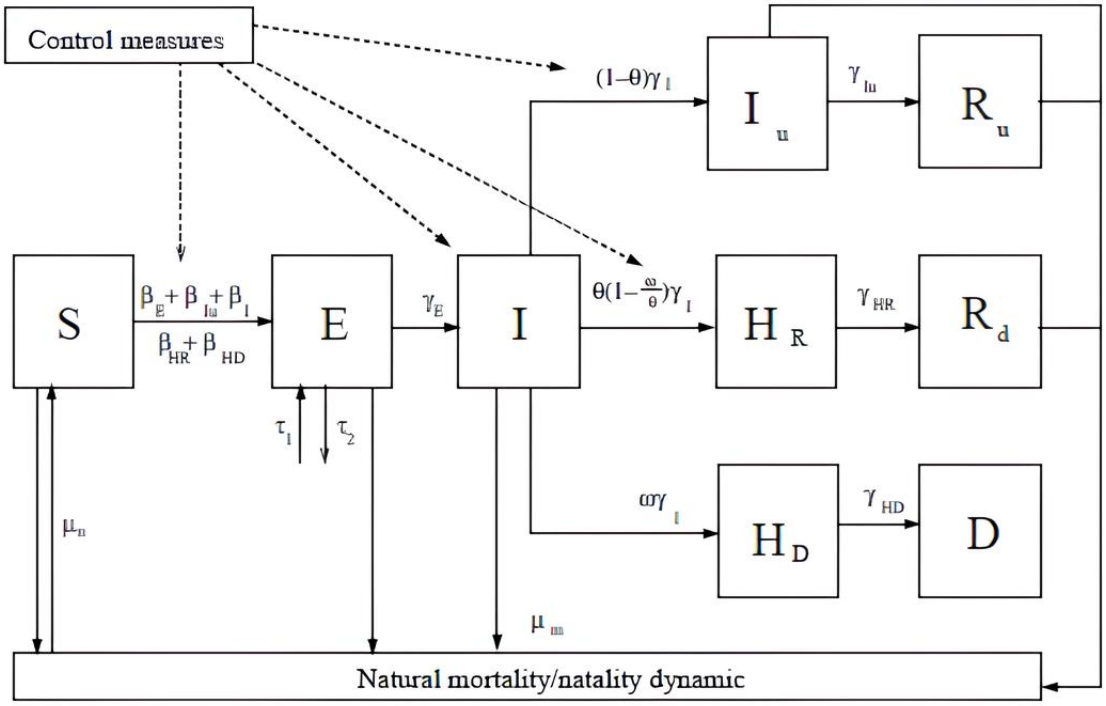

Abbildung 3.7: Diagramm des θ-SEIRHD-Modells (Ivorra u. a. 2020)

Kategorie	Parameter	Beschreibung
Demografische Parameter	$N^{(i)}$	Die Anzahl der Menschen im Territorium i vor dem Beginn der Pandemie.
	$\mu_n^{(i)}$	Die Geburtenrate im Territorium i. Sie gibt an, wie viele Geburten am Tag stattfinden.
	$\mu_m^{(i)}$	Die Sterberate im Territorium i. Sie gibt an, wie viele Todesfälle am Tag stattfinden.
Epidemiologische Parameter	$\omega^{(i)}(t)$	Die Fallsterblichkeitsrate im Territorium i zum Zeitpunkt t. Sie gibt den Anteil der Todesfälle im Vergleich zur Gesamtzahl der Infektionsfälle an. Hierbei sind $\underline{\omega}^{(i)}$ und $\overline{\omega}^{(i)}$ die minimalen und maximalen Fallsterblichkeitsraten für das Territorium i.
	$\theta^{(i)}(t)$	Der Anteil der infizierten Personen, die von den Behörden im Territorium i zum Zeitpunkt t entdeckt und dokumentiert werden. Einfachheitshalber wird angenommen, dass alle COVID-19-bedingten Todesfälle entdeckt und gemeldet werden, sodass $\theta^{(i)}(t) \geq \omega^{(i)}$.
Übertragungsraten	$\beta_E^{(i)}$	Die Übertragungsrate einer Person im Zustand E im Territorium i.
	$\beta_I^{(i)}$	Die Übertragungsrate einer Person im Zustand I im Territorium i.
	$\beta_{I_\mathrm{u}}^{(i)}$	Die Übertragungsrate einer Person im Zustand I_u im Territorium i.
	$\beta_{H_R}^{(i)}$	Die Übertragungsrate einer Person im Zustand H_R im Territorium i.
	$\beta_{H_D}^{(i)}$	Die Übertragungsrate einer Person im Zustand H_D im Territorium i.
	$\beta_{I_\mathrm{u}}^{(i)}(\theta)$	Die Übertragungsrate einer Person im Zustand I_u im Territorium i, wobei der Anteil der infizierten Personen, die entdeckt werden, berücksichtigt wird.

Tabelle 3.1: Zusammenfassung der Variablen und Parameter im θ-SEIRHD-Modell - Teil 1

Kategorie	Parameter	Beschreibung
Übergangsraten	γ_E	Die Übergangsrate vom Zustand E zum Zustand I. Diese Rate ist für alle Territorien gleich.
	$\gamma_I^{(i)}(t)$	Die Übergangsrate vom Zustand I zu den Zuständen I_u, H_R oder H_D im Territorium i zum Zeitpunkt t.
	$\gamma_{I_\mathrm{u}}^{(i)}(t)$	Die Übergangsrate vom Zustand I_u zum Zustand R_u im Territorium i zum Zeitpunkt t.
	$\gamma_{H_R}^{(i)}(t)$	Die Übergangsrate vom Zustand H_R zum Zustand R_d im Territorium i zum Zeitpunkt t.
	$\gamma_{H_D}^{(i)}(t)$	Die Übergangsrate vom Zustand H_D zum Zustand D im Territorium i zum Zeitpunkt t.
Kontrollmaßnahmen	$m_E^{(i)}(t)$	Wirkungsgrad der Kontrollmaßnahmen für den Zustand E im Territorium i zum Zeitpunkt t.
	$m_I^{(i)}(t)$	Wirkungsgrad der Kontrollmaßnahmen für den Zustand I im Territorium i zum Zeitpunkt t.
	$m_{I_\mathrm{u}}^{(i)}(t)$	Wirkungsgrad der Kontrollmaßnahmen für den Zustand I_u im Territorium i zum Zeitpunkt t.
	$m_{H_R}^{(i)}(t)$	Wirkungsgrad der Kontrollmaßnahmen für den Zustand H_R im Territorium i zum Zeitpunkt t.
	$m_{H_D}^{(i)}$	Wirkungsgrad der Kontrollmaßnahmen für den Zustand H_D im Territorium i zum Zeitpunkt t.
Migration	$\tau_1^{(i)}$	Die Rate der Personen, die aus anderen Territorien in das Territorium i einreisen und infiziert sind.
	$\tau_2^{(i)}$	Die Rate der Personen, die das Territorium i verlassen und infiziert sind.

Tabelle 3.2: Zusammenfassung der Variablen und Parameter im θ-SEIRHD-Modell - Teil 2

Kategorie	Parameter	Beschreibung
Initialbedingungen	$S^{(i)}(t_0)$	Anzahl der anfälligen Personen zu Beginn der Modellierung.
	$E^{(i)}(t_0)$	Anzahl der exponierten Personen zu Beginn der Modellierung.
	$I^{(i)}(t_0)$	Anzahl der infizierten Personen zu Beginn der Modellierung.
	$I_u^{(i)}(t_0)$	Anzahl der undetektierten infizierten Personen zu Beginn der Modellierung.
	$H_R^{(i)}(t_0)$	Anzahl der hospitalisierten und sich erholenden Personen zu Beginn der Modellierung.
	$H_D^{(i)}(t_0)$	Anzahl der hospitalisierten und zum Tode bestimmten Personen zu Beginn der Modellierung.
	$R_d^{(i)}(t_0)$	Anzahl der genesenen Personen, die in einem Krankenhaus behandelt wurden, zu Beginn der Modellierung.
	$R_u^{(i)}(t_0)$	Anzahl der genesenen Personen, die undetektiert geblieben sind, zu Beginn der Modellierung.
	$D^{(i)}(t_0)$	Anzahl der verstorbenen Personen zu Beginn der Modellierung.

Tabelle 3.3: Zusammenfassung der Variablen und Parameter im θ-SEIRHD-Modell - Teil 3

Differentialgleichungen:

$$\frac{\mathrm{d}S^{(i)}}{\mathrm{d}t}(t) = -\frac{S^{(i)}(t)}{N^{(i)}}\left(m_E^{(i)}(t)\beta_E^{(i)}E^{(i)}(t) + m_I^{(i)}(t)\beta_I^{(i)}I^{(i)}(t) + m_{I_u}^{(i)}(t)\beta_{I_u}^{(i)}(\theta^{(i)}(t))I_u^{(i)}(t)\right)$$

$$-\frac{S^{(i)}(t)}{N^{(i)}}\left(m_{H_R}^{(i)}(t)\beta_{H_R}^{(i)}(t)H_R^{(i)}(t) + m_{H_D}^{(i)}(t)\beta_{H_D}^{(i)}(t)H_D^{(i)}(t)\right)$$

$$-\mu_m^{(i)}S^{(i)}(t) + \mu_n^{(i)}\left(S^{(i)}(t) + E^{(i)}(t) + I^{(i)}(t) + I_{\mathrm{u}}^{(i)}(t) + R_{\mathrm{d}}^{(i)}(t) + R_{\mathrm{u}}^{(i)}(t)\right)$$

Diese Gleichung beschreibt die Änderung der Anzahl der anfälligen Personen $S^{(i)}$ in einem bestimmten Gebiet i im Laufe der Zeit t. Die Anzahl der anfälligen Personen nimmt ab, wenn diese durch Kontakt mit infizierten Individuen aus den Kompartments (E, I, I_u, H_R, H_D) infiziert werden. (m_E, m_I, m_{I_u}, m_{H_R}, m_{H_D}) sind Funktionen für die Ansteckungsraten von (β_E, β_I, β_{I_u}, β_{H_R}, β_{H_D}), welche die Wirkung von Kontrollmaßnahmen

darstellen. $-\mu_m^{(i)} S^{(i)}(t)$ beschreibt die natürliche Sterberate und $\mu_n^{(i)}$ die Geburtenrate.

$$\frac{\mathrm{d}E^{(i)}}{\mathrm{d}t}(t) = \frac{S^{(i)}(t)}{N^{(i)}} \left(m_E^{(i)}(t)\beta_E^{(i)} E^{(i)}(t) + m_I^{(i)}(t)\beta_I^{(i)} I^{(i)}(t) + m_{I_u}^{(i)}(t)\beta_{I_u}^{(i)}(\theta^{(i)}(t)) I_u^{(i)}(t) \right)$$
$$+ \frac{S^{(i)}(t)}{N^{(i)}} \left(m_{H_R}^{(i)}(t)\beta_{H_R}^{(i)}(t) H_R^{(i)}(t) + m_{H_D}^{(i)}(t)\beta_{H_D}^{(i)}(t) H_D^{(i)}(t) \right)$$
$$-\mu_m^{(i)} E^{(i)}(t) - \gamma_E E^{(i)}(t) + \tau_1^{(i)}(t) - \tau_2^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der exponierten Personen $E^{(i)}$, die durch Kontakt mit Infizierten infiziert, aber noch nicht infektiös sind. Die Zunahme erfolgt durch die Infektion der anfälligen Personen $S^{(i)}$ von Infizierten (E, I, I_u, H_R, H_D). Die Abhnahme der Exponierten erfolgt durch die Sterblichkeit μ_m, dem Wechsel zur Infektionsphase γ_E, sowie durch Ein- und Ausreisen aus dem Gebiet τ_1, τ_2.

$$\frac{\mathrm{d}I^{(i)}}{\mathrm{d}t}(t) = \gamma_E E^{(i)}(t) - (\mu_m^{(i)} + \gamma_I^{(i)}(t)) I^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der infizierten und nachweisbaren Personen $I^{(i)}$. Die Anzahl nimmt zu, wenn Exponierte $E^{(i)}$ infektiös werden γ_E. Die Abnahme erfolgt durch natürliche Sterblichkeit μ_m und durch Übergang in den Zustand der Hospitalisierten oder Verstorbenen γ_I.

$$\frac{\mathrm{d}I_{\mathrm{u}}^{(i)}}{\mathrm{d}t}(t) = (1 - \theta^{(i)}(t))\gamma_I^{(i)}(t) I^{(i)}(t) - (\mu_m^{(i)} + \gamma_{I_u}^{(i)}(t)) I_{\mathrm{u}}^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der infizierten, aber nicht als infiziert erkannten Personen $I_u^{(i)}$. Diese Gruppe erhält einen Anteil von $1 - \theta$ der Infizierten I und verringert sich durch natürliche Sterblichkeit und Genesung γ_{I_u}.

$$\frac{\mathrm{d}H_R^{(i)}}{\mathrm{d}t}(t) = \theta^{(i)}(t) \left(1 - \frac{\omega^{(i)}(t)}{\theta^{(i)}(t)}\right) \gamma_I^{(i)}(t) I^{(i)}(t) - \gamma_{H_R}^{(i)}(t) H_R^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der hospitalisierten Personen, die sich erholen werden $H_R^{(i)}$. Der Zuwachs erfolgt durch die Hospitalisierung θ von Infizierten I und die Abnahme durch Genesung γ_{H_R}.

$$\frac{\mathrm{d}H_D^{(i)}}{\mathrm{d}t}(t) = \omega^{(i)}(t)\gamma_I^{(i)}(t)I^{(i)}(t) - \gamma_{H_D}^{(i)}(t)H_D^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der hospitalisierten Personen, die sterben werden $H_D^{(i)}$. Der Zuwachs erfolgt durch die Hospitalisierung der detektierten Infizierten I die nicht sterben werden und die Abnahme durch den Tod γ_{H_D}.

$$\frac{\mathrm{d}R_{\mathrm{d}}^{(i)}}{\mathrm{d}t}(t) = \gamma_{H_R}^{(i)}(t)H_R^{(i)}(t) - \mu_m^{(i)}R_{\mathrm{d}}^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der genesenen und detektierten Personen $R_d^{(i)}$. Der Zuwachs erfolgt durch Genesung γ_{H_R} von hospitalisierten H_R und die Abhnahme durch den natürlichen Tod mu_m.

$$\frac{\mathrm{d}R_{\mathrm{u}}^{(i)}}{\mathrm{d}t}(t) = \gamma_{I_u}^{(i)}(t)I_u^{(i)}(t) - \mu_m^{(i)}R_u^{(i)}(t)$$

Diese Gleichung beschreibt die Änderung der Anzahl der genesenen, aber nicht detektierten Personen $R_u^{(i)}$. Der Zuwachs erfolgt durch Genesung γ_{I_u} von infizierten, undetektierten Personen I_u und die Abhnahme durch den natürlichen Tod mu_m.

$$\frac{\mathrm{d}D^{(i)}}{\mathrm{d}t}(t) = \gamma_{HD}^{(i)}(t)H_D^{(i)}(t),$$

Diese Gleichung beschreibt die Änderung der Anzahl der Verstorbenen $D^{(i)}$, die wegen dem Tod aus dem Krankenhaus entlassen wurden γ_{H_D}.

3.2.4 Testergebnisse

Die Simulation des θ-SEIRHD-Modells hat zu neuen Erkenntnissen über die Dynamik der Pandemie und die Wirksamkeit von Kontrollmaßnahmen geführt. Die Simulation berücksichtigt unbekannte Fälle von COVID-19 Infektionen. Dieser neue Ansatz untersucht den Anteil erkannter Fälle im Verhältnis zur tatsächlichen Gesamtzahl der Infektionen und zeigt die Bedeutung dieses Verhältnisses auf die Auswirkungen von COVID-19.

Das Modell schätzt, dass etwa 168.500 Menschen in China infiziert gewesen sein könnten, wenn man unentdeckte Fälle einbezieht. Diese unentdeckten Fälle, die etwa 51% der Gesamtfälle ausmachen könnten, könnten etwa 37% der gesamten Infektionen verursacht

haben. Ivorra u. a. (2020) sind zur Erkentniss gekommen, dass eine Erhöhung des Anteils erkannter Fälle die Ausbreitung der Epidemie drastisch reduzieren könnte. Außerdem wurde die Grundreproduktionszahl für COVID-19 in China auf 4,225 geschätzt. Zudem sank die effektive Reproduktionszahl dank der Kontrollmaßnahmen und erreichte nach dem 1. Februar 2020 Werte unter 1.

Die strikten Kontrollmaßnahmen in China erwiesen sich also als effizient und zeigten eine Wirkung. Der sichtbare Effekt dieser Maßnahmen wurde jedoch erst mit einer Verzögerung von zwei Wochen nach ihrem Beginn deutlich. Bei der Validierung des Modells passten die Ergebnisse gut zu den von der WHO gemeldeten Daten hinsichtlich des Zeitpunkts und der Anzahl der Höhepunkte neuer Fälle, neuer Todesfälle und hospitalisierter Personen. Zur besseren Anpassung der Modellierung wurde eine gefilterte Version der WHO-Daten verwendet, um den plötzlichen Anstieg von 17.414 Fällen am 17. Februar 2020 gleichmäßig über die vorherigen Daten zu verteilen.

Das Modell konzentriert sich auf die Ausbreitung von COVID-19 innerhalb eines Landes oder Territoriums mit einer signifikanten Anzahl von Infektionen, wo die lokale Übertragung von Bedeutung ist. Die Ausbreitung zwischen Ländern wurde in dieser Studie nicht berücksichtigt.

Diese Vereinfachung wurde damit gerechtfertigt, dass die Studie primär darauf abzielte, die Wirksamkeit von Kontrollmaßnahmen innerhalb Chinas zu bewerten, wo die lokale Übertragung die Hauptursache der Epidemie war. Ein weiterer Grund für die Vereinfachung des Modells war die begrenzte Verfügbarkeit von Daten zur länderübergreifenden Verbreitung des Virus, insbesondere in Bezug auf Reisemuster. Zudem wurde die Studie unter hohem Zeitdruck durchgeführt, um schnell Erkenntnisse zur Eindämmung der Pandemie zu liefern. Eine detailliertere Modellierung der globalen Ausbreitung wäre in dem kurzen Zeitraum wegen der fehlenden Daten möglicherweise ungenauer gewesen.

Dadurch beeinflussen die getroffenen Vereinfachungen allerdings die Gültigkeit und Interpretierbarkeit der Ergebnisse. Die Modellvorhersagen sind primär für Länder oder Regionen mit hoher Infektionszahl und dominanter lokaler Übertragung bestätigt. Eine Übertragung auf andere Kontexte, insbesondere solche mit geringer Fallzahl oder starker internationaler Vernetzung, ist daher nicht aussagekräftig.

Ivorra u. a. (2020) erkennen an, dass ein Teil des entwickelten Modells verbessert werden kann. Insgesamt kann man aber sagen dass trotz der Vereinfachungen das SEIRHD-Modell wertvolle Erkenntnisse über die Ausbreitung von COVID-19 in China und die

Wirksamkeit von Kontrollmaßnahmen lieferte. Es demonstriert den Wert mathematischer Modelle in der Epidemiologie.

4 Methodik

4.1 Motivation

Künstliche Intelligenz (KI) hat sich in den letzten Jahren zu einer Schlüsseltechnologie entwickelt und durchdringt immer mehr Bereiche unseres Lebens. Dabei ist Deep Learning, eine spezielle Methode des maschinellen Lernens, besonders relevant geworden. Die stetig wachsende Anzahl erfolgreicher Anwendungen steigert das Interesse an Deep Learning weiter und motiviert Forscher und Entwickler, neue Einsatzmöglichkeiten zu entdecken. Die Erforschung der Potenziale von Deep Learning ist somit zu einem zentralen Thema in Wissenschaft und Industrie geworden.

Das Paper von Chopra u. a. (2023) ist ein gutes Beispiel dieser Einsatzmöglichkeiten. Es nutzt einen hybriden Ansatz zur Modellierung in der Epidemiologie. Die Autoren kombinieren verschiedene Modellierungskonzepte und nutzen Deep Learning, um ein Modell zu entwickeln, das sich automatisiert optimiert. Nach eingehender Auseinandersetzung mit den Techniken, die im GRADABM-Modell verwendet werden, sowie den gängigen epidemiologischen Modellierungsansätzen wie Kompartment-, agentenbasierten und Netzwerkmodellen, bin ich zu dem Entschluss gekommen, dass selbst dieser innovative Ansatz noch Verbesserungspotenzial aufweist.

GRADABM adressiert das Problem der Homogenität in Kompartmentmodellen durch die Verwendung eines Kontaktgraphen, der individuelle Interaktionen abbildet. Um die Performanz zu gewährleisten, werden Personen jedoch nicht als autonome Agenten implementiert, sondern ihre Zustände in einem Tensor gespeichert. Diese Designentscheidung führt zu vier wesentlichen Problemen, die im Paper nicht berücksichtigt werden:

- **Anwenden von Interventionen:** Soziale Distanzierungsmaßnahmen wie die Ein-Personen-Kontaktbeschränkung lassen sich nicht korrekt durch einfaches Entfernen von Kanten im Kontaktgraphen darstellen. Die Maßnahme beschränkt nicht die Anzahl der Kontakte, sondern die Gleichzeitigkeit.

- **Wechselwirkung zwischen Kontakt und Infektion:** Der Kontaktgraph wird vor dem Training festgelegt und bleibt statisch. In der Realität beeinflusst jedoch die Infektion das Kontaktverhalten. Erkrankte Personen reduzieren oft ihre sozialen Kontakte, was im Modell nicht berücksichtigt wird.
- **Deterministischer Ansatz:** Das Modell ist linear deterministisch und vernachlässigt somit den Einfluss von Zufallsereignissen und nichtlinearen Beziehungen auf den Krankheitsverlauf. In der Realität können jedoch unvorhergesehene Ereignisse oder komplexe Wechselwirkungen den Verlauf einer Epidemie beeinflussen.
- **Darstellung der Agenten als Tensoren:** Die fehlende Implementierung von Personen als eigenständige Agenten verhindert die Beobachtung von emergentem Verhalten. Dadurch können wichtige Erkenntnisse über die Dynamik der Ausbreitung und unerwartete Phänomene auf der Makroebene verloren gehen.

Wegen dieser Probleme habe ich mich dazu entschlossen, an einem eigenen Modellentwurf zu arbeiten, der sie lösen kann.

4.2 Stärken bisheriger Modelle

Die verschiedenen Modellierungskonzepte in der Epidemiologie haben trotz ihrer jeweiligen Herausforderungen wichtige Erkenntnisse und Techniken hervorgebracht, die für die Entwicklung effektiver Modelle unerlässlich sind:

- **Differenzierbare Modelle:** Die Implementierung eines differenzierbaren Modells ermöglicht die nahtlose Integration mit neuronalen Netzen (NN) und somit eine automatische Optimierung der Modellparameter. Da neuronale Netze im Grunde eine Abfolge differenzierbarer Operationen darstellen, kann ein differenzierbares Modell als Erweiterung dieser Schichten betrachtet werden. Obwohl sich die spezifischen Operationen und ihre Reihenfolge unterscheiden können, ermöglicht die Differenzierbarkeit die Anwendung des Backpropagation-Algorithmus zur effizienten Parameteroptimierung. Dadurch werden die Vorhersagekraft und Anpassungsfähigkeit des Modells erheblich verbessert.
- **Agenten in der Mikroebene:** Die Darstellung von Individuen als Agenten in ABMs erhöht die Flexibilität und Aussagekraft des Modells. Durch die modulare Struktur von ABMs können gesellschaftliche Muster realistischer abgebildet und

Interventionen direkt implementiert werden, ohne aufwendige Transformationen vornehmen zu müssen. Zudem ermöglicht die individuelle Anpassung der Agenten, die Auswirkungen einer Infektion auf das Verhalten zu berücksichtigen und nichtlineare oder deterministische Prozesse einzubeziehen.

- **Kompartments als Zustände:** Die Verwendung von Kompartments als Zustände in epidemiologischen Modellen hat sich als sinnvoll erwiesen, da sie empirisch bestätigte Krankheitsverläufe abbilden. Diese Unterteilung in verschiedene Zustände ermöglicht eine differenzierte Betrachtung des Krankheitsverlaufs und erleichtert die Modellierung der Dynamik einer Epidemie.
- **Tensoren zur Berechnung:** Obwohl die Darstellung von Individuen als Tensor die Aussagekraft eines Modells einschränken kann, bietet diese Methode Vorteile bei gleichbleibenden Berechnungen mit großen Datenmengen. Tensoren ermöglichen Hardwarebeschleunigung und effiziente Gradientenberechnung durch automatische Differenzierung, was die Performanz des Modells erheblich steigert.

4.3 Ziel dieser Forschung

Ziel dieser Arbeit ist die Entwicklung eines hybriden Modells, das ähnlich wie GRADABM bekannte Konzepte und Stärken kombiniert, jedoch die zuvor genannten Probleme löst. Das Modell wird in ein Übertragungs- und ein Verlaufsmodell unterteilt. Das Übertragungsmodell ist agentenbasiert und optimiert seine Parameter selbstständig mithilfe eines neuronalen Netzes, um das Kalibrierungsproblem herkömmlicher ABMs zu lösen. Für die Zustände der Agenten werden Kompartments verwendet. Zur Steigerung der Performanz und aufgrund der begrenzten Anzahl von Möglichkeiten im Krankheitsverlauf werden Tensoren für das deterministische Verlaufsmodell eingesetzt. Ein deterministisches Verlaufsmodell ist in diesem Kontext sinnvoll, da der biologische Verlauf einer Krankheit oft einem klaren Muster folgt und durch spezifische Parameter wie Inkubationszeit, Dauer der Infektiosität und Wahrscheinlichkeiten für verschiedene Krankheitsausgänge gut beschrieben werden kann. Ein zentraler Aspekt des Modells ist die Entkopplung von Verlaufs- und Übertragungsmodell sowie die Entkopplung von Agenten und Umgebung innerhalb des Übertragungsmodells. Diese Entkopplung erhöht die Flexibilität und ermöglicht die Anwendung des Modells in unterschiedlichen Kontexten. Da das Verhalten der Menschen auf verschiedenen Teilen der Erde ähnlich ist, braucht man bei der Übertragung des Modells auf verschiedene Regionen lediglich die Umgebung anzupassen. Bei

der Anwendung auf unterschiedliche Krankheiten muss hingegen nur das Verlaufsmodell modifiziert werden.

4.4 Herausforderungen bei der Umsetzung des hybriden Modells

Der Entwurf eines Modells, das die in Abschnitt 4.3 beschriebenen Anforderungen erfüllt, stellt einige Herausforderungen dar:

- **Sparse-Tensor API:** Für die effiziente Implementierung der Tensoren, die zur Darstellung der Zustände im Verlaufsmodell verwendet werden, ist eine Sparse-Tensor API erforderlich. Diese API ermöglicht die effiziente Speicherung und Verarbeitung von Tensoren, was für die Skalierbarkeit des Modells entscheidend ist.
- **Differenzierbarkeit und Backpropagation:** Während GRADABM ein vollständig differenzierbares Modell verwendet, um die Backpropagation zu ermöglichen, sind ABMs und emergentes Verhalten inhärent nicht differenzierbar. Um die automatische Kalibrierung der Parameter durch das neuronale Netz zu ermöglichen, muss eine Möglichkeit gefunden werden, Differenzierbarkeit in das ABM zu integrieren.
- **Hardwarebeschränkungen:** Im Gegensatz zu Forschungseinrichtungen, die oft über leistungsstarke Rechenressourcen verfügen, ist die Hardware, die für diese Arbeit zur Verfügung steht, begrenzt. Daher muss eine Implementierung gefunden werden, die vereinfacht ist, aber dennoch die Gültigkeit des vorgeschlagenen Modells demonstriert. Zudem ist die Analyse umfangreicher epidemiologischer Datensätze aufgrund der begrenzten Rechenleistung erschwert.

4.5 Realisierung des Modells

4.5.1 Verwendete Softwarekomponenten

Für die Umsetzung meines epidemiologischen Modells habe ich mich für das MARS Framework (vgl. MARS-Group 2024a) entschieden. MARS hat sich bereits in der Entwicklung verschiedener komplexer Simulationen bewährt und bietet nützliche Werkzeuge zur

Vereinfachung der Implementierung von ABMs. Da MARS in C# geschrieben ist, war die direkte Verwendung von PyTorch für Tensoren und neuronale Netze nicht möglich. Stattdessen habe ich auf die Implementierungen des SciSharp (2024) Teams zurückgegriffen, die TensorFlow.NET für Tensoren und TensorFlow.Keras für neuronale Netze bereitstellen. TensorFlow.Keras ist eine High-Level-API für Deep Learning, die die Erstellung und das Training neuronaler Netze erleichtert. Sie abstrahiert die Komplexität der zugrunde liegenden TensorFlow-Bibliothek und ermöglicht eine schnelle Entwicklung. Für bestimmte mathematische Berechnungen habe ich zusätzlich das MathNet.Numerics-Paket verwendet, das eine umfangreiche Sammlung mathematischer Funktionen und Algorithmen für C# bietet.

4.5.2 Implementierung und Validierung

Anstatt die Modelle anhand epidemiologischer Daten zu trainieren, habe ich mich nur für einen von mir bestimmten Zielwert entschieden, nämlich die Anzahl der Todesfälle.

Um die optimale Implementierung des Modells zu ermitteln, habe ich zwei unterschiedliche Ansätze verfolgt:

- **model-epidemic-spread-self-contained:** Jeder Agent verfügt über einen eigenen Zustandstensor, und das Verlaufsmodell wird individuell für jeden Agenten ausgeführt. Eine zentrale Instanz ermittelt am Ende jedes Zeitschritts die Anzahl der verstorbenen Agenten. Zur Vereinfachung der Berechnung wird eine Sammlung verwendet, die jedem Agenten mitteilt, mit welchen anderen Agenten er in einem Zeitschritt interagiert. Diese Umgebung ähnelt einem Kontaktgraphen.
- **model-epidemic-spread-combined:** Jeder Agent hat zwar einen Zustand, dieser wird jedoch nicht als Tensor gespeichert. Stattdessen besitzt eine zentrale Instanz den Zustand jedes Agenten als einen großen Tensor und verwaltet die Zustände aller Agenten. In jedem Tick liest ein Agent seinen Zustand aus der zentralen Instanz aus. Das Verlaufsmodell wird von dieser zentralen Instanz implementiert. Auch hier wird eine Sammlung verwendet, um die Interaktionen der Agenten zu bestimmen.

Die Bestimmung der Kontakte der Agenten basiert auf einer Vorgabe von GRADABM[1].

Beide Modelle nutzen, ähnlich wie GRADABM, ein neuronales Netz zur Kalibrierung. Allerdings setze ich ein vereinfachtes neuronales Netz SimpleCalibNN ein, das aus einem

[1] https://github.com/AdityaLab/GradABM

Feedforward-Netzwerk (FFN) anstelle eines Transformers besteht. Dieses Netz erhält keine Eingabe und orientiert sich während des Trainings an einem vorgegebenen Zielwert.

Zur Validierung meiner Modelle verwende ich zwei Implementierungen:

- **model-epidemic-spread-gradabm:** Eine exakte Nachbildung der Funktionsweise von GRADABM.
- **model-epidemic-spread-without-tensors:** Ein herkömmliches ABM ohne neuronales Netz oder Tensoren, um die Skalierbarkeit meiner Modelle zu überprüfen.

Durch den Vergleich dieser Implementierungen kann ich die Leistung, Effizienz und Skalierbarkeit meines Modells bewerten und sicherstellen, dass es eine Verbesserung gegenüber bestehenden Ansätzen darstellt.

4.5.3 Beweis der Gültigkeit

In diesem Abschnitt erläutere ich meine Entscheidung für die spezifische Herangehensweise zur Umsetzung meines Modells und warum diese ausreichend ist, um die erfolgreiche Zielerreichung zu demonstrieren.

```
using (var tape = tf.GradientTape())
{
  var x = tf.Variable(3.0f);
  var random = new Random();
  var process1 = x * 2;
  var process2 = tf.constant(random.Next());
  var process3 = process1 + process2;
  var gradients = tape.gradient(process3, x);

  Console.WriteLine($"Gradient nach x: {gradients.numpy()}");
}
```

Listing 4.1: Einfache Berechnung eines Gradienten

Chopra u. a. (2023) betonen in ihrer Arbeit wiederholt die vollständige Differenzierbarkeit ihres Übertragungsmodells und erwecken den Eindruck, dass die Kalibrierung eines Modells durch ein neuronales Netz nur mit vollständig differenzierbaren Prozessen möglich ist. Dies ist jedoch nicht der Fall. Für die Optimierung der Parameter eines ABMs durch ein NN müssen lediglich die Prozesse differenzierbar sein, die den zu optimierenden Parameter enthalten. Teilprozesse, deren Ergebnisse in einem solchen Prozess verwendet

werden, müssen nicht differenzierbar sein. Entscheidend ist, dass der Gradientenfluss ab der Verwendung eines NN-Parameters erhalten bleibt. Im Code 4.1 ist process2 nicht differenzierbar, da eine Zufallszahl generiert wird. Dennoch kann der Gradient nach x aus der Summe von process1 und process2 berechnet werden. Die Ausgabe des Codeabschnitts ist: "Gradient nach x: 2".

Dies zeigt, dass nicht-differenzierbare Teilprozesse erlaubt sind, solange die relevanten Prozesse differenzierbar bleiben. Somit ist es möglich, ein ABM mit objektbasierten Agenten zu entwickeln, das durch ein NN kalibriert wird, indem sichergestellt wird, dass die relevanten Prozesse differenzierbar sind.

Ich habe zuvor die unzureichende Darstellung von Interventionen im Kontaktnetzwerk von GRADABM kritisiert, verwende jedoch in meinen eigenen Modellen eine ähnlich vereinfachte Umgebung. Dies liegt an den begrenzten Rechenressourcen meines Computers, der bei einer großen Anzahl von Agenten keine komplexen Bewegungsdynamiken in angemessener Zeit simulieren kann. Allerdings ist diese Vereinfachung nicht problematisch, da ich das MARS Framework für die Modellierung verwende. MARS ermöglicht den Austausch der Agentenumgebung, sodass ich bei Bedarf auf komplexere Modelle zurückgreifen kann. Ein Beispiel hierfür ist das SmartOpenHamburg (SOH) Modell, das bereits erfolgreich in MARS (vgl. MARS-Group 2024b) implementiert wurde. SOH ist ein hochdetailliertes Mobilitätssimulationsmodell für großflächige Szenarien, das das individuelle Mobilitätsverhalten von Agenten und deren Interaktionen simuliert. Sollte mein Experiment erfolgreich sein, könnte ich das Übertragungsmodell durch das SOH-Modell ersetzen und somit detailliertere Simulationen durchführen, die komplexere Bewegungsdynamiken und Interventionen berücksichtigen.

Da mein Modell nicht auf epidemiologischen Daten trainiert wird, ist eine Validierung anhand solcher Daten nicht sinnvoll. Stattdessen nutze ich das model-epidemic-spread-gradabm Modell als Referenz. Dieses Modell ist eine direkte Umsetzung von GRADABM in das MARS Framework. GRADABM wurde bereits anhand umfangreicher Experimente mit realen COVID-19- und Influenza-Datensätzen validiert. Dabei wurde GRADABM erfolgreich auf verschiedene Landkreise in Massachusetts angewendet. Wenn mein Modell unter Verwendung der gleichen Parameter wie model-epidemic-spread-gradabm einen identischen Krankheitsverlauf simuliert, kann ich davon ausgehen, dass es ebenfalls in der Lage ist, Epidemien und Pandemien realistisch abzubilden.

Um zu demonstrieren, dass ein Transformer wie CalibNN in der Lage ist, Parameter für mein Modell zu generieren, reicht es aus, ein Beispiel mit einem einfacheren Feedforward

Neural Network zu verwenden. Das liegt daran, dass ein Transformer im Wesentlichen eine komplexere Variante eines neuronalen Netzwerks ist. Beide Architekturen, FFN und Transformer, basieren auf dem gleichen Grundprinzip: Sie lernen aus Daten und passen ihre Parameter an, um eine bestimmte Aufgabe zu erfüllen. In diesem Fall ist die Aufgabe die Generierung von Parametern für ein epidemiologisches Modell. Ein Transformer verfügt über zusätzliche Mechanismen wie Selbstaufmerksamkeit und mehrere Schichten, die ihn leistungsfähiger machen als ein einfaches FFN. Diese zusätzlichen Mechanismen ändern jedoch nicht das grundlegende Prinzip der Parametergenerierung durch ein neuronales Netz.

5 Implementierung

5.1 Beschreibung der Simulations- und Trainingsschleifen der Modelle

Zum Finden eines Modells das meinen Anforderungen entspricht habe ich verschiedene Modelle implementiert und getestet. Das UML-Diagramm 5.1 beschreibt den Ablauf der Modelle, die sich an dem Ablauf von GRADABM orientieren. Die Modelle arbeiten grundsätzlich nach dem gleichen Prinzip, abgesehen von kleinen Unterschieden in der Implementierung.

Simulationsschleife: Die Simulation ist tick-basiert, das heißt, sie wird in festgelegten Zeitintervallen (Ticks) durchgeführt. Jeder Tick ist in drei Phasen unterteilt:

1. **Interaktionen:** In der ersten Phase interagieren die Agenten miteinander. Dabei kann es zu einer Übertragung der Infektion kommen, wenn bestimmte Bedingungen erfüllt sind.

2. **Kompartmentswechsel:** In der zweiten Phase wechseln die Agenten ihre Kompartments. Zum Beispiel kann ein Agent von infiziert zu erholt wechseln.

3. **Todesfälle aggregieren:** In der dritten Phase wird die Anzahl der Agenten, die im aktuellen Tick gestorben sind, erfasst und aggregiert.

Trainingsschleife: Das Kalibrierungsnetzwerk, ein neuronales Netz, generiert die epidemiologischen Parameter für die Simulation. Die Simulation läuft über n Ticks. Am Ende der Simulation wird die Gesamtzahl der Todesfälle $\hat{y}^w$ ermittelt und mit einem vorgegebenen Zielwert y verglichen.

Der Verlust wird mithilfe der Kostenfunktion Mean-Squared Error (MSE) berechnet: $\mathcal{L}\left(\hat{y}^w, y; \left(\theta_T^t, \theta_P^t\right)^w\right) = \mathrm{MSE}\left(\hat{y}^w, y\right)$

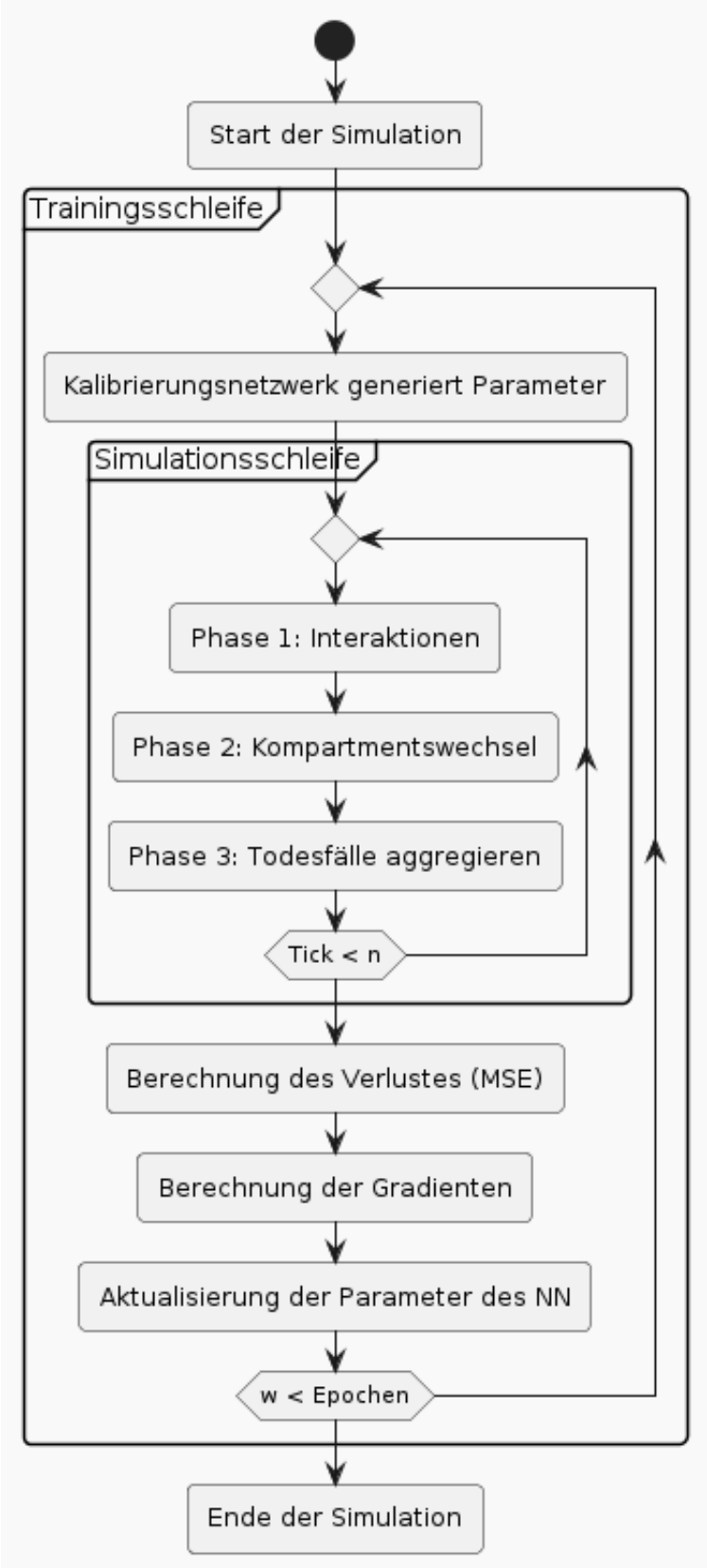

Abbildung 5.1: Allgemeiner Ablauf der Simulation

Anschließend wird die Gradientenbasierte Optimierung über die Parameter des neuronalen Netzwerks ϕ ausgeführt, um die Parameter des Netzwerks zu aktualisieren:

$$\phi^{w+1} = \phi^w - \alpha \frac{\partial \mathcal{L}\left(\hat{y}^w, y; \left(\theta_T^t, \theta_P^t\right)^w\right)}{\partial \phi}$$

Dabei ist α die Lernrate.

Komponenten:

- **Kalibrierungsnetzwerk:** Das Neuronale Netz SimpleCalibNN generiert die epidemiologischen Parameter (θ_T^t und θ_P^t). θ_T^t ist die effektive Reproduktionszahl und wird im Übertragungsmodell verwendet. θ_P^t sind die Parameter für das Verlaufsmodell, wie die initiale Infektionsrate und die Sterberate.
- **Agenten:** Die Agenten haben verschiedene Eigenschaften, die die Ausbreitung der Infektion beeinflussen:
  - Altersgruppe $\alpha \in \{0-10, 11-20, 21-30, ..., 71-80, 80+\}$
  - Das Kompartment in das sich der Agent befindet $d^t \in \{S, E, I, R, M\}$
  - Der Zeitpunkt der letzten Infektion $e^t \in \{-1, ..., t-1\}$
  - $\hat{I}$ die erwartete Anzahl von Interaktionen in einem Tick
  - $S_j \in \{0.35, 0.69, 1.03, 1.03, 1.03, 1.03, 1.27, 1.52\}$ die Empfänglichkeit für eine Infektion
- **Layer:** Eine zentrale Instanz, die nach dem Agieren der Agenten die Anzahl der Todesfälle aggregiert und das Verlaufsmodell simuliert. Sie verwaltet auch die Agenten und die Umgebung des Modells.
- **Umgebung:** Ermöglicht es den Agenten, miteinander zu interagieren. In model-epidemic-spread-gradabm übernimmt die Umgebung auch die Infektionsausbreitung.

Interaktion: Bei jeder Interaktion zwischen zwei Agenten i und j , bei der ein Agent i im Kompartment $d_i^t \in \{S\}$ und ein Agent j im Kompartment $d_j^t \in \{E, I\}$ ist, kommt es mit einer Wahrscheinlichkeit q zur Übertragung einer Infektion. Die Wahrscheinlichkeit q wird wie folgt berechnet:

$q(d_i^t, d_j^t) = 1 - e^{-\lambda(R,S_i,T_j,\Delta E_j^t)}$.

$\lambda(R, S_i, T_j, \Delta E_j^t) = \frac{RS_iT_j}{\hat{I}_i} \int_{\Delta E_j^t - 1}^{\Delta E_j^t} G_\Gamma(u; \mu_j, \sigma_j^2) du$, wobei $(\Delta E_j^t = t - e_j^t)$ die Zeit seit der letzten Infektion und T_j die Übertragbarkeit des Infizierers ist.

Das Modell model-epidemic-spread-without-tensors verwendet kein Kalibrierungsnetzwerk und hat keine Trainingsschleife.

Eine detaillierte Erklärung der Implementierungen der verschiedenen Modelle befindet sich im Anhang.

6 Ergebnisse und Diskussion

6.1 Spezifikationen der genutzten Hardware

Prozessor:

- Name: Intel(R) Core(TM) i7-6700K
- Anzahl physischer Kerne: 4
- Taktfrequenz des Prozessors: 4.00 GHz

Installierter RAM: 16 GB

Betriebssystem: Windows 10 Pro

Grafikkarte:

- Name: NVIDIA GeForce GTX 1070
- CUDA-Kerne: 1920
- Grafiktakt: 1506 MHz
- Prozessortakt: 1683 MHz
- Speichertakt: 8 Gbps
- Standardspeicherkonfig.: 8 GB GDDR5

6.2 Validierung

In diesem Abschnitt wird anhand von Graphen, die die Simulationsverläufe bei verschiedenen Parametern darstellen, demonstriert, dass alle Modelle funktionsfähig sind. Die Darstellung der Modellgraphen erfolgt in alphabetischer Reihenfolge der Modellnamen: model-epidemic-spread-combined, model-epidemic-spread-gradabm, model-epidemic-spread-self-contained, model-epidemic-spread-without-tensors.

Die für die Simulation verwendeten Kontaktinformationen der Agenten sind in der Datei contact_edges.csv aus Baran (2024) zu finden.

Parameter: 3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 5.18, Ticks 50

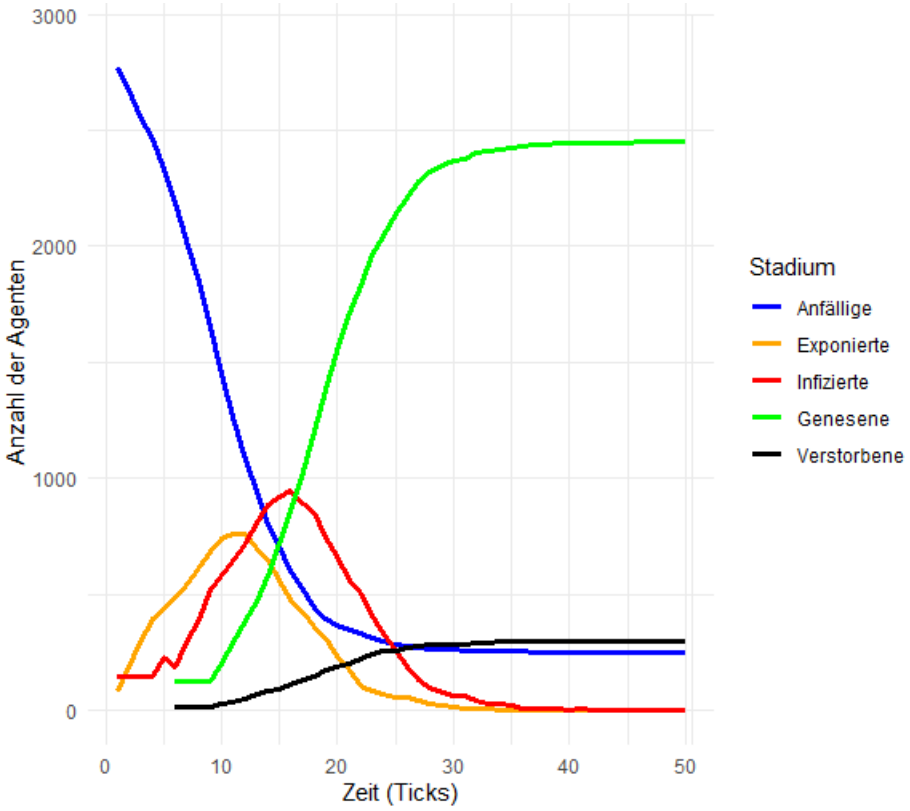

Abbildung 6.1: Krankeitsverlauf im Modell model-epidemic-spread-combined mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 5.18, Ticks 50)

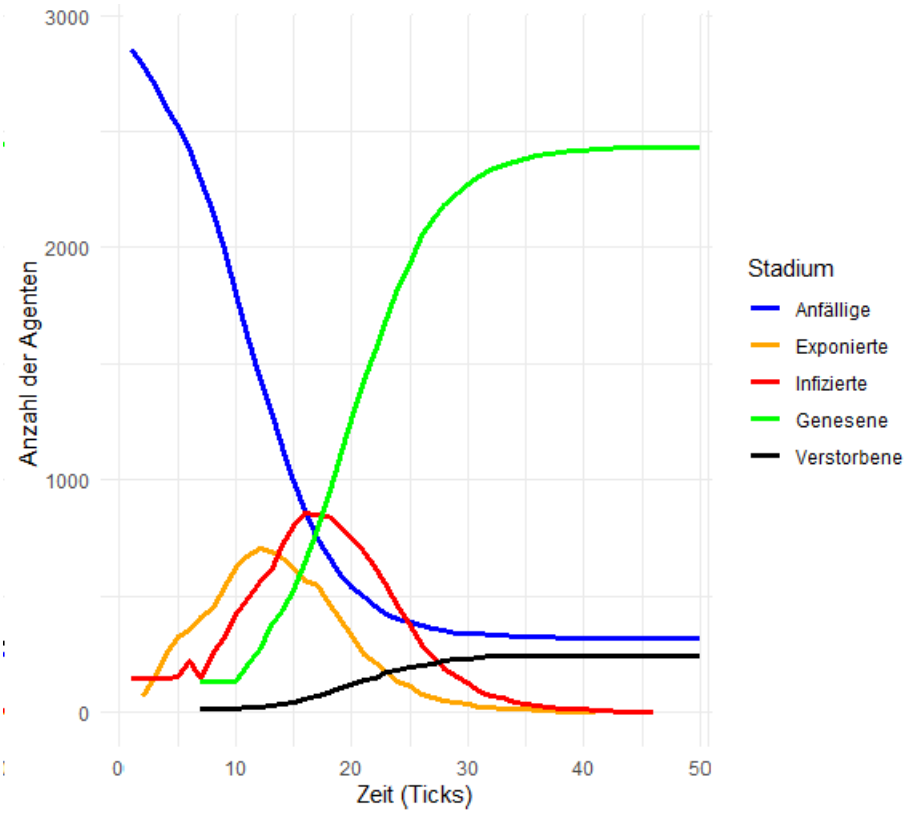

Abbildung 6.2: Krankeitsverlauf im Modell model-epidemic-spread-gradabm mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 5.18, Ticks 50)

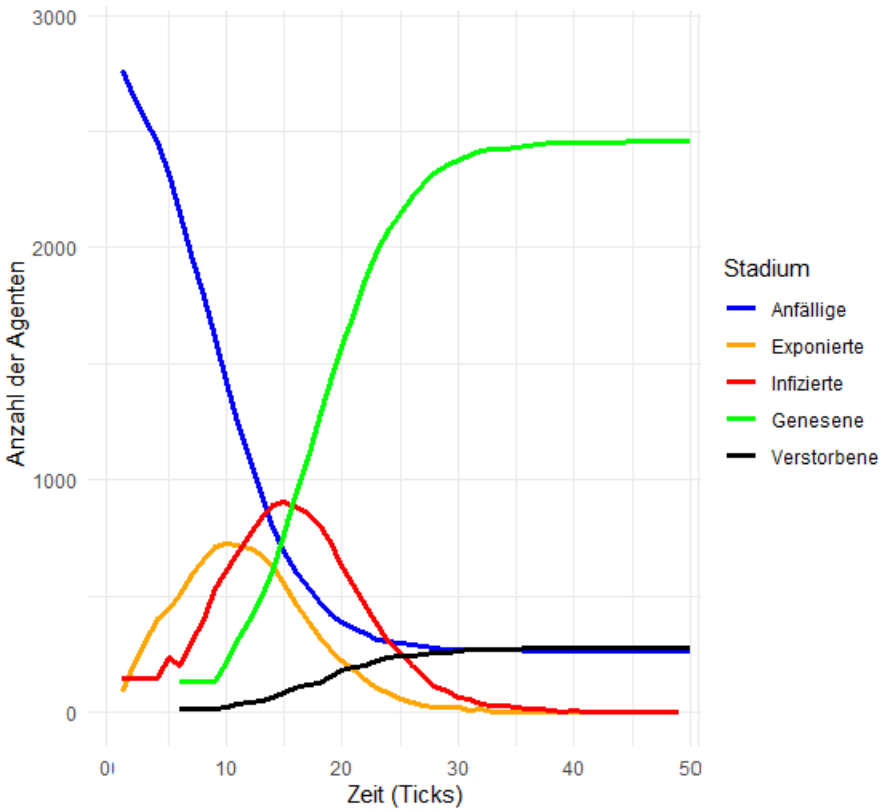

Abbildung 6.3: Krankeitsverlauf im Modell model-epidemic-spread-self-contained mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 5.18, Ticks 50)

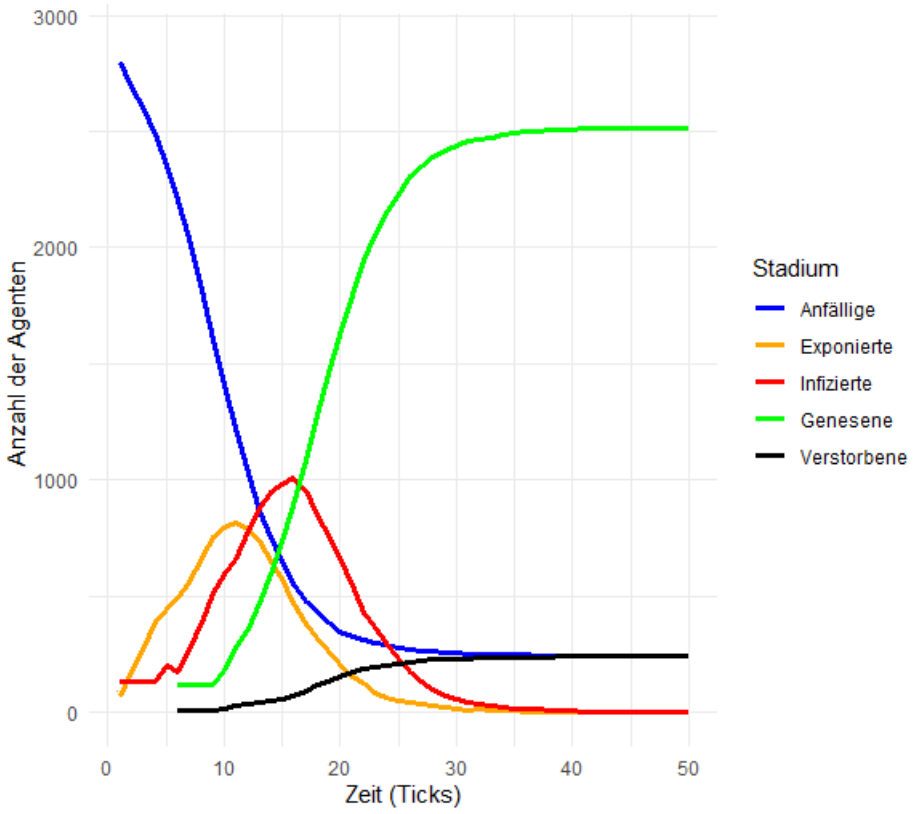

Abbildung 6.4: Krankeitsverlauf im Modell model-epidemic-spread-without-tensors mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 5.18, Ticks 50)

Parameter: 3000 Agenten, Initiale Infektionsrate 0.2, Sterberate 0.4, effektive Reproduktionszahl 7, Ticks 50

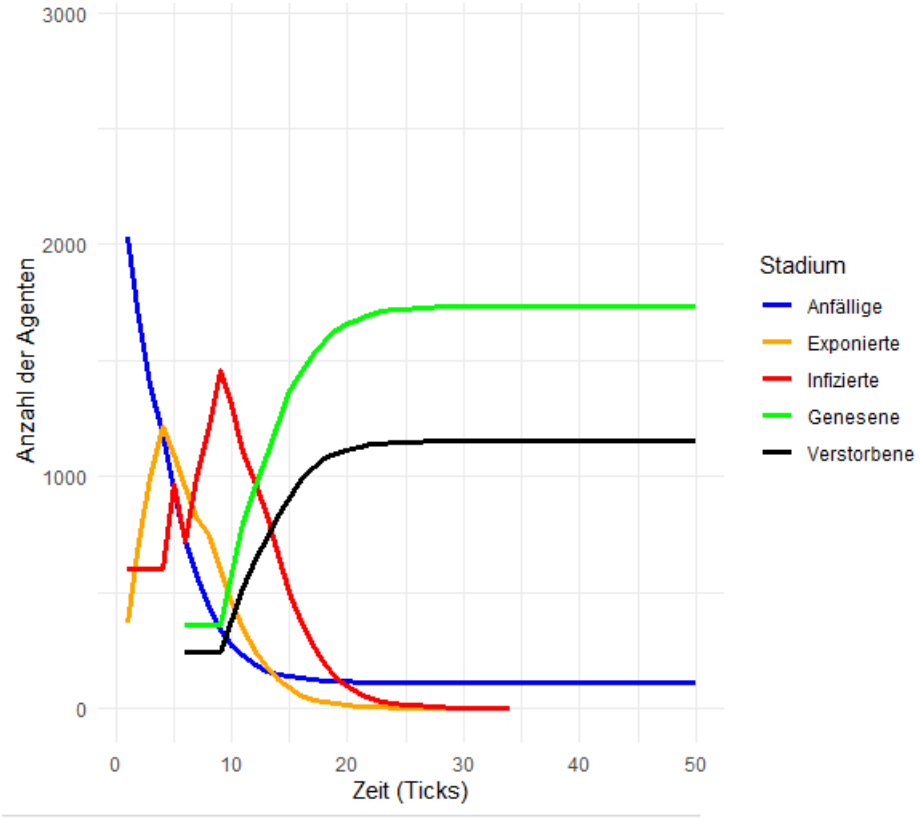

Abbildung 6.5: Krankeitsverlauf im Modell model-epidemic-spread-combined mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.2, Sterberate 0.4, effektive Reproduktionszahl 7, Ticks 50)

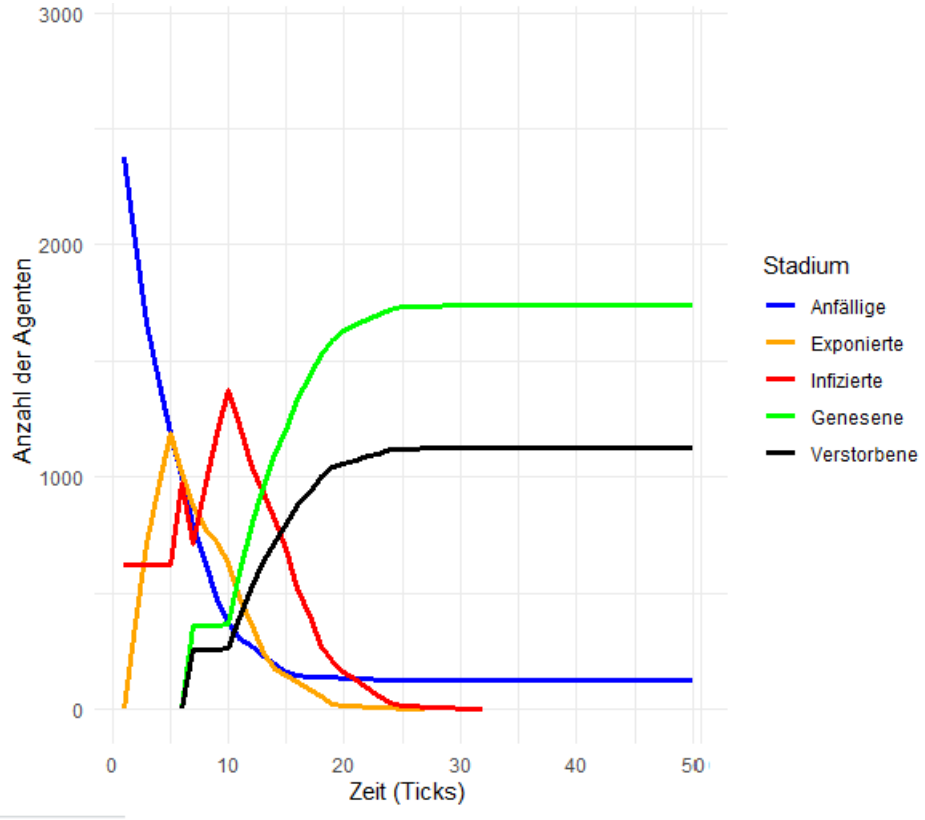

Abbildung 6.6: Krankeitsverlauf im Modell model-epidemic-spread-gradabm mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.2, Sterberate 0.4, effektive Reproduktionszahl 7, Ticks 50)

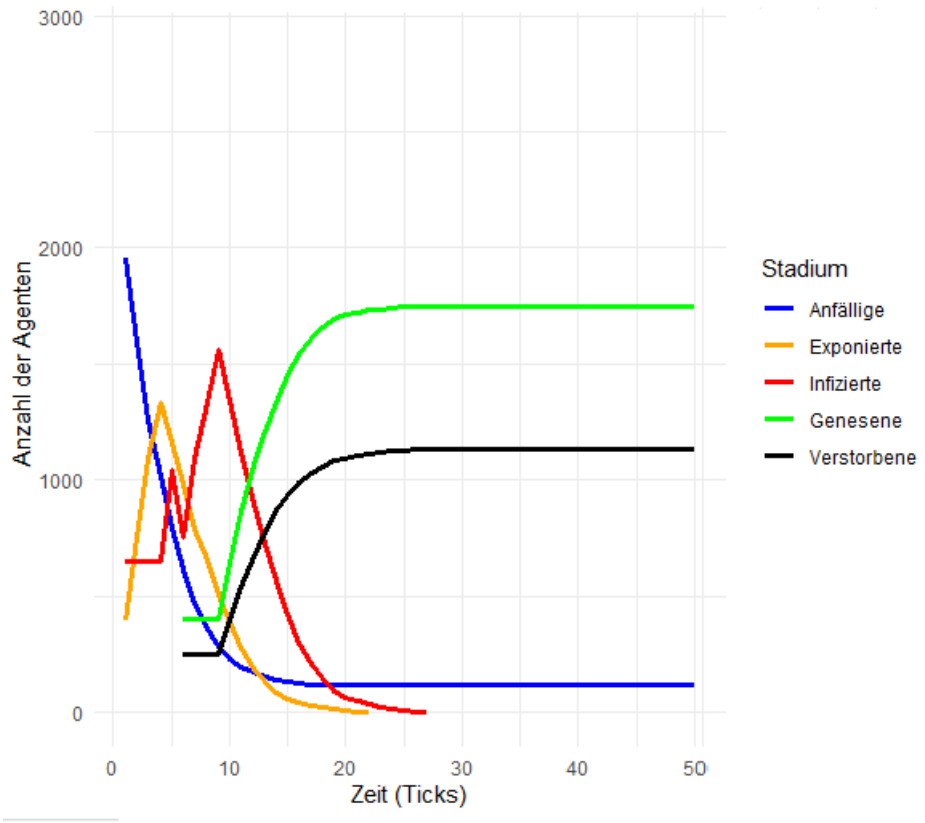

Abbildung 6.7: Krankeitsverlauf im Modell model-epidemic-spread-self-contained mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.2, Sterberate 0.4, effektive Reproduktionszahl 7, Ticks 50)

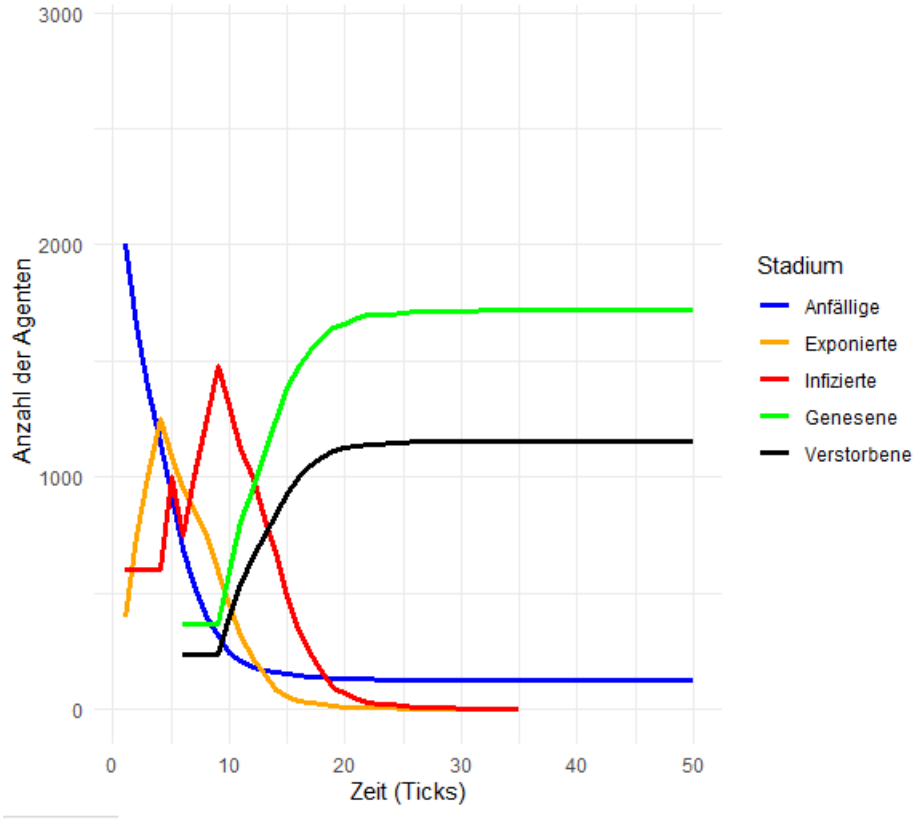

Abbildung 6.8: Krankeitsverlauf im Modell model-epidemic-spread-without-tensors mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.2, Sterberate 0.4, effektive Reproduktionszahl 7, Ticks 50)

Parameter: 3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 8, Ticks 50

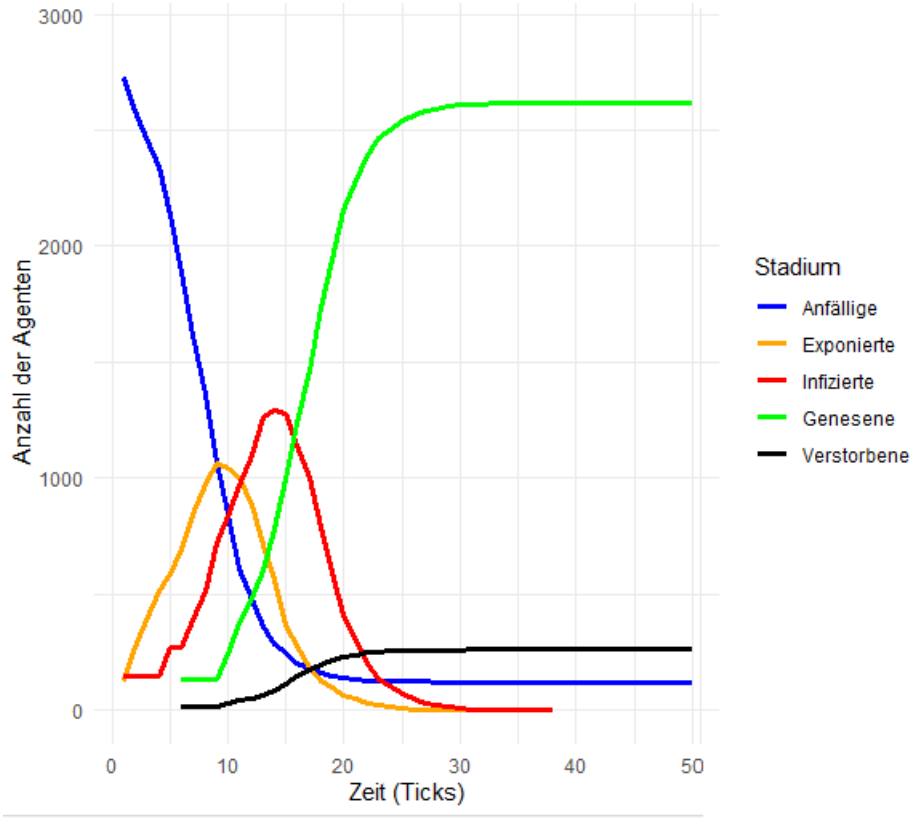

Abbildung 6.9: Krankeitsverlauf im Modell model-epidemic-spread-combined mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 8, Ticks 50)

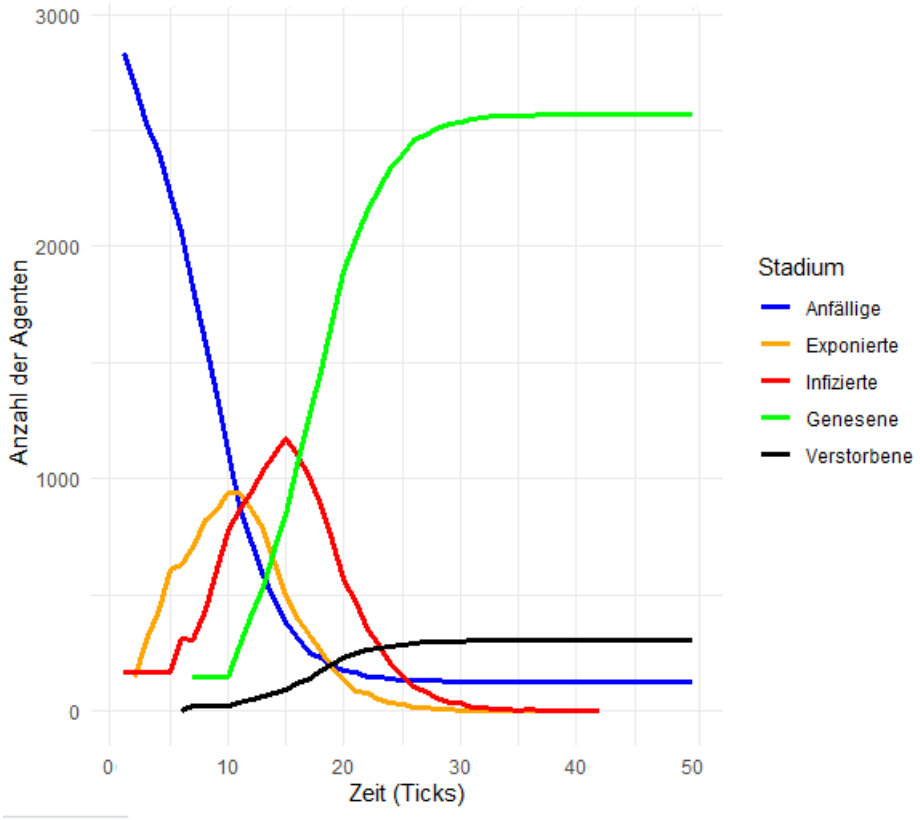

Abbildung 6.10: Krankeitsverlauf im Modell model-epidemic-spread-gradabm mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 8, Ticks 50)

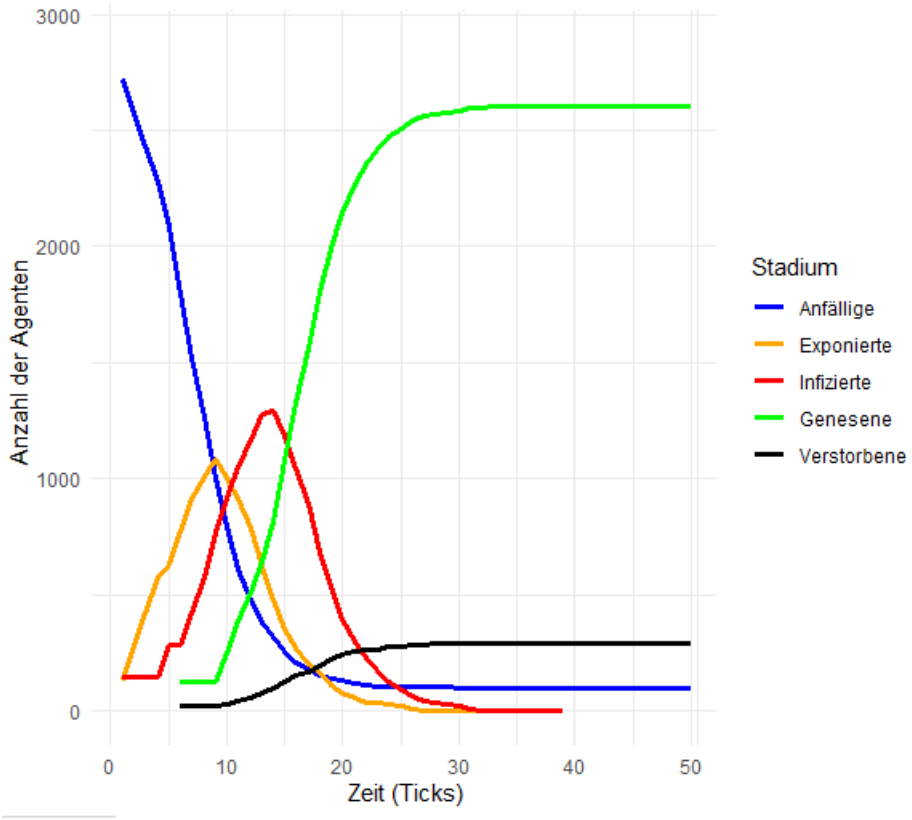

Abbildung 6.11: Krankeitsverlauf im Modell model-epidemic-spread-self-contained mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 8, Ticks 50)

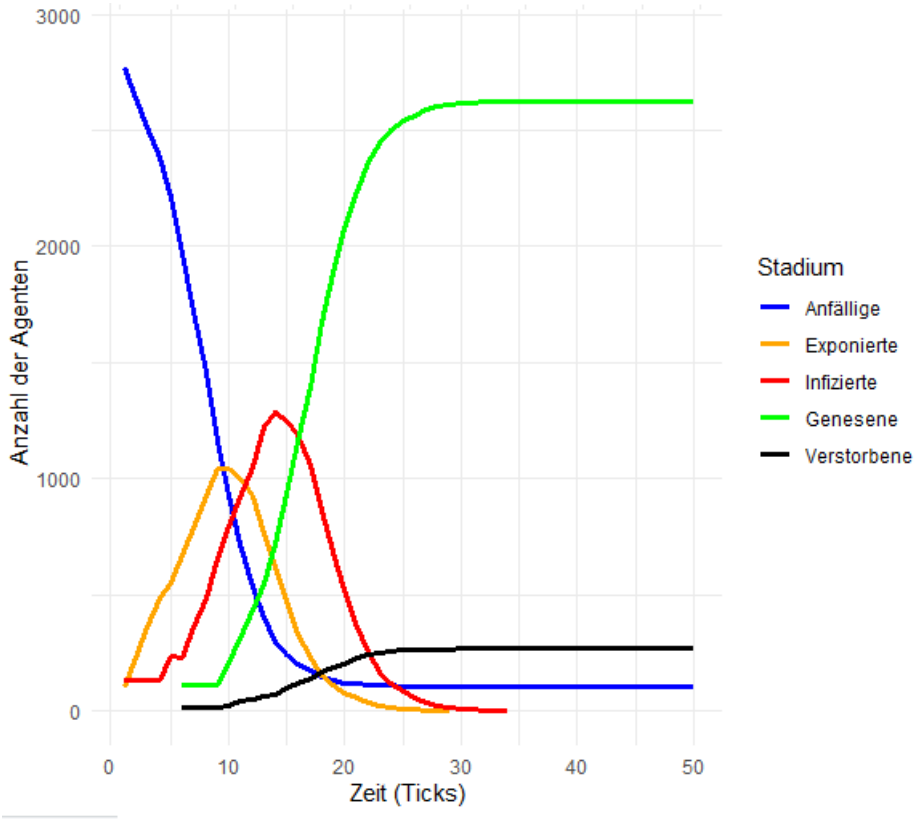

Abbildung 6.12: Krankeitsverlauf im Modell model-epidemic-spread-without-tensors mit den Parametern: (3000 Agenten, Initiale Infektionsrate 0.05, Sterberate 0.1, effektive Reproduktionszahl 8, Ticks 50)

6.3 Skalierung

In diesem Abschnitt wird getestet wie gut die verschiedenen Modelle skalieren.

Modellname	combined	gradabm	self-contained	without-tensors
Zeit in Sekunden	0	8	6	5

Tabelle 6.1: 3000 Agenten, Ticks 50

Modellname	combined	gradabm	self-contained	without-tensors
Zeit in Sekunden	4	86	65	54

Tabelle 6.2: 3000 Agenten, Ticks 500

Modellname	combined	gradabm	self-contained	without-tensors
Zeit in Sekunden	1	25	9	6

Tabelle 6.3: 8000 Agenten, Ticks 50

Modellname	combined	without-tensors
Zeit in Sekunden	9	17

Tabelle 6.4: 128562 Agenten, Ticks 50

Modellname	combined
Zeit in Sekunden	10

Tabelle 6.5: 128562 Agenten, Ticks 50, GPU

6.4 Training

Simulationsparameter: 50 Ticks, 15000 Agenten

Trainingsparameter: 100 Epochen, 3000 Tode als Zielwert

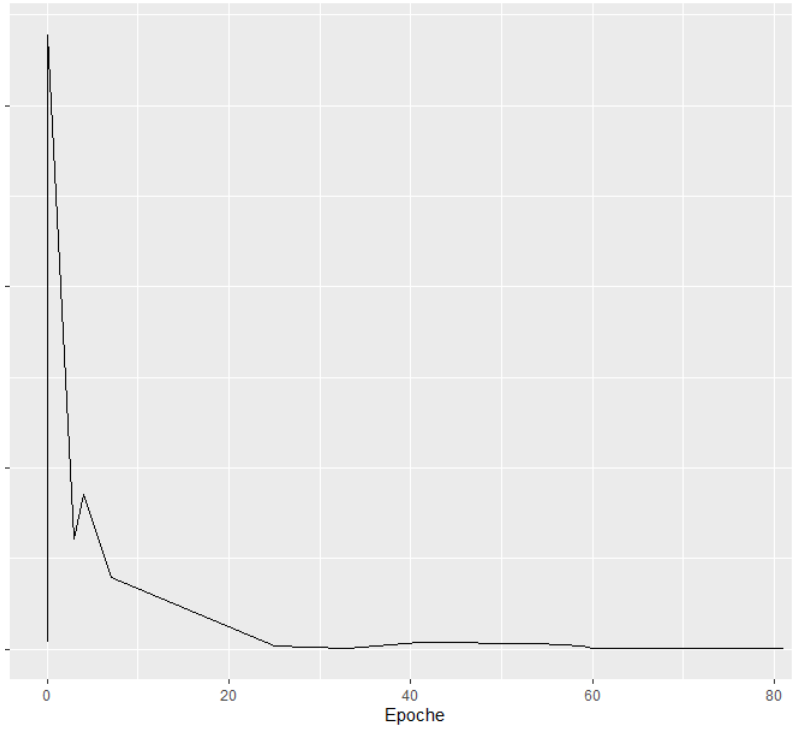

Abbildung 6.13: Verlustverlauf während des Trainings

Optimale Parameter mit einem Verlust von 0.009728432:
Initiale Infektionsrate 0.31933674, Sterberate 0.2152318

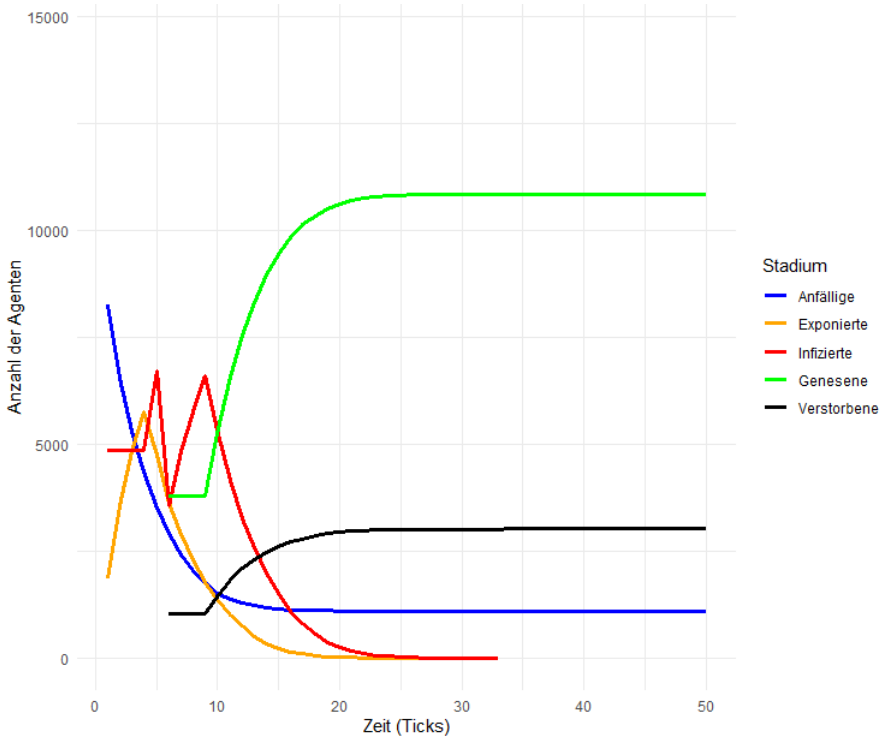

Abbildung 6.14: Krankheitsverlauf bei optimierten Parametern mit 3000 Todesfällen

Nach dem Training war das neuronale Netzwerk in der Lage, erfolgreich epidemiologische Parameter zu generieren, die zu der von mir vorgegebenen Anzahl von 3000 Todesfällen geführt haben.

6.5 Diskussion

Die ähnlichen Krankheitsverläufe der verschiedenen Modelle bestätigen, dass alle korrekt funktionieren. Kleinere Abweichungen sind auf die stochastische Natur der Infektionsübertragung und des Übergangs zwischen den Kompartments zurückzuführen.

Die Skalierungsergebnisse sind nicht überraschend. Obwohl das model-epidemic-spread-gradabm Modell ausschließlich mit Tensoren arbeitet, führt die One-Hot-Tensor-Lösung in der TensorGraphEnvironment Klasse zu einem unnötig großen Tensor, was die Skalierbarkeit beeinträchtigt. Das model-epidemic-spread-self-contained Modell schneidet aufgrund der individuellen Tensoren für jeden Agenten ebenfalls schlechter ab, da die Recheneffizienz von Tensoren erst bei größeren Berechnungen Wirkung zeigt.

Das model-epidemic-spread-combined Modell ist am schnellsten, da das Verlaufsmodell der Simulation durch große Tensoren berechnet wird. Allerdings nimmt die Geschwindigkeit bei zunehmender Agentenzahl ab, da das Übertragungsmodell weiterhin von den Agenten ausgeführt wird. Die Analyse mit dem Rider Profiler zeigte, dass die Interact-Methode der Agenten 1,25-mal mehr Zeit benötigt als die PostTick-Methode der InfectionLayer Klasse.

Der Einsatz der Grafikkarte GTX 1070 brachte keine Leistungsverbesserung, was möglicherweise auf das Alter der Grafikkarte zurückzuführen ist, da sie bereits seit 2016 auf dem Markt ist.

7 Fazit und Ausblick

Das von mir entwickelte Modell model-epidemic-spread-combined konnte erfolgreich mithilfe eines FFN optimierte Parameter generieren, die zu dem angestrebten Zielwert der Anzahl der Todesfälle am Ende der Simulation führten. Da ein Transformer im Wesentlichen ein neuronales Netzwerk ist, könnte das Modell durch ein Kalibrierungsnetzwerk erweitert werden, das ähnlich wie GRADABM, aus epidemiologischen Daten lernt und daraus Parameter generiert. Durch die Auslagerung des Verlaufsmodells auf Tensor-Berechnungen konnte ich eine erhebliche Geschwindigkeitssteigerung im Vergleich zu einem herkömmlichen agentenbasierten Modell (ABM) erzielen. Allerdings gelang es mir nicht, das Modell so zu gestalten, dass der Parameter der effektiven Reproduktionszahl R optimiert wird. Die Optimierungen beschränken sich auf die im Verlaufsmodell verwendeten Parameter. Der Grund dafür ist das ich leider keine effiziente Methode finden konnte, um die Ableitung der Reproduktionszahl zu berechnen.

Modelle wie das θ-SEIRHD werden meiner Meinung nach weiterhin in der Zukunft nützlich sein, da sie eine schnellere Annäherung an die epidemiologische Ausbreitung ermöglichen, insbesondere wenn nur wenige Daten vorliegen. Trotz der Geschwindigkeit von Modellen wie θ-SEIRHD oder GRADABM ist es wichtig zu erkennen, dass sie bei der Simulation von Übertragungsdynamiken Einschränkungen haben und niemals die Detailtiefe eines ABM erreichen können, das die Übertragung und menschliche Interaktionen detaillierter simuliert. ABMs ermöglichen zudem eine realistischere Umsetzung von Interventionen. Obwohl die Erstellung eines präzisen Modells aufwendig ist, wird die Wahrscheinlichkeit zukünftiger Pandemien steigen, was realistischere Modelle erfordert. Wie Leiserson u. a. (2020) in ihrem Paper betonen, wird Hardware zunehmend auf Parallelverarbeitung optimiert, auch wenn sie durch das Moore'sche Gesetz irgendwann ihre Leistungsgrenzen erreicht. Somit wird die Simulation mit ABMs von Jahr zu Jahr performanter. Die Qualität der Vorhersagen durch Näherungsverfahren wird jedoch irgendwann an ihre Grenzen stoßen. Deshalb sollten wir weiterhin an der Verbesserung der Simulationstechniken für agentenbasierte Modelle arbeiten um deren Leistung zu steigern.

Literaturverzeichnis

[Abueg u. a. 2021] Abueg, Matthew ; Hinch, Robert ; Wu, Neo ; Liu, Luyang ; Probert, William ; Wu, Austin ; Eastham, Paul ; Shafi, Yusef ; Rosencrantz, Matt ; Dikovsky, Michael ; Cheng, Zhao ; Nurtay, Anel ; Abeler-Dörner, Lucie ; Bonsall, David ; McConnell, Michael V. ; O'Banion, Shawn ; Fraser, Christophe: Modeling the Effect of Exposure Notification and Non-Pharmaceutical Interventions on COVID-19 Transmission in Washington State. In: *npj Digital Medicine* (2021), März, S. 49. – URL https://www.nature.com/articles/s41746-021-00422-7. – ISSN 2398-6352

[Andrychowicz u. a. 2016] Andrychowicz, Marcin ; Denil, Misha ; Gomez, Sergio ; Hoffman, Matthew W. ; Pfau, David ; Schaul, Tom ; Shillingford, Brendan ; De Freitas, Nando: Learning to learn by gradient descent by gradient descent. In: *Advances in neural information processing systems* 29 (2016). – URL https://arxiv.org/abs/1606.04474

[Arik u. a. 2020] Arik, Sercan ; Li, Chun-Liang ; Yoon, Jinsung ; Sinha, Rajarishi ; Epshteyn, Arkady ; Le, Long ; Menon, Vikas ; Singh, Shashank ; Zhang, Leyou ; Nikoltchev, Martin u. a.: Interpretable sequence learning for COVID-19 forecasting. In: *Advances in Neural Information Processing Systems* 33 (2020), S. 18807–18818. – URL https://arxiv.org/abs/2008.00646

[Balcan u. a. 2009] Balcan, Duygu ; Colizza, Vittoria ; Gonçalves, Bruno ; Hu, Hao ; Ramasco, José J. ; Vespignani, Alessandro: Multiscale Mobility Networks and the Spatial Spreading of Infectious Diseases. In: *Proceedings of the National Academy of Sciences* (2009), Dezember, S. 21484–21489. – URL https://pnas.org/doi/full/10.1073/pnas.0906910106. – ISSN 0027-8424, 1091-6490

[Baran 2024] Baran, Ersan: *Implementation of Model Epidemic Spread Combined*. May 2024. – URL https://github.com/ersba/model-epidemic-spread-combined

[Baydin u. a. 2018] Baydin, Atilim G. ; Pearlmutter, Barak A. ; Radul, Alexey A. ; Siskind, Jeffrey M.: *Automatic Differentiation in Machine Learning: A Survey*. Februar 2018. – URL http://arxiv.org/abs/1502.05767

[Blondel u. a. 2020] Blondel, Mathieu ; Teboul, Olivier ; Berthet, Quentin ; Djolonga, Josip: *Fast Differentiable Sorting and Ranking*. Juni 2020. – URL http://arxiv.org/abs/2002.08871

[Bonabeau 2002] Bonabeau, Eric: Agent-Based Modeling: Methods and Techniques for Simulating Human Systems. In: *Proceedings of the National Academy of Sciences* (2002), Mai. – URL https://pnas.org/doi/full/10.1073/pnas.082080899. – ISSN 0027-8424, 1091-6490

[Buchner 2024] Buchner: *Automatisches Differenzieren*. 2024. – URL https://florianbuchner.com/automatisches-differenzieren/

[Chen u. a. 2020] Chen, Simiao ; Yang, Juntao ; Yang, Weizhong ; Wang, Chen ; Bärnighausen, Till: COVID-19 control in China during mass population movements at New Year. In: *The Lancet* 395 (2020), März, Nr. 10226, S. 764–766. – URL http://dx.doi.org/10.1016/s0140-6736(20)30421-9. – ISSN 0140-6736

[Chopra u. a. 2023] Chopra, Ayush ; Quera-Bofarull, Arnau ; Rodríguez, Alexander ; Krishnamurthy, Balaji ; Subramanian, Jayakumar ; Prakash, B A. ; Raskar, Ramesh: Differentiable Agent-based Epidemiology. (2023). – URL https://arxiv.org/abs/2207.09714

[Constandache und Leon 2012] Constandache, Alexandra ; Leon, Florin: A .NET Reinforcement Learning Platform for Multiagent Systems. In: *Bulletin of the Polytechnic Institute of Iasi, section Automatic Control and Computer Science* LVIII (LXII) (2012), 07, S. 93–112. – URL https://www.researchgate.net/publication/265110533_A_NET_Reinforcement_Learning_Platform_for_Multiagent_Systems

[Dayhoff und DeLeo 2001] Dayhoff, Judith E. ; DeLeo, James M.: Artificial neural networks: Opening the black box. In: *Cancer* 91 (2001), Nr. S8, S. 1615–1635. – URL http://dx.doi.org/10.1002/1097-0142(20010415)91:8. – ISSN 1097-0142

[Delamater u. a. 2019] Delamater, P. ; Street, Erica J. ; Leslie, Timothy F. ; Yang, Y. T. ; Jacobsen, K.: Complexity of the Basic Reproduction Number (R0).

In: *Emerging Infectious Diseases* 25 (2019), S. 1 – 4. – URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6302597/

[DiPietro und Hager 2020] DiPietro, Robert ; Hager, Gregory D.: *Deep learning: RNNs and LSTM*. S. 503–519. In: *Handbook of Medical Image Computing and Computer Assisted Intervention*, Elsevier, 2020. – URL http://dx.doi.org/10.1016/b978-0-12-816176-0.00026-0. – ISBN 9780128161760

[Dorri u. a. 2018] Dorri, Ali ; Kanhere, Salil S. ; Jurdak, Raja: Multi-Agent Systems: A Survey. In: *IEEE Access* 6 (2018), S. 28573–28593. – URL http://dx.doi.org/10.1109/access.2018.2831228. – ISSN 2169-3536

[Doutor u. a. 2016] Doutor, Paulo ; Rodrigues, Paula ; Soares, Maria do C. ; Chalub, Fabio A. C. C.: Optimal Vaccination Strategies and Rational Behaviour in Seasonal Epidemics. In: *Journal of Mathematical Biology* (2016), Dezember. – URL http://arxiv.org/abs/1507.02940. – ISSN 0303-6812, 1432-1416

[Elliott 2018] Elliott, Conal: The simple essence of automatic differentiation. In: *Proceedings of the ACM on Programming Languages* 2 (2018), Juli, Nr. ICFP, S. 1–29. – URL http://dx.doi.org/10.1145/3236765. – ISSN 2475-1421

[For the Influenza Forecasting Contest Working Group u. a. 2016] For the Influenza Forecasting Contest Working Group ; Biggerstaff, Matthew ; Alper, David ; Dredze, Mark ; Fox, Spencer ; Fung, Isaac Chun-Hai ; Hickmann, Kyle S. ; Lewis, Bryan ; Rosenfeld, Roni ; Shaman, Jeffrey ; Tsou, Ming-Hsiang ; Velardi, Paola ; Vespignani, Alessandro ; Finelli, Lyn: Results from the Centers for Disease Control and Prevention's Predict the 2013–2014 Influenza Season Challenge. In: *BMC Infectious Diseases* (2016), Dezember, S. 357. – URL https://bmcinfectdis.biomedcentral.com/articles/10.1186/s12879-016-1669-x. – ISSN 1471-2334

[Hethcote 2000] Hethcote, Herbert W.: The Mathematics of Infectious Diseases. In: *SIAM Review* 42 (2000), Januar, Nr. 4, S. 599–653. – URL http://dx.doi.org/10.1137/s0036144500371907. – ISSN 1095-7200

[Hinch u. a. 2021] Hinch, Robert ; Probert, William J. M. ; Nurtay, Anel ; Kendall, Michelle ; Wymant, Chris ; Hall, Matthew ; Lythgoe, Katrina ; Bulas Cruz, Ana ; Zhao, Lele ; Stewart, Andrea ; Ferretti, Luca ; Montero,

Daniel ; Warren, James ; Mather, Nicole ; Abueg, Matthew ; Wu, Neo ; Legat, Olivier ; Bentley, Katie ; Mead, Thomas ; Van-Vuuren, Kelvin ; Feldner-Busztin, Dylan ; Ristori, Tommaso ; Finkelstein, Anthony ; Bonsall, David G. ; Abeler-Dörner, Lucie ; Fraser, Christophe: OpenABM-Covid19—An Agent-Based Model for Non-Pharmaceutical Interventions against COVID-19 Including Contact Tracing. In: *PLOS Computational Biology* (2021), Juli, S. e1009146. – URL https://dx.plos.org/10.1371/journal.pcbi.1009146. – ISSN 1553-7358

[Ivorra u. a. 2020] Ivorra, B. ; Ferrández, M. ; Vela-Pérez, M. ; Ramos, A.: Mathematical modeling of the spread of the coronavirus disease 2019 (COVID-19) taking into account the undetected infections. The case of China. In: *Communications in Nonlinear Science & Numerical Simulation* 88 (2020), S. 105303 – 105303

[Jang u. a. 2016] Jang, Eric ; Gu, S. ; Poole, Ben: Categorical Reparameterization with Gumbel-Softmax. In: *ArXiv* abs/1611.01144 (2016). – URL https://arxiv.org/abs/1611.01144

[Jang u. a. 2024] Jang, Eric ; Gu, Shixiang S. ; Poole, Ben: *Gumbel-Softmax*. May 2024. – URL https://youtu.be/JFgXEbgcT7g?si=ZXEV9o3_TTe9SgXD&t=307

[Kass 2024] Kass, Dmitrij: *Gradient Descent with Linear Regression from Scratch*. Apr 2024. – URL https://dmitrijskass.netlify.app/2021/04/03/gradient-descent-with-linear-regression-from-scratch/

[Keeling und Eames 2005] Keeling, Matt J. ; Eames, Ken T.: Networks and epidemic models. In: *Journal of The Royal Society Interface* 2 (2005), Juni, Nr. 4, S. 295–307. – URL http://dx.doi.org/10.1098/rsif.2005.0051. – ISSN 1742-5662

[Kipf 2024] Kipf, Thomas: *Graph Convolutional Networks*. May 2024. – URL https://tkipf.github.io/graph-convolutional-networks/

[Kossaifi u. a. 2019] Kossaifi, Jean ; Panagakis, Yannis ; Anandkumar, Anima ; Pantic, Maja: TensorLy: tensor learning in python. In: *Journal Of Machine Learning Research* 20 (2019), Januar, Nr. 26, S. 925–930. – URL https://jmlr.org/papers/volume20/18-277/18-277.pdf

[LeCun u. a. 2015] LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *Nature* 521 (2015), Mai, Nr. 7553, S. 436–444. – URL http://dx.doi.org/10.1038/nature14539. – ISSN 1476-4687

[Leiserson u. a. 2020] Leiserson, Charles E. ; Thompson, Neil C. ; Emer, Joel S. ; Kuszmaul, Bradley C. ; Lampson, Butler W. ; Sanchez, Daniel ; Schardl, Tao B.: There's plenty of room at the Top: What will drive computer performance after Moore's law? In: *Science* 368 (2020), Juni, Nr. 6495. – URL http://dx.doi.org/10.1126/science.aam9744. – ISSN 1095-9203

[MARS-Group 2024a] MARS-Group: *MARS Multi-Agent Research & Simulation*. May 2024. – URL https://www.mars-group.org/

[MARS-Group 2024b] MARS-Group: *SmartOpenHamburg*. May 2024. – URL https://www.mars-group.org/docs/tutorial/soh/

[Mossong u. a. 2008] Mossong, Joël ; Hens, Niel ; Jit, Mark ; Beutels, Philippe ; Auranen, Kari ; Mikolajczyk, Rafael ; Massari, Marco ; Salmaso, Stefania ; Tomba, Gianpaolo S. ; Wallinga, Jacco ; Heijne, Janneke ; Sadkowska-Todys, Malgorzata ; Rosinska, Magdalena ; Edmunds, W. J.: Social Contacts and Mixing Patterns Relevant to the Spread of Infectious Diseases. In: *PLoS Medicine* (2008), März. – URL https://dx.plos.org/10.1371/journal.pmed.0050074. – ISSN 1549-1676

[Norton u. a. 2019] Norton, Kerri-Ann ; Gong, Chang ; Jamalian, Samira ; Popel, Aleksander: Multiscale Agent-Based and Hybrid Modeling of the Tumor Immune Microenvironment. In: *Processes* (2019), Januar, S. 37. – URL http://www.mdpi.com/2227-9717/7/1/37. – ISSN 2227-9717

[Pellis u. a. 2015] Pellis, Lorenzo ; Ball, Frank ; Bansal, Shweta ; Eames, Ken ; House, Thomas ; Isham, Valerie ; Trapman, Pieter: Eight Challenges for Network Epidemic Models. In: *Epidemics* (2015), März. – URL https://linkinghub.elsevier.com/retrieve/pii/S1755436514000334. – ISSN 17554365

[PyTorch 2024] PyTorch: *PyTorch: An Open Source Machine Learning Framework*. May 2024. – URL https://pytorch.org/docs/stable/index.html

[Romero-Brufau u. a. 2021] Romero-Brufau, Santiago ; Chopra, Ayush ; Ryu, Alex J. ; Gel, Esma ; Raskar, Ramesh ; Kremers, Walter ; Anderson, Karen S. ; Subramanian, Jayakumar ; Krishnamurthy, Balaji ; Singh, Abhishek ; Pasupathy, Kalyan ; Dong, Yue ; O'Horo, John C. ; Wilson, Walter R. ; Mitchell, Oscar ; Kingsley, Thomas C.: Public Health Impact of Delaying Second Dose of

BNT162b2 or mRNA-1273 Covid-19 Vaccine: Simulation Agent Based Modeling Study. In: *BMJ* (2021), Mai, S. n1087. – URL https://www.bmj.com/lookup/doi/10.1136/bmj.n1087. – ISSN 1756-1833

[Ruder 2017] Ruder, Sebastian: *An Overview of Gradient Descent Optimization Algorithms*. Juni 2017. – URL http://arxiv.org/abs/1609.04747

[SciSharp 2024] SciSharp: *TensorFlow.NET Documentation: Tensor*. May 2024. – URL https://scisharp.github.io/tensorflow-net-docs/#/components/tensor

[Sidiropoulos u. a. 2016] Sidiropoulos, N. ; Lathauwer, L. D. ; Fu, Xiao ; Huang, Kejun ; Papalexakis, E. ; Faloutsos, C.: Tensor Decomposition for Signal Processing and Machine Learning. In: *IEEE Transactions on Signal Processing* 65 (2016), S. 3551–3582

[Silva und Torres 2017] Silva, Cristiana J. ; Torres, Delfim F. M.: A SICA Compartmental Model in Epidemiology with Application to HIV/AIDS in Cape Verde. In: *Ecological Complexity* (2017), Juni, S. 70–75. – URL http://arxiv.org/abs/1612.00732. – ISSN 1476945X

[Stern 1996] Stern, Hal S.: Neural Networks in Applied Statistics. In: *Technometrics* 38 (1996), August, Nr. 3, S. 205. – URL http://dx.doi.org/10.2307/1270601. – ISSN 0040-1706

[Tabataba u. a. 2017] Tabataba, Farzaneh S. ; Chakraborty, Prithwish ; Ramakrishnan, Naren ; Venkatramanan, Srinivasan ; Chen, Jiangzhuo ; Lewis, Bryan ; Marathe, Madhav: A Framework for Evaluating Epidemic Forecasts. In: *BMC Infectious Diseases* (2017), Dezember, S. 345. – URL http://bmcinfectdis.biomedcentral.com/articles/10.1186/s12879-017-2365-1. – ISSN 1471-2334

[TensorFlow 2024] TensorFlow: *TensorFlow: An Open Source Machine Learning Framework for Everyone*. May 2024. – URL https://www.tensorflow.org/

[Vaswani u. a. 2017] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Ł u. ; Polosukhin, Illia: Attention is All you Need. In: Guyon, I. (Hrsg.) ; Luxburg, U. V. (Hrsg.) ; Bengio, S. (Hrsg.) ; Wallach, H. (Hrsg.) ; Fergus, R. (Hrsg.) ; Vishwanathan, S. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 30, Curran Associates, Inc., 2017. – URL https://proceedings.neurips.cc/pap

er_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[Wu u. a. 2020] Wu, Joseph T. ; Leung, Kathy ; Leung, Gabriel M.: Nowcasting and Forecasting the Potential Domestic and International Spread of the 2019-nCoV Outbreak Originating in Wuhan, China: A Modelling Study. In: *The Lancet* (2020), Februar, S. 689–697. – URL https://linkinghub.elsevier.com/retrieve/pii/S0140673620302609. – ISSN 01406736

[Zaghloul und Elsayed 2021] Zaghloul, Zaghloul S. ; Elsayed, Nelly: The FPGA Hardware Implementation of the Gated Recurrent Unit Architecture. In: *SoutheastCon 2021* (2021), S. 1–5. – URL https://www.researchgate.net/publication/352385793_The_FPGA_Hardware_Implementation_of_the_Gated_Recurrent_Unit_Architecture

[Zheng u. a. 2021] Zheng, Stephan ; Trott, Alexander ; Srinivasa, Sunil ; Parkes, David C. ; Socher, Richard: The AI Economist: Optimal Economic Policy Design via Two-level Deep Reinforcement Learning. In: *SSRN Electronic Journal* (2021). – URL http://dx.doi.org/10.2139/ssrn.3900018. – ISSN 1556-5068

A Anhang

A.1 model-epidemic-spread-gradabm

Obwohl in model-epidemic-spread-gradabm alle Berechnungen mithilfe von Tensoren durchgeführt werden, existieren dennoch objektbasierte Agenten. Diese Agenten haben die Aufgaben, einzelne Einträge aus dem Zustandstensor auszulesen und als Instanzvariablen zu speichern. Außerdem werden sie verwendet um die Altersgruppe jedes Agenten dem InfectionLayer zu übergeben. Diese Mechanismen dienen ausschließlich der Datenextraktion und Visualisierung. Die Simulationsergebnisse werden durch diese nicht beeinflusst.

A.1.1 Klassendiagramm

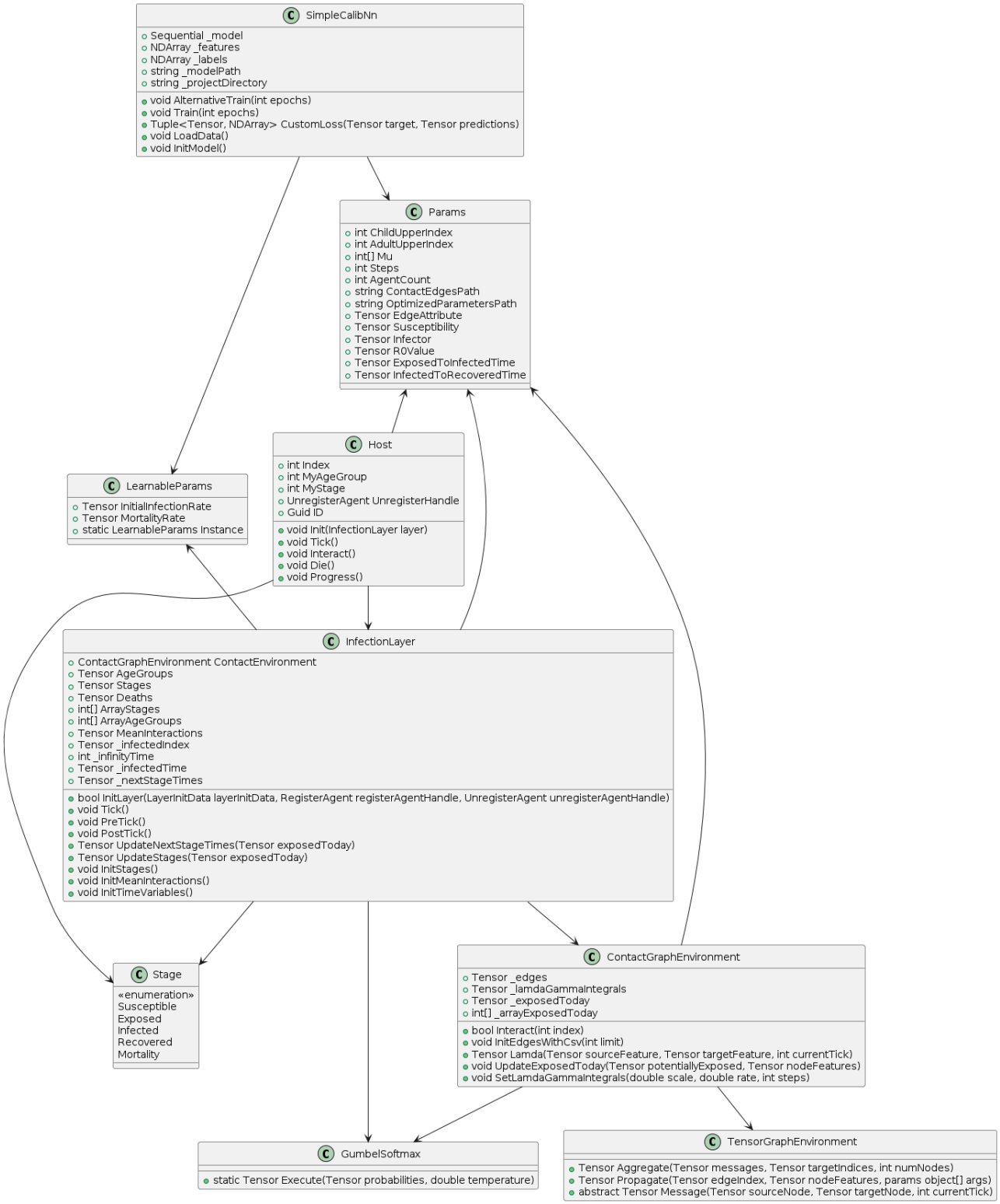

Abbildung A.1: Klassendiagramm von model-epidemic-spread-gradabm

A.1.2 Klassen

Aufgaben vom InfectionLayer

Bei der Initialisierung:

- Erstellung der Kontaktumgebung: ContactEnvironment wird als eine neue ContactGraphEnvironment Instanz erstellt. Diese Umgebung modelliert die Interaktionen zwischen den Agenten.
- Agenten-Erzeugung: Eine Liste von Agenten der Klasse Host wird erzeugt.
- Initialisierung der Stadien: InitStages wird aufgerufen, um die Infektionsstadien der Agenten zu initialisieren.
- Initialisierung der Zeitvariablen: InitTimeVariables wird aufgerufen, um den Zeitpunkt der Infektion und den Zeitpunkt zum Wechsel des Stadiums für die Agenten zu setzen.
- Initialisierung der Todesfälle: Deaths wird auf null gesetzt.

In einem Tick:

- Berechnung der Genesenen und Verstorbenen: recoveredAndDead stellt die Agenten dar, die im aktuellen Tick entweder genesen oder gestorben sind, basierend auf den Infektionsstadien und den Übergangszeiten. Die Anzahl der Verstorbenen wird mit Hilfe der Sterblichkeitsrate (_learnableParams.MortalityRate) ausgerechnet und mit Deaths aggregiert.
- Bestimmung der exponierten Agenten:
  1. Erstellung der Knotenmerkmale: nodeFeatures kombiniert die Eigenschaften der Agenten (Altersgruppen, Stadien, Infektionsstatus, Infektionszeit, durchschnittliche Interaktionen) zu einem Tensor.
  2. Aufruf des Kontaktgraphen: Die Methode Forward der ContactEnvironment wird ausgeführt, die die Ausbreitung der Infektion basierend auf den Knotenmerkmalen und dem aktuellen Tick simuliert.

- Aktualisierung der Stadien: UpdateStages wird aufgerufen, um die Infektionsstadien der Agenten basierend auf den im aktuellen Tick exponierten Agenten zu aktualisieren.
- Aktualisierung der Übergangszeiten der Stadien: UpdateNextStageTimes wird aufgerufen, um die Zeiten, zu denen die Agenten in das nächste Stadium übergehen, zu aktualisieren.
- Aktualisierung der Infektionszeiten: _infectedTime wird aktualisiert, um die Zeiten der neuen Infektionen zu erfassen.
- Aktualisierung der Stadien als Array: ArrayStages wird aktualisiert, um die Stadien der Agenten als Array darzustellen. Aus diesem Array lesen die Host Agenten in jedem Tick ihren Zustand aus.

Aufgaben vom Host

Bei der Initialisierung:

- Setzen der Altersgruppe: Die Altersgruppe des Agenten MyAgeGroup wird in das Array der Altersgruppen in der InfectionLayer eingefügt. Dies ermöglicht dem InfectionLayer die Altersgruppen aller Agenten zu erfahren.

In einem Tick:

- Interaktion: Die Methode Interact wird aufgerufen, um zu prüfen, ob der Agent mit anderen Agenten interagiert und möglicherweise exponiert wird. Sie liest dazu nur einen Eintrag aus dem ContactEnvironment.
- Fortschritt des Infektionsstadiums: Die Methode Progress wird aufgerufen, um den Zustand des Agenten zu aktualisieren, indem ein Eintrag aus ArrayStages gelesen wird.
- Überprüfung auf Tod: Wenn der Agent im Stadium Mortality ist, kann die Methode Die aufgerufen werden, um den Tod des Agenten zu simulieren und ihn aus der Simulation zu entfernen.

Aufgaben vom ContactGraphEnvironment

Bei der Initialisierung:

- Initialisierung der Kanten: InitEdgesWithCsv(Params.AgentCount) wird aufgerufen, um die Kanten des Kontaktgraphen aus einer CSV-Datei zu lesen. Diese Kanten repräsentieren die Verbindungen zwischen den Agenten.
- Setzen der Lambda-Gamma-Integrale: SetLamdaGammaIntegrals(5.15, 2.14, Params.Steps) wird aufgerufen, um Integrale für die Gamma-Verteilung zu setzen, die zur Berechnung der Infektionswahrscheinlichkeit verwendet werden.
- Initialisierung des Exposition-Arrays: _arrayExposedToday wird als Array initialisiert, das speichert, welche Agenten im aktuellen Tick exponiert wurden. Das Array wird von den Host Agenten gelesen.

In einem Tick:

Forward Methode(Vorwärtsdurchlauf des GNNs) -> Propagierung der Kanten:

- Aufruf der Message Methode: Für jedes Kantenpaar wird basierend auf den Features (Knoten) und den Kantenattributen eine Nachricht berechnet.
- Aufruf der Aggregate Methode: Die berechneten Nachrichten werden aggregiert um λ zu erhalten.
- Berechnung der Infektionswahrscheinlichkeit: Mit Hilfe von λ wird die Infektionswahrscheinlichkeit der Agenten berechnet.
- Bestimmung der exponierten Agenten: Die Infektionswahrscheinlichkeit wird der GumbelSoftmax Funktion übergeben um die exponierten Agenten zu erhalten.
- Rückgabe des Exposition-Tensors: Der Tensor _exposedToday wird zurückgegeben, der für jeden Agenten bestimmt ob dieser Agent im aktuellen Tick exponiert wurde.

Aufgaben vom TensorGraphEnvironment

Chopra u. a. (2023) verwendeten für die Implementierung eines GNNs die Message Passing Klasse, diese ist allerdings in C# nicht verfügbar, aus diesem Grund habe ich eine eigene Klasse geschrieben die dieselbe Funktion hat:

```
public abstract class TensorGraphEnvironment
{
  private Tensor Aggregate(Tensor messages, Tensor targetIndices, int
      numNodes)
  {
    var oneHotIndices = tf.one_hot(targetIndices, depth: numNodes);

    var expandedMessages = tf.expand_dims(messages, axis: -1);
    var weightedMessages = expandedMessages * oneHotIndices;

    var aggregatedMessages = tf.reduce_sum(weightedMessages, axis: 0);
    return aggregatedMessages;
  }

  protected Tensor Propagate(Tensor edgeIndex, Tensor nodeFeatures, params
      object[] args)
  {
    var sourceNode = tf.gather(edgeIndex, tf.constant(0));
    var targetNode = tf.gather(edgeIndex, tf.constant(1));
    var sourceFeature = tf.gather(nodeFeatures, sourceNode);
    var targetFeature = tf.gather(nodeFeatures, targetNode);

    var edgeMessages = Message(sourceFeature, targetFeature, (int) args[0]);
    var outFeatures = Aggregate(edgeMessages, targetNode,
    (int)nodeFeatures.shape[0]);
    return outFeatures;
  }

  protected abstract Tensor Message(Tensor sourceNode, Tensor targetNode,
      int currentTick);
}
```

Listing A.1: Implementierung vom TensorGraphEnvironment

Die Klasse TensorGraphEnvironment A.1 imitiert die Struktur eines typischen Graph Neural Networks (GNN) mit den Methoden Aggregate, Propagate und Message. Al-

lerdings ist die Zuordnung von Nachrichten zu den korrekten Zielknoten innerhalb der Aggregate-Methode nicht so einfach.

Eine einfache Lösung wäre die Verwendung einer Schleife, um jede Nachricht einzeln dem entsprechenden Zielknoten zuzuordnen. Dies würde jedoch die Berechnungsgeschwindigkeit erheblich verlangsamen, insbesondere bei großen Graphen.

Um dieses Problem zu umgehen, wird ein Trick eingesetzt: die One-Hot-Kodierung. Dabei wird für jeden Zielknoten eine Zeile in einem Tensor erstellt, die größtenteils mit Nullen gefüllt ist, außer an der Position des Zielknotenindex, wo eine 1 steht. Die Anzahl der Spalten in diesem Tensor entspricht der Anzahl der Agenten. Dieser One-Hot-Tensor wird dann mit einem weiteren Tensor multipliziert (expandedMessages), der die Nachrichten aller Agenten enthält. Durch diese Multiplikation werden die Nachrichten effektiv so verteilt, dass in jeder Spalte des resultierenden Tensors nur die Nachrichten für den Agenten stehen, dessen Index im One-Hot-Tensor den Wert 1 hat. Schließlich werden die Spalten dieses Tensors aufsummiert. Dadurch erhält man in jeder Spalte die Summe der Nachrichten für den jeweiligen Agenten, also die aggregierte Nachricht. Das Problem ist allerdings dass der One-Hot-Tensor die Form (Anzahl der Nachrichten x Anzahl der Agenten) hat. Bei Simulationen mit einer großen Anzahl von Agenten kann dieser Tensor sehr groß werden, was zu längeren Berechnungszeiten führt.

A.2 model-epidemic-spread-combined

A.2.1 Klassendiagramm

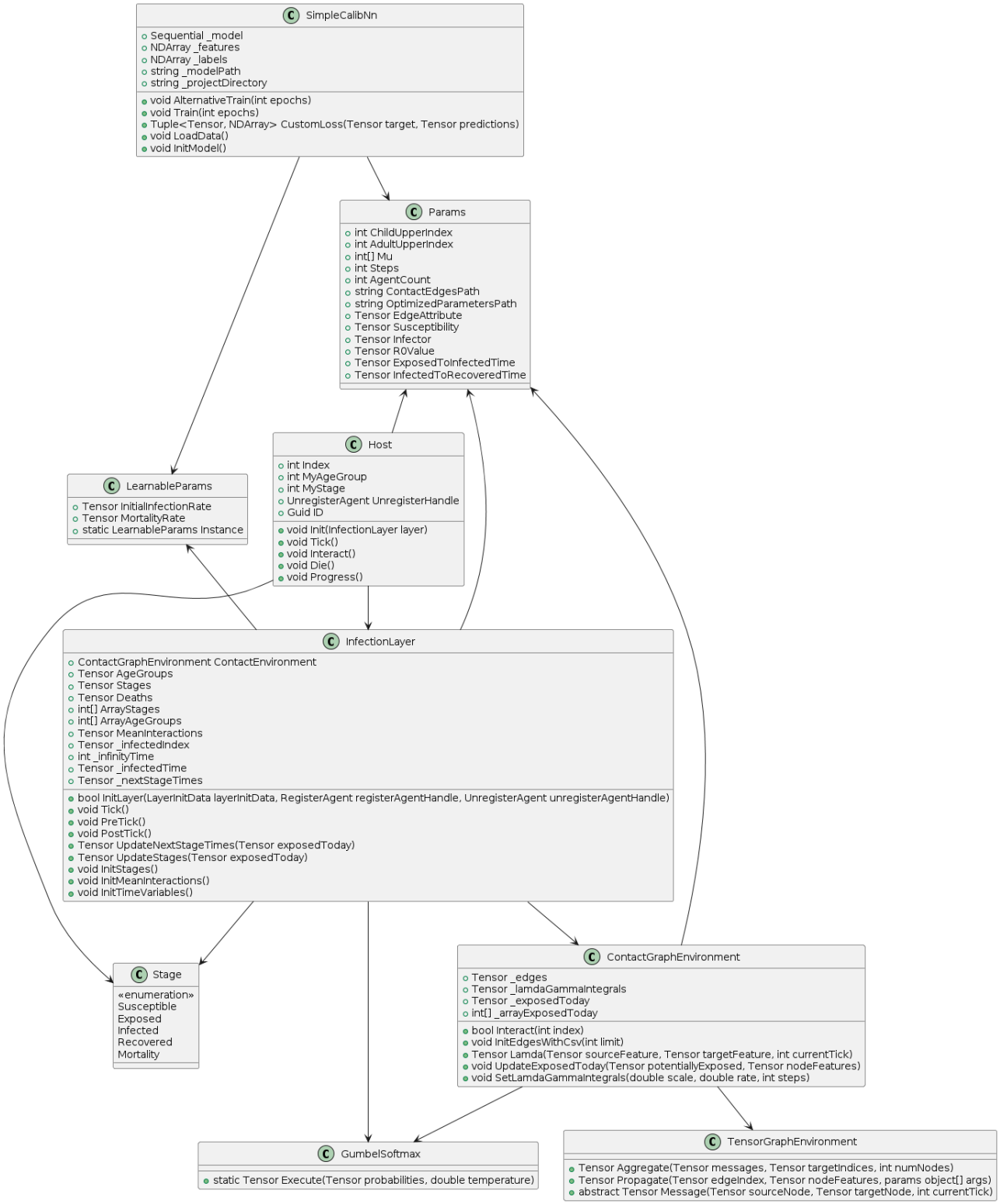

Abbildung A.2: Klassendiagramm von model-epidemic-spread-combined

A.2.2 Klassen

Aufgaben vom InfectionLayer

Bei der Initialisierung:

- Erstellung der Kontaktumgebung: ContactEnvironment wird als Instanz von ContactGraphEnvironment initialisiert, um die Interaktionen zwischen den Agenten zu modellieren.
- Agenten-Erzeugung: Eine Liste von Agenten der Klasse Host wird erzeugt.
- Initialisierung der Stadien: InitStages wird aufgerufen, um die Infektionsstadien der Agenten zu setzen.
- Initialisierung der Übergangszeiten: InitNextStageTimes wird aufgerufen, um die Zeiten, zu denen die Agenten in das nächste Infektionsstadium übergehen, zu setzen.
- Initialisierung der Gamma-Integrale: Host.InitLamdaGammaIntegrals wird aufgerufen, um notwendige Integrale für die Berechnung der Infektionswahrscheinlichkeiten zu setzen.
- Initialisierung des Exposition Arrays: ArrayExposedToday wird als Array mit nullen initialisiert. Dieser Array wird in jedem Tick von den Agenten mit einer eins am Index des Agenten befüllt wenn dieser exponiert ist.
- Lesen der CSV-Daten: ContactEnvironment.ReadCSV() wird aufgerufen, um die Kontaktdaten, die bestimmen welche Agenten in einem Tick mit einander interagieren, aus einer CSV-Datei zu lesen.
- Initialisierung der Todesfälle: Deaths wird auf null gesetzt.

Nach einem Tick:

- Berechnung der Genesenen und Verstorbenen: recoveredAndDead stellt die Agenten dar, die im aktuellen Tick entweder genesen oder gestorben sind, basierend auf den Infektionsstadien und den Übergangszeiten. Die Anzahl der Verstorbenen wird mit Hilfe der Sterblichkeitsrate _learnableParams.MortalityRate ausgerechnet und mit Deaths aggregiert.

- Bestimmung der exponierten Agenten: Der Tensor exposedToday wird aus ArrayExposedToday bestimmt.
- Aktualisierung der Stadien: UpdateStages wird aufgerufen, um die Infektionsstadien der Agenten basierend auf den im aktuellen Tick exponierten Agenten zu aktualisieren.
- Aktualisierung der Übergangszeiten der Stadien: UpdateNextStageTimes wird aufgerufen, um die Zeiten, zu denen die Agenten in das nächste Stadium übergehen, zu aktualisieren.
- Zurücksetzen des Exposition Arrays: ArrayExposedToday wird auf null gesetzt, für den nächsten Tick.
- Aktualisierung der Stadien als Array: ArrayStages wird aktualisiert, um die Stadien der Agenten als Array darzustellen. Aus diesem Array lesen die Host Agenten in jedem Tick ihren Zustand aus.

Aufgaben vom Host

Bei der Initialisierung:

- Einfügen in die Kontaktumgebung: Der Agent wird in die ContactEnvironment der InfectionLayer eingefügt, damit er mit anderen Agenten in der Umgebung interagieren kann.
- Setzen der Anfälligkeit: Die Anfälligkeit _susceptibility des Agenten wird basierend auf dessen Altersgruppe MyAgeGroup gesetzt, indem sie aus den globalen Parametern Params.Susceptibility abgerufen wird.
- Initialisierung des Infektionsstadiums: InitStage wird aufgerufen, um das aktuelle Stadium des Agenten zu setzen.
- Initialisierung der durchschnittlichen Interaktionen: InitMeanInteractions wird aufgerufen, um die durchschnittliche Anzahl der Interaktionen des Agenten basierend auf dessen Altersgruppe zu setzen.
- Initialisierung der Infektionszeit: InitInfectedTime wird aufgerufen, um die Zeit zu setzen, wie lange der Agent bereits infiziert ist, basierend auf seinem aktuellen Stadium.

In einem Tick:

- Zurücksetzen des Expositionsstatus: _exposedToday wird auf false gesetzt, um sicherzustellen, dass der Agent zu Beginn des Ticks nicht exponiert ist.
- Interaktionen mit anderen Agenten: Interact wird aufgerufen, um zu prüfen, ob der Agent mit anderen Agenten interagiert und möglicherweise exponiert wird. Wenn es zu einer Exponierung kommt wird _exposedToday auf true gesetzt und ArrayExposedToday der InfectionLayer Klasse am Index des Agenten auf 1 gesetzt.
- Fortschritt des Infektionsstadiums: Progress wird aufgerufen, um den Zustand des Agenten zu aktualisieren. Wenn der Agent exponiert wurde, wird der Zustand auf exponiert gesetzt, ansonsten wird der Zustand aus der ArrayStages Klasse des InfectionLayers am Index des Agenten ausgelesen.
- Überprüfung auf Tod: Wenn der Agent im Stadium Mortality ist, kann die Methode Die aufgerufen werden, um den Tod des Agenten zu simulieren und ihn aus der Simulation zu entfernen.

Aufgaben vom ContactGraphEnvironment

Bei der Initialisierung:

- Initialisieren der Sammlungen: Ein Dictionary _hosts, das die Hosts speichert, wobei der Schlüssel der Index des Hosts ist und der Wert eine Instanz der Host-Klasse wird initialisiert. Ein Dictionary _edges, das die Kontakte (Kanten) zwischen den Hosts speichert. Der Schlüssel ist der Index eines Hosts, und der Wert ist eine Liste der Nachbarn, mit denen der Host Kontakt hat.
- Insert Methode: Diese Methode wird von den Agenten aufgerufen und fügt sie als neuen Host in die Kontaktgraph-Umgebung ein. Der Host wird in das _hosts-Dictionary eingefügt, wobei der Index des Hosts als Schlüssel verwendet wird. Eine leere Liste für die Nachbarn des Hosts wird im _edges-Dictionary unter dem Index des Hosts erstellt.
- ReadCSV Methode: Diese Methode wird von der InfectionLayer Klasse aufgerufen. Diese Methode liest eine CSV-Datei ein, die die Kontakte zwischen den Hosts in einem Tick angibt. Jede Zeile der Datei enthält zwei Indizes, die einen Kontakt zwischen zwei Hosts darstellt. Für jedes Paar von Indizes werden die entsprechenden

Hosts aus dem _hosts-Dictionary abgerufen und gegenseitig als Nachbarn in das _edges-Dictionary eingefügt.

In einem Tick:

- GetNeighbors Methode. Diese Methode wird von den Hosts aufgerufen. Sie gibt dem aufrufenden Host eine Liste seiner Kontakte zurück. Es wird geprüft, ob der angegebene Host Nachbarn hat, indem der Index in _edges nachgeschlagen wird. Wenn Nachbarn vorhanden sind, wird die Liste der Nachbarn zurückgegeben. Ansonsten wird eine leere Liste zurückgegeben.

A.3 model-epidemic-spread-self-contained

A.3.1 Klassendiagramm

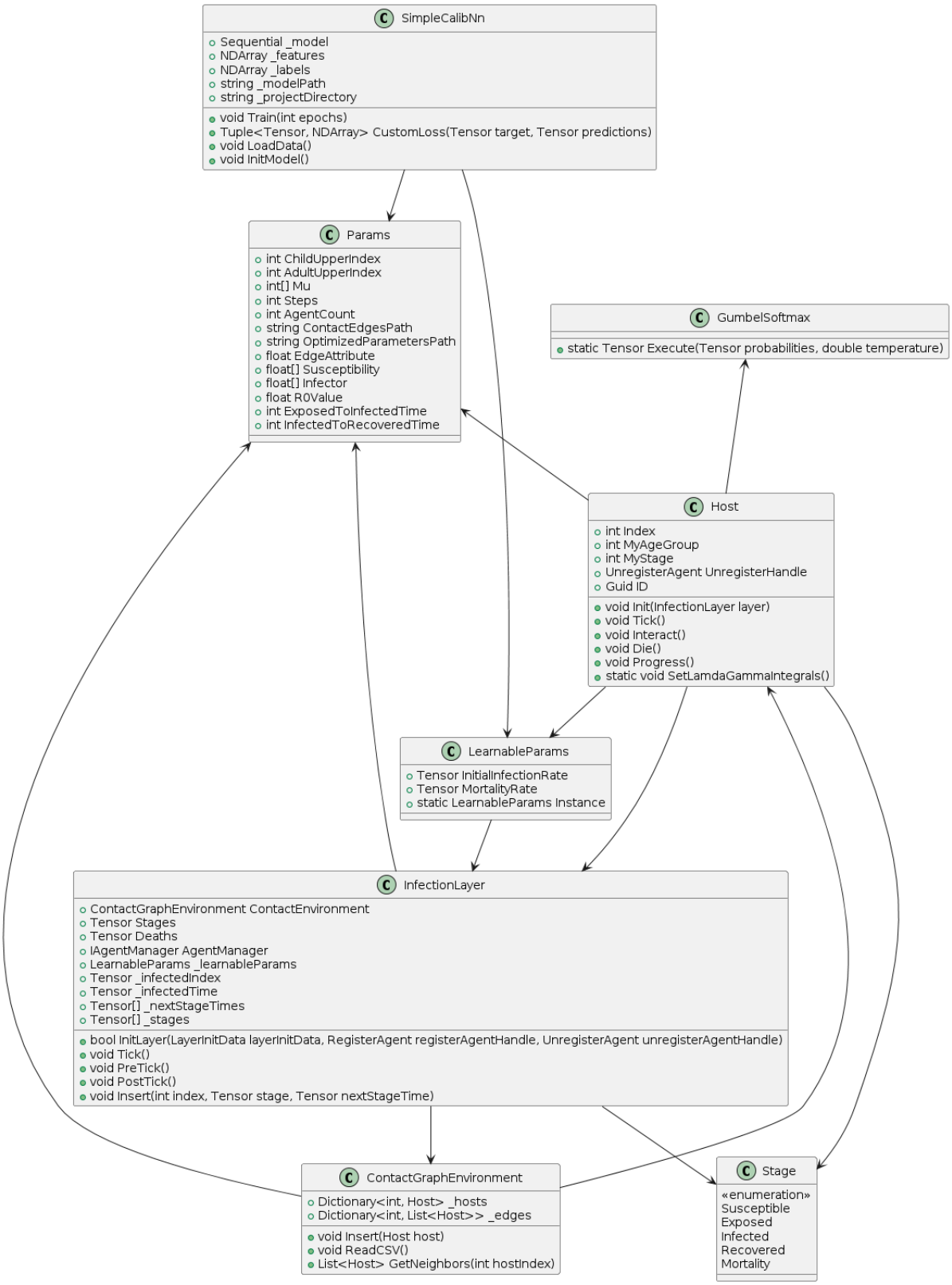

Abbildung A.3: Klassendiagramm von model-epidemic-spread-self-contained

A.3.2 Klassen

Aufgaben vom InfectionLayer

Bei der Initialisierung:

Die Aufgaben der InfectionLayer Klasse sind bei der Initialisierung identisch mit der InfectionLayer Klasse aus dem model-epidemic-spread-combined

In einem Tick:

Insert Methode: Die Methode wird von den Host Agenten aufgerufen. Sie übergibt dem InfectionLayer das aktuelle Stadium die Übergangszeit des Agenten.

Nach einem Tick:

- Berechnung der Genesenen und Verstorbenen: recoveredAndDead stellt die Agenten dar, die im aktuellen Tick entweder genesen oder gestorben sind, basierend auf den Infektionsstadien und den Übergangszeiten. Die Anzahl der Verstorbenen wird mit Hilfe der Sterblichkeitsrate _learnableParams.MortalityRate ausgerechnet und mit Deaths aggregiert.

Aufgaben vom Host

Bei der Initialisierung:

- Einfügen in die Kontaktumgebung: Der Agent wird in die ContactEnvironment der InfectionLayer eingefügt, damit er mit anderen Agenten in der Umgebung interagieren kann.
- Initialisierung des Infektionsstadiums: InitStage wird aufgerufen, um das aktuelle Stadium des Agenten zu setzen.
- Initialisierung der Zeitvariablen: InitTimeVariables wird aufgerufen, um die Zeit zu setzen, in der der Agent infiziert wurde und wann er in das nächste Stadium übergeht.
- Initialisierung der durchschnittlichen Interaktionen: InitMeanInteractions wird aufgerufen, um die durchschnittliche Anzahl der Interaktionen des Agenten basierend auf dessen Altersgruppe zu setzen.

- Setzen der Anfälligkeit: Die Anfälligkeit _susceptibility des Agenten wird basierend auf dessen Altersgruppe MyAgeGroup gesetzt, indem sie aus den globalen Parametern Params.Susceptibility abgerufen wird.

In einem Tick:

- Einfügen der aktuellen Daten: Die Methode fügt den aktuellen Zustand des Agenten und den Übergangszeitpunkt dem InfectionLayer hinzu, damit diese später aggregiert werden können.
- Zurücksetzen des Expositionsstatus: _exposedToday wird auf false gesetzt, um sicherzustellen, dass der Agent zu Beginn des Ticks nicht exponiert ist.
- Interaktionen mit anderen Agenten: Interact wird aufgerufen, um zu prüfen, ob der Agent mit anderen Agenten interagiert und möglicherweise exponiert wird. Wenn es zu einer Exponierung kommt wird _exposedToday auf true gesetzt.
- Fortschritt des Infektionsstadiums: Progress wird aufgerufen, um den Zustand des Agenten zu aktualisieren. Basierend darauf ob der Agent im aktuellen Tick exponiert wurde, wird der neue Zustand des Agenten berechnet und gesetzt.
- Überprüfung auf Tod: Wenn der Agent im Stadium Mortality ist, kann die Methode Die aufgerufen werden, um den Tod des Agenten zu simulieren und ihn aus der Simulation zu entfernen.

Ein wichtiger Unterschied zum Host in model-epidemic-spread-combined ist, dass dieser Host das Verlaufsmodell selber implementiert und nicht dafür das InfectionLayer verwendet.

Aufgaben vom ContactGraphEnvironment

Die Aufgaben der ContactGraphEnvironment Klasse sind identisch mit denen der ContactGraphEnvironment Klasse aus dem model-epidemic-spread-combined Modell.

A.4 model-epidemic-spread-without-tensors

A.4.1 Klassendiagramm

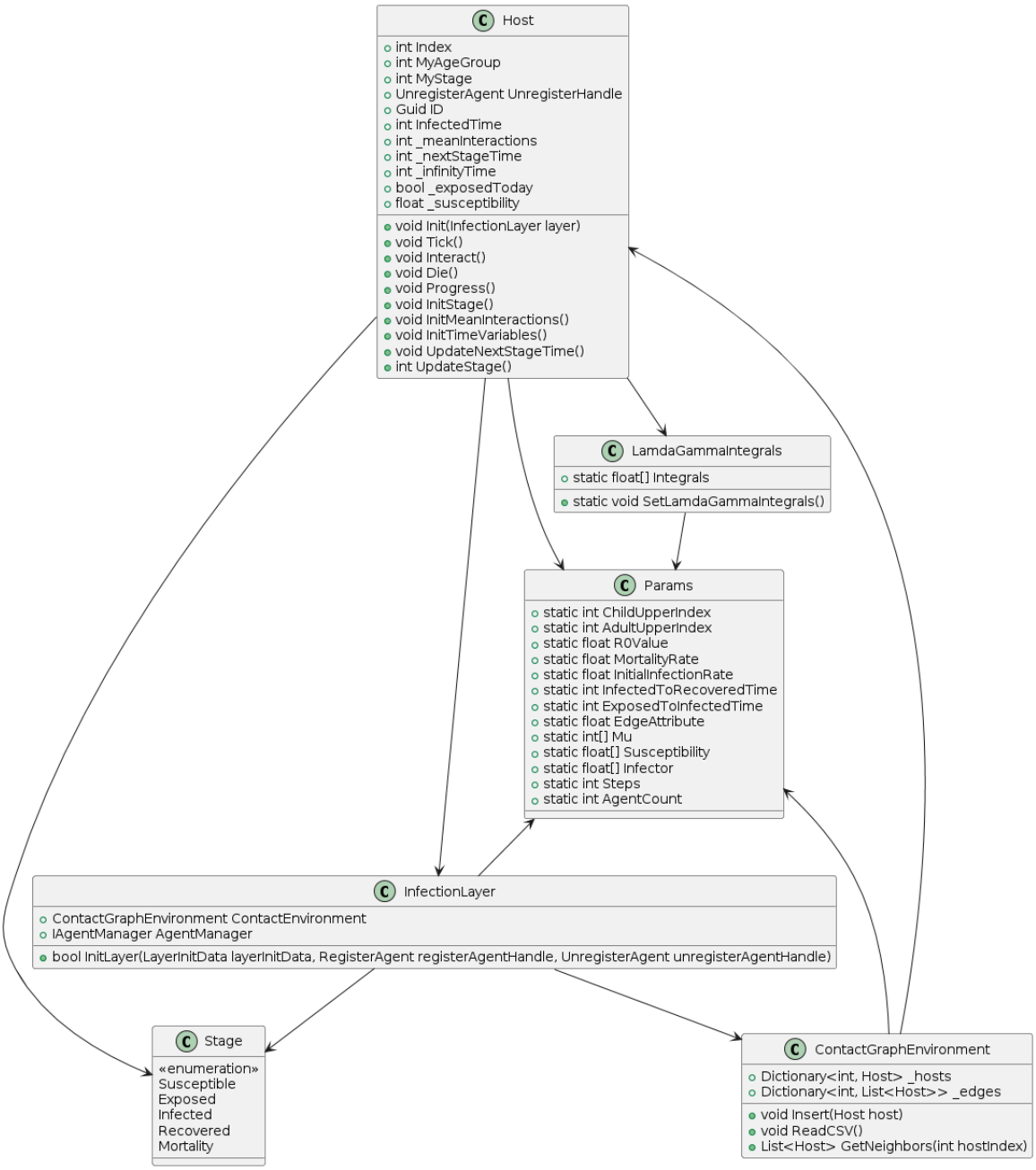

Abbildung A.4: Klassendiagramm von model-epidemic-spread-without-tensors

A.4.2 Klassen

Aufgaben vom InfectionLayer

Bei der Initialisierung:

- Erstellung der Kontaktumgebung: ContactEnvironment wird als Instanz von ContactGraphEnvironment initialisiert, um die Interaktionen zwischen den Agenten zu modellieren.
- Agenten-Erzeugung: Eine Liste von Agenten der Klasse Host wird erzeugt.
- Lesen der CSV-Daten: ContactEnvironment.ReadCSV() wird aufgerufen, um die Kontaktdaten, die bestimmen welche Agenten in einem Tick mit einander interagieren, aus einer CSV-Datei zu lesen.
- Initialisierung der Gamma-Integrale: LamdaGammaIntegrals.SetLamdaGammaIntegrals() wird aufgerufen, um notwendige Integrale für die Berechnung der Infektionswahrscheinlichkeiten zu setzen.

Die InfectionLayer Klasse agiert nicht während der Simulation.

Aufgaben vom Host

Bei der Initialisierung:

- Einfügen in die Kontaktumgebung: Der Agent wird in die ContactEnvironment der InfectionLayer eingefügt, damit er mit anderen Agenten in der Umgebung interagieren kann.
- Initialisierung des Infektionsstadiums: InitStage wird aufgerufen, um das aktuelle Stadium des Agenten zufällig basierend auf der initialen Infektionsrate zu setzen.
- Initialisierung der Zeitvariablen: InitTimeVariables wird aufgerufen, um die Zeit zu setzen, zu der der Agent infiziert wurde und wann er in das nächste Stadium übergeht.
- Setzen der Anfälligkeit: Die Anfälligkeit _susceptibility des Agenten wird basierend auf dessen Altersgruppe MyAgeGroup gesetzt, indem sie aus den globalen Parametern Params.Susceptibility abgerufen wird.

- Initialisierung der durchschnittlichen Interaktionen: InitMeanInteractions wird aufgerufen, um die durchschnittliche Anzahl der Interaktionen des Agenten basierend auf dessen Altersgruppe zu setzen.

In einem Tick:

- Zurücksetzen des Expositionsstatus: _exposedToday wird auf false gesetzt, um sicherzustellen, dass der Agent zu Beginn des Ticks nicht exponiert ist.
- Interaktionen mit anderen Agenten: Interact wird aufgerufen, um zu prüfen, ob der Agent mit anderen Agenten interagiert und möglicherweise exponiert wird. Wenn es zu einer Exponierung kommt wird _exposedToday auf true gesetzt.
- Fortschritt des Infektionsstadiums: Progress wird aufgerufen, um den Zustand des Agenten zu aktualisieren. Basierend darauf ob der Agent im aktuellen Tick exponiert wurde, wird der neue Zustand des Agenten berechnet und gesetzt. UpdateNextStageTime wird aufgerufen, um die Zeit zu aktualisieren, wann der Agent in das nächste Stadium übergeht. Wenn der Agent im aktuellen Tick exponiert wurde, wird InfectedTime auf den aktuellen Tick gesetzt.
- Überprüfung auf Tod: Wenn der Agent im Stadium Mortality ist, kann die Methode Die aufgerufen werden, um den Tod des Agenten zu simulieren und ihn aus der Simulation zu entfernen.

Aufgaben vom ContactGraphEnvironment

Die Aufgaben der ContactGraphEnvironment Klasse sind identisch mit denen der ContactGraphEnvironment Klasse aus dem model-epidemic-spread-combined Modell.

Aufgabe von LamdaGammaIntegrals

LamdaGammaIntegrals ist eine Utility Klasse. Die Methode SetLamdaGammaIntegrals wird von der InfectionLayer Klasse aufgerufen um die Integrale der Gamma-Verteilung zu berechnen. Die Berechnung dieser Integrale ist aufwendig und sie werden mehrmals verwendet. Aus diesem Grund werden sie einmal zum Beginn der Simulation berechnet und in Integrals gespeichert um sie bei Bedarf auszulesen.

A.4.3 Gemeinsame Klassen

Da alle von mir implementierten Modelle, dasselbe Problem lösen, verwenden sie auch identische Klassen, die im Folgenden näher erläutert werden:

Aufgabe von LearnableParams

Die Klasse LearnableParams ist eine Singleton-Klasse, die dazu dient die epidemiologischen Parameter aus dem Kalibrierungsnetzwerk zu erhalten.

Aufgabe von Params

Die Klasse Params dient als Referenz für verschiedene Parameter, die in der Simulation benötigt werden. Sie ermöglicht einen einfachen Zugriff auf diese Parameter.

Aufgabe von GumbelSoftmax

Die Klasse GumbelSoftmax implementiert die Gumbel-Softmax-Funktion.

```
1  public static class GumbelSoftmax
2  {
3    public static Tensor Execute(Tensor probabilities, double temperature =
         1.0)
4    {
5      var gumbelNoise =
           -tf.math.log(-tf.math.log(tf.random.uniform(probabilities.shape)));
6      var softSample = tf.nn.softmax((tf.math.log(probabilities + 1e-9) +
           gumbelNoise) / temperature);
7      var cutSoftSample = tf.constant(softSample.numpy());
8      var hardSample = tf.cast(tf.equal(softSample, tf.reduce_max(softSample,
           axis: 1, keepdims: true)),TF_DataType.TF_INT32);
9      softSample = tf.stop_gradient(hardSample - cutSoftSample) + softSample;
10     return softSample;
11   }
12 }
```

Listing A.2: Implementierung der Gumbel-Softmax-Funktion

Beim Einsatz von tf.stop_gradient() wird der Gradienten-Trick angewendet, um trotz diskreter Ergebnisse die Differenzierbarkeit zu gewährleisten. tf.stop_gradient() stoppt den Gradientenfluss durch den angegebenen Tensor oder die Operation. Jedoch trat bei der Verwendung von softSample = tf.stop_gradient(hardSample - softSample) + softSample ein Fehler auf. Um den Gradientenfluss bei der Variable softSample dennoch zu unterbrechen, wurde der Wert von softSample zu cutSoftSample zugewiesen.

Aufgaben von SimpleCalibNn

Die Klasse SimpleCalibNn implementiert das Kalibrierungsnetzwerk und ist dafür zuständig, ein einfaches neuronales Netzwerk (FFN) zu trainieren, um die epidemiologischen Parameter zu optimieren:

- Bei der Initialisierung: Die Zielwerte nach denen sich das NN optimieren soll werden geladen. Das FFN wird initialisiert. Dabei wird entweder ein vorhandenes Modell geladen oder ein neues Modell mit einer vordefinierten Architektur erstellt.
- Während der Trainingsschleife: Die Methode Train() führt den Trainingsprozess des neuronalen Netzwerks durch. Sie iteriert über eine festgelegte Anzahl von Epochen, berechnet in jeder Epoche die Modellvorhersagen und verwendet eine benutzerdefinierte Verlustfunktion CustomLoss(), um die Modellparameter anzupassen. Die Verlustfunktion integriert das Ausbreitungsmodell, um den Verlust basierend auf den simulierten Todesfällen und den vorgegebenen Zielwerten zu berechnen. Die besten epidemiologischen Parameter werden während des Trainings gespeichert.

```
using (var tape = tf.GradientTape())
{
  var predictions = (Tensor)_model.predict(np.array(10f)
  .reshape(new Shape(-1, 1)));
  (var loss, var boundedPredictions) = CustomLoss(_labels, predictions);

  if (loss.numpy() < bestEpochloss)
  {
    bestEpochloss = loss.ToArray<float>()[0];
    bestBoundedPredictions = boundedPredictions.ToArray<float>();
  }
  var gradients = tape.gradient(loss, _model.TrainableVariables);
  optimizer.apply_gradients(zip(gradients, _model.TrainableVariables));

  Console.WriteLine($"epoch: {epoch + 1}, loss: {loss.numpy()}");
  Console.Write("gradients: ");
  tf.print(gradients[7]);
}
```

Listing A.3: Implementierung der Trainingsschleife

Da in meinem Fall die Ausgabe des neuronalen Netzwerks nicht direkt mit einem Zielwert verglichen wird, sondern erst durch eine epidemiologische Simulation verarbeitet wird, musste die Trainingsschleife angepasst werden. Um mehr Flexibilität in der Trainingsschleife zu haben und direkten Zugriff auf die Gradienten zu erhalten, habe ich anstelle der üblichen fit-Methode eine eigene Optimierungsmethode A.3 implementiert, die mit einem GradientTape arbeitet. Dieser Ansatz erwies sich als hilfreich beim Experimentieren mit verschiedenen Optimierungsstrategien. Ein GradientTape ist ein Kontext, der innerhalb des using-Blocks alle Operationen aufzeichnet, die für die Berechnung der Gradienten relevant sind. Die implementierte Optimierungsmethode funktioniert wie folgt:

1. Das NN generiert seine Ausgaben, die den epidemiologischen Parametern entsprechen.
2. Diese Ausgaben werden zusammen mit den Zielwerten an die modifizierte Verlustfunktion übergeben.
3. Falls der berechnete Verlust geringer ist als alle bisherigen Verluste, die durch andere Parameter entstanden sind, werden die aktuellen Parameter gespeichert.
4. Mithilfe des Verlusts und der Neuronen wird der Gradient bestimmt.

5. Basierend auf dem Gradienten wird eine gradientenbasierte Optimierung durchgeführt.

6. Zur Kontrolle und Überwachung werden die Gradienten auf der Konsole ausgegeben.

```
private (Tensor, NDArray) CustomLoss(Tensor target, Tensor predictions)
{
  var lowerBounds = tf.constant(new [] {0.001f, 0.01f});
  var upperBounds = tf.constant(new [] {0.9f, 0.9f});
  var boundedPred = lowerBounds + (upperBounds - lowerBounds) * predictions;
  LearnableParams learnableParams = LearnableParams.Instance;

  Console.WriteLine("-----------------------------------------------------");
  Console.Write("parameters:");
  tf.print(boundedPred);
  learnableParams.InitialInfectionRate = boundedPred[0, 0];
  learnableParams.MortalityRate = boundedPred[0, 1];
  var predictedDeaths = Program.EpidemicSpreadSimulation(true);

  Console.Write("deaths: ");
  tf.print(predictedDeaths);

  var loss = tf.reduce_mean(tf.square(target - predictedDeaths));

  return (loss, boundedPred.numpy());
}
```

Listing A.4: Implementierung der Verlustfunktion

Die Methode CustomLoss A.4 ist die benutzerdefinierte Verlustfunktion, die zur Optimierung des neuronalen Netzwerks eingesetzt wird. Die Methode funktioniert wie folgt:

- Die übergebenen epidemiologischen Parameter werden auf einen gültigen Wertebereich begrenzt. Hierfür wird die gleiche Technik wie in Arik u. a. (2020) verwendet: $\theta = \theta_L + (\theta_U - \theta_L) \cdot \sigma(\text{FFN}(\mathbf{o}))$
  θ_L und θ_U sind die untere und obere Grenze der epidemiologischen Parameter θ und σ ist die Sigmoidfunktion.
- Die begrenzten Parameter werden an die LearnableParams-Instanz übergeben.
- Die Simulation wird ausgeführt, und das Ergebnis, die Anzahl der Todesfälle, wird in der Variable predictedDeaths gespeichert.
- Aus dem Zielwert und den simulierten Todesfällen wird der MSE berechnet. Dieser dient als Verlustwert für die Optimierung.

- Die Methode gibt sowohl den berechneten Verlust als auch die begrenzten epidemiologischen Parameter zurück.

A.5 Quellcode

Der Quellcode für das Modell model-epidemic-spread-combined ist unter folgendem Link verfügbar:

https://github.com/ersba/model-epidemic-spread-combined

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original