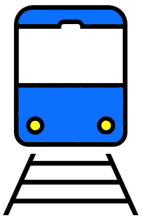
InformatiCup 2022 - Bahn-Simulator

Theoretische Ausarbeitung

von



Team Schmalegg HBF

Pascal Knoll (Knoll@Stud.dhbw-ravensburg.de)

Tom Zehle (Zehle@Stud.dhbw-ravensburg.de)

Luisa Ibele (Ibele@Stud.dhbw-ravensburg.de)

16. Januar 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

15figure.caption.18

Abkürzungsverzeichnis

CPU	Central Processing Unit
GNN	Graph Neural Networks
GPU	Graphics Processing Unit
MCTS	Monte Carlo Tree Search
NN	Neruonale Netze
PPO	Proximal Policy Optimization
PyG	PyTorch Geometric
QSAT	Quantified Boolean Formula Satisfaction
TRPO	Trust Region Policy Optimization

1 Einleitung

In diesem Kapitel geht es um die wichtigen Grundlagen, bevor anschließend im zweiten Kapitel die Details unserer genauen Lösung erläutert werden.
Dabei wird unter anderem auf die genaue Aufgabenstellung des InformatiCups 2022 eingegangen, der konkrete Projektablauf, der zur vorliegenden Lösung geführt hat, beschrieben und es werden mögliche Lösungsansätze für die Problemstellung diskutiert und ausgewertet.

1.1 Vorstellung des Teams

Unser Team besteht aus Pascal Knoll, Tom Zehle und Luisa Ibele.
Wir studieren dual im dritten Semester Wirtschaftsinformatik mit dem Schwerpunkt Data Science an der DHBW in Ravensburg.

1.2 Die Aufgabenstellung

Die Aufgabe des InformatiCups dieses Jahres besteht darin, einen Fahrplan für ein Schienennetzwerk zu erstellen. Dabei soll ein Programm entworfen werden, das die Berechnung von optimalen Fahrplänen für ein gegebenes Netzwerk vornimmt. Dadurch soll dieses Programm die Gesamtverspätung aller Fahrgäste dieses Schienennetzwerkes verringern und damit in einem weiteren Schritt die Zufriedenheit der Kunden mit dem gesamten Netzwerk steigern.

Generell wird das Schienennetzwerk hierbei in Runden ausgeführt, wobei eine Runde eine fiktive Zeiteinheit darstellt. Ähnlich zur Realität besteht das Bahnnetz aus vier Elementen: Bahnhöfe, Strecken, Passagiere und Züge. Diese Elemente haben jeweils verschiedene Eigenschaften, wie zum Beispiel einem Namen, beziehungsweise ID, oder einer gewissen Kapazität, beziehungsweise Größe. Während die Bahnhöfe und Strecken eine bestimmte Kapazität für Anzahl an Zügen haben, fahren die Züge in unterschiedlichen Geschwindigkeiten und starten von verschiedenen Startpositionen. Die Passagiere werden in Gruppen als Einheit zusammengefasst, wobei die Anzahl an Personen in einer Gruppe auch variiert und damit die Kapazität der entsprechenden Züge beeinflusst. Zusätzlich hat jede Passagiergruppe einen unterschiedlichen Start- und Zielbahnhof. Zwischen den Start- und Zielbahnhöfen liegen die Strecken, wobei diese unterschiedliche Längen haben und die Züge darauf in unterschiedlichen Geschwindigkeiten fahren. Was hierbei schnell klar wird, ist, dass die zuerst simpel erscheinende Aufgabenstellung durch diese einzelnen Details und Vorgaben einige Herausforderungen birgt und gegensätzlich zum ersten Eindruck einiges komplexer ist. Die vollständige Aufgabenstellung kann im GitHub-Repository des InformatiCups 2022 nachgelesen werden (https://github.com/informatiCup/informatiCup2022).

1.3 Der Projektablauf

Unseren Arbeitsverlauf für die Erarbeitung einer passenden Lösung für den InformatiCup lässt sich in verschiedene Schritte gliedern.
Dabei haben wir in einem ersten Schritt die Aufgabenstellung genauer angeschaut, bevor wir uns mit der Entwicklung eines konkreten Lösungsansatz beschäftigt haben. Das heißt, wir haben uns erst überlegt, inwiefern ein solcher Fahrplan entworfen werden kann und welche Details dabei genau beachtet werden müssen, beziehungsweise Einfluss auf den Entwurf haben. Um uns einen Überblick zu schaffen, haben wir ein UML-Diagramm entworfen, welches in Abbildung 1 zu sehen ist. Dieses Diagramm zeigt die einzelnen Elemente der Aufgabenstellung zusammen mit ihren Eigenschaften sowie die Verbindungen, beziehungsweise die Einflussnahme der einzelnen Eigen-

schaften auf andere Elemente und welche Informationen für andere Elemente für die Erstellung des Fahrplans benötigt werden. Auch allein durch die vielen Verbindung der einzelnen Elemente untereinander im Diagramm wird klar, dass es sich um ein komplexes logistisches Optimierungsproblem handelt.

In einem weiteren Schritt haben wir das bereitgestellte Git-Repository des InformatiCups genauer ins Auge genommen und die einzelnen Go und Text Dateien genauer analysiert, um uns eine Vorstellung davon zu machen, wie der Input, das heißt das Schienennetzwerk, und das Testen des erstellten Fahrplans später aussehen kann.
Danach haben wir uns dann mit den möglichen Ansätzen für die Lösung des gegebenen Problems auseinandergesetzt.
Wir haben unsere Lösung für den InformatiCup in verschiedenen Schritten erarbeitet. In einem ersten Schritt gilt es die Aufgabenstellung genauer zu analysieren, bevor die Grundrisse des Programms entworfen werden können. Dabei sind wir auf verschiedene Quellen und Ansätze (s. Kapitel 1.5) gestoßen, welche wir dann ausgewertet (s. Kapitel 1.6) und uns für einen möglichen Ansatz entschieden haben.
Und erst dann haben wir uns um die Umsetzung dieses spezifischen Lösungsansatzes, welcher in den folgenden Kapitel noch genauer analysiert wird, gekümmert und mit dem Programmieren in unseren gewählten Programmiersprache und Entwicklungsumgebung angefangen.

1.4 Related Works

Die Problemstellung der Aufgabe des InformatiCups 2022 beschreibt kein ungewöhnliches Problem. Logistische Probleme wie dieses, wo es darum geht aus mehreren möglichen Optionen die beste zu wählen mit Auswirkung auf die darauf folgenden möglichen Optionen, existieren in den unterschiedlichsten Bereichen. Sie gehören auch den sogenannten QSAT[1]-Problemen an. Ein berühmtes Beispiel für den Umgang mit genau diesen Problemen ist das Computerprogramm AlphaZero, dessen Algorithmus mehrere komplexe Brettspiele einzig anhand der Spielregeln und Siegbedingungen

[1] Quantified Boolean Formula Satisfaction

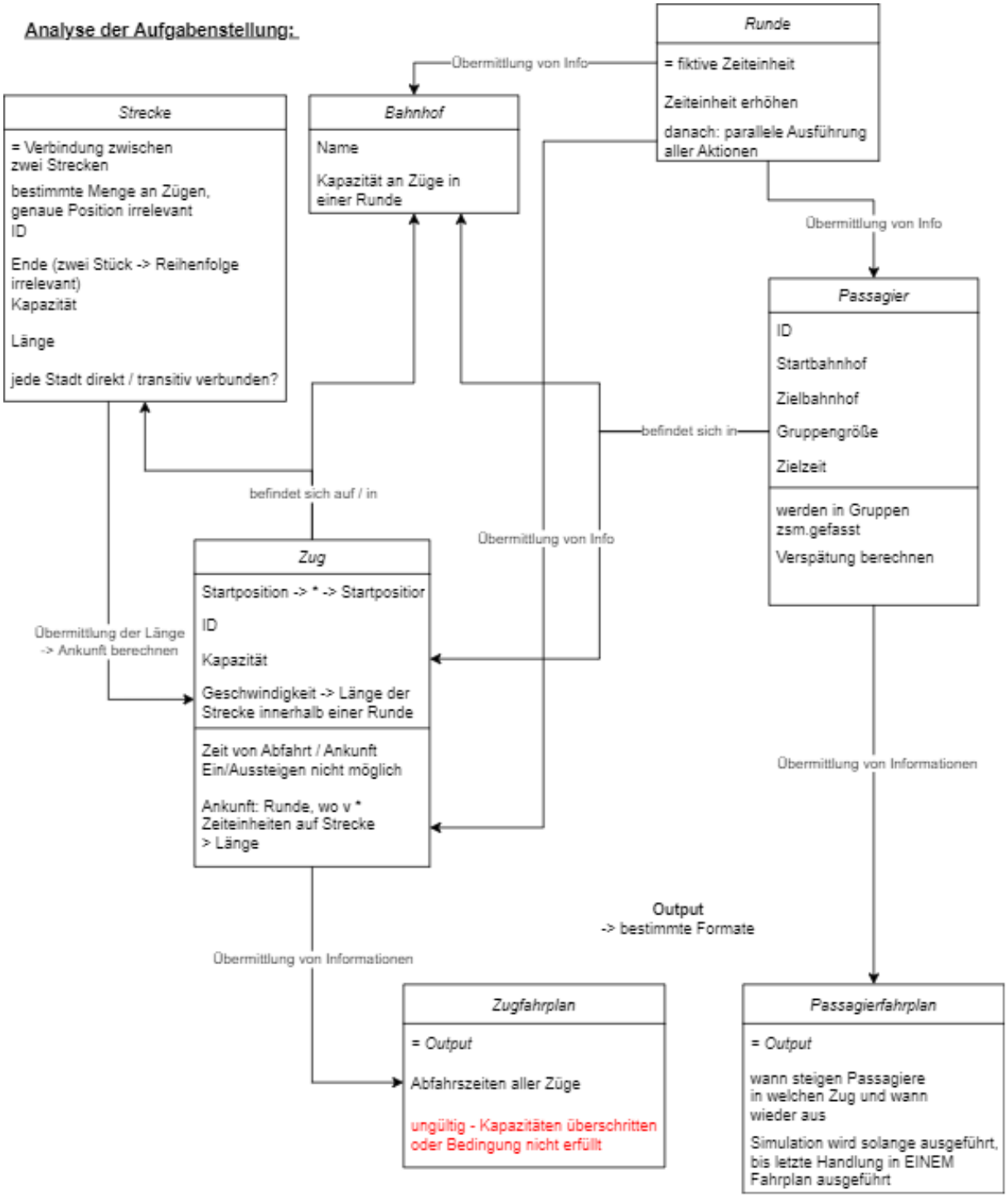


Abbildung 1: Analyse der Aufgabenstellung

sowie durch intensives Spielen gegen sich selbst erlernt. Dabei wurden bei AlphaZero Graph Neural Networks in Verbindung mit Reinforcement Learning und der Monte Carlo Tree Search, einem Baumsuchalgorithmus, kombiniert, wodurch das Programm in der Lage war, bestmögliche Spielzüge bei Spielen wie Schach, Go oder Shōgi zu

bestimmen.

Generell ist das Thema Reinforcement Learning in Verbindung mit (Graph) Neural Networks sehr populär in Sachen künstlicher Intelligenz und Data Science. Es bietet großes Potential und viel Rechenpower, um komplexe Problemstellungen durch automatisierte Lernprozesse zu lösen. Gerade in der Forschung wird derzeit Reinforcement Learning auf seine Möglichkeiten und Kapazitäten in unterschiedlichen Projekten getestet. Dies erkennt man auch daran, dass sich das übergeordnete Thema Machine Learning - Reinforcement ist als Teilbereich des Machine Learnings definiert - im Gartner Hype Cycle 2021 bereits auf dem Weg in den "Trog der Ernüchterung"befindet (s. Abbildung 2).

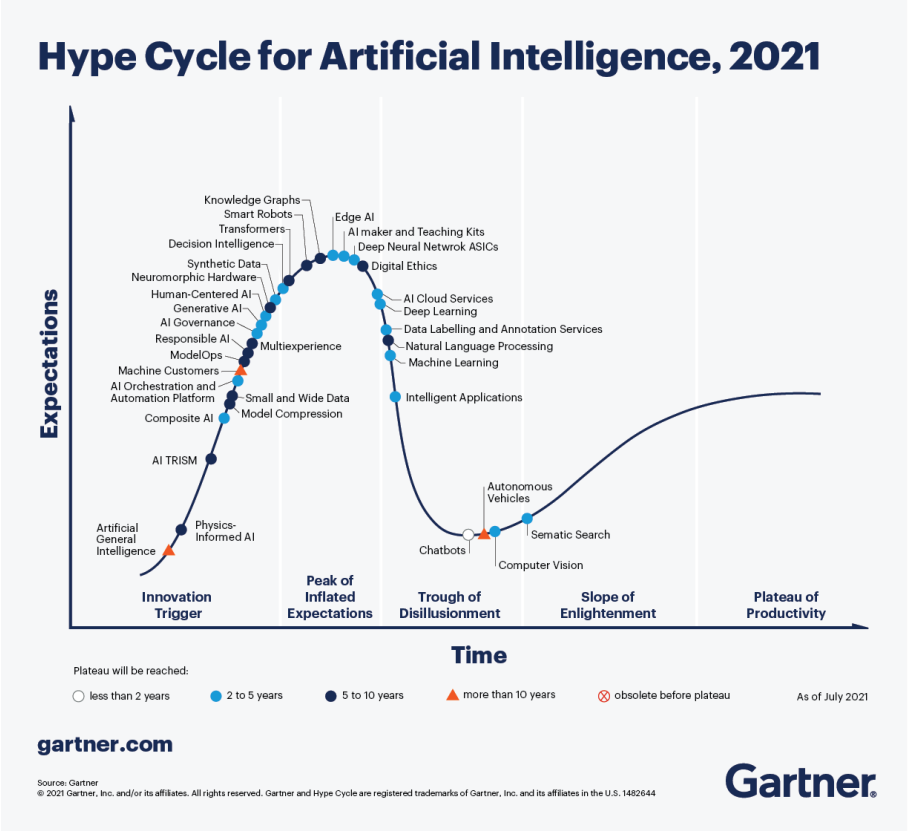


Abbildung 2: Gartner Hype Cycle 2021
(Vashisth, 2021)

Daraus lässt sich schießen, dass zum Themengebiet des Machine Learnings und somit auch zu dem des Reinforcement Learnings bereits einige Forschungsprojekte zur Erkundung dieser durchgeführt wurden. Dadurch existieren dazu auch Literatur und praktische Anwendungsbeispiele.

1.5 Mögliche Lösungsansätze

Generell ist anfangs zu überlegen, welche Möglichkeiten es gibt, um eine solche Software, die für die Berechnung von optimalen Fahrplänen zuständig ist, zu entwickeln.
Dabei kommt einem natürlich als erstes der traditionelle, regelbasierte Ansatz in den Sinn. Hierbei könnte anhand von verschiedener Algorithmen die Aufgabenstellung in der gewünschten Programmiersprache abgehandelt und ein optimaler Fahrplan erstellt werden.
In heutigen Zeiten, wo Künstliche Intelligenz und Machine Learning auf dem Vorreiter sind, kommt aber auch in einem weiteren Schritt die Frage auf, ob ein „intelligenter" Ansatz des maschinellen Lernens eine kompetente Lösung für das vorliegende Problem liefern könnte.
Also steht man als erstes vor der prinzipiellen Frage, ob man eher einen regelbasierten oder einen automatisierten, intelligenten Ansatz wählen möchte. Beide dieser groben Ansätze bieten ihre Vorteile, halten aber auch gewisse Herausforderungen verborgen. Jedoch ist diese Frage nicht die allumfassende Frage, die zur Wahl einer Lösungsstrategie und einer optimal erfolgreichen Lösung führt, denn sowohl der regelbasiert als auch der automatisierte Ansatz bieten einen Blumenstrauß an weiteren detaillierten Methoden. Wenn man sich diese genauer anschaut, kommen unter anderem folgende Ansätze in Frage: Während sich regelbasiert beispielsweise eine Lösung mit dem Dijkstra oder dem Bellman-Ford-Algorithmus entwickeln ließe, könnte man bei einem Machine Learning Ansatz entweder mit Neuronalen Netzen oder mit dem K-Nearest Neighbor Algorithmus eine potentielle Lösung für das logistische Optimierungsproblem des Bahnnetzes entwerfen.
Und nachdem beide Grobansätze mit ihren einzelnen Methoden, Algorithmen oder Strategien genauer betrachtet wurden, galt es, den zu wählen, der die optimalste Lösung in Bezug auf die Aufgabenstellung des InformatiCups bieten könnte.

1.6 Auswertung der Lösungsansätzen

Um einen passenden aus den vier vorgeschlagenen Ansätzen zu wählen, haben wir uns mit diesen vier nochmals genauer auseinandergesetzt. Der Dijkstra Algorithmus ist ein sogenannter Greedy Algorithmus und hilft generell dabei, den kürzesten, kostengünstigsten Weg zwischen Nodes in einem Graphen mithilfe von Kantengewichten zu berechnen. Dabei könnte man im Fall des Bahnnetzes die Bahnhöfe als Nodes im Graphen darstellen und die Strecken dazwischen mit Gewichten versehen. Durch den Ablauf dieses Algorithmus könnte dann innerhalb mehreren Iterationen die kostengünstigste Strecke für die Züge bestimmt werden.
Der Bellman-Ford-Algorithmus ist dem Dijkstra Algorithmus prinzipiell sehr ähnlich. Auch er ist dafür ausgelegt, ausgehen von einem Startknoten den kürzesten Weg zu allen anderen Knoten in einem Graphen zu finden. Bellman-Ford kann jedoch auch mit einem negativem Kantengewicht umgehen. Auch hier könnte man das Schienennetzwerk wie beim Djikstra Algorithmus in einem Graphen darstellen und damit die kurzmöglichsten Fahrten für die einzelnen Züge und somit einen optimalen Fahrplan berechnen.
Beide relativ simpel funktionierende, regelbasierte Algorithmen könnten eine Option für die Lösung des InformatiCup Problems darstellen. Jedoch aufgrund unseres Studiengangs mit der Fachrichtung Data Science waren wir von Anfang an motiviert, eher einen Ansatz Richtung maschinellen Lernens zu wählen. Auch wenn die regelbasierten Ansätze Vorteile und gute Möglichkeiten für die Entwicklung einer Lösung bieten, haben wir den automatisierten Ansatz gewählt.
Wie bereits schon erwähnt wäre der k Nearest Neighbor (kNN) auch eine Alternative, mit der man einen optimalen Fahrplan für das Bahnnetz entwickeln könnte. Der kNN Algorithmus ist ein Klassifikationsverfahren, das zum Bereich des Supervised Learnings des maschinellen Lernens gehört. Dabei geht es darum, ein bestimmtes „K“ zu definieren und dann gemäß einer berechneten Distanz die nächsten Nachbarn von k zu wählen. Im Fall der Erstellung eines Zugfahrplans könnte man damit die zeitbezogen schnellstmöglichen Strecken für die einzelnen Züge im Bahnnetz auswählen. KNN an sich ist ein relativ simpler Algorithmus, der einfach zu verstehen und programmieren

ist. Er bietet eine einfache Handhabung von fehlenden Werten, sowie die Beschränkung der Abstandsberechnung auf einen Unterraum und die asymptotische Fehlklassifikationsrate ist nach oben hin durch den doppelten Bayes-Fehler begrenzt. Jedoch auf der anderen Seite wird kNN stark durch die lokal vorgegebene Struktur beeinflusst und ist gegenüber irrelevanten Merkmalen sehr empfindlich, was den Algorithmus insgesamt unstabil macht. Auch die Wahl des optimalen K stellt meist eine Herausforderung dar und in großen Dimensionen kann der „nächste Nachbar“ auch sehr weit entfernt sein, wodurch der Definition „am nächsten“ wenig Bedeutung zu kommt. Des Weiteren wird der kNN Algorithmus als „Lazy-Learner“ bezeichnet, das bedeutet, dass er nicht aus einem Trainingsdatensatz lernt, sondern den Datensatz speichert und immer eine bestimmte Action zur Klassifizierung der vorliegenden Daten ausführt. Aus diesen Gründen haben wir uns dann gegen die Verwendung des kNN Algorithmus entschieden.

Währenddessen haben wir uns auch mit der Thematik der Neuronalen Netzen (NN[1]) beschäftigt. Diese gehören zum Unsupervised Learning Bereich des Machine Learnings und stellt eine Reihe von Algorithmen dar, die versuchen, zugrunde liegende Beziehung in einer Reihen von Daten durch einen Lernprozess zu erkennen. NNs zeichnen sich dadurch aus, dass sie anpassungsfähig sind, das heißt sie modifizieren sich selbst, während sie aus dem anfänglichen Training lernen, und nachfolgende Durchläufe liefern mehr Informationen über die Welt. In Verbindung mit Reinforcement Learning, welches ermöglicht, ohne Ausgansdaten und menschliches Vorwissen, Lösungen zu komplexen zu finden, könnten NNs einen vielversprechenden und effektiven Ansatz für die Lösung unseres Ausgansproblems darstellen. Durch ihre große Rechenpower könnten NNs bei der Entwicklung des Fahrplans auch die unterschiedlichen Details, beziehungsweise Eigenschaften, der einzelnen Elemente berücksichtigen und selbstständig durch das Trial-and-Error-Vorgehen des Reinforcement Learnings die optimalen Wege für die einzelnen Züge im Bahnnetz bestimmen. Gerade aufgrund der großen Rechenpower und der Flexibilität der NNs haben wir uns schlussendlich dazu entschieden, diesen Ansatz zu wählen.

[1] Neruonale Netze

2 Theoretischer Lösungsansatz

In diesem Kapitel werden die unterschiedlichen theoretischen Grundlagen beschrieben, auf welchen unsere Lösung basiert. Dabei werden grundlegend die Begriffe Reinforcement Learning und Graph Neural Networks erklärt. Während bei Ersterem noch genauer auf die Proximal Policy Optimization und die Monte Carlo Tree Search eingegangen wird, werden nach den Graph Neural Networks noch die Struktur, sowie die Eigenschaften, Funktionen und die Herangehensweise und ihre Vorteile unseres Lösungsansatzes erläutert.

2.1 Reinforcement Learning

Dieses Kapitel stellt einen groben Einstieg in das Thema Reinforcement Learning dar und wird später anhand der Proximal Policy Optimization und der Monte Carlo Tree Search in Bezug auf das Q-Learning konkretisiert.
Das Reinforcement Learning, zu Deutsch Bestärkendes oder Verstärkendes Lernen, ist ein Gebiet des Machine Learnings. Allgemein lässt sich das maschinelle Lernen in drei Untergebiete gliedern: dem Supervised Learning, dem Unsupervised Learning und dem Reinforcement Learning.

Im Gegensatz zum Supervised und Unsupervised Learning braucht das Reinforcement Learning vorab keine Daten. Genauer gesagt steht das Reinforcement Learning für eine ganze Reihe an Einzelmethoden, die darauf basieren, erwünschtes Verhalten eines bestimmten Software-Agenten zu belohnen und/oder unerwünschtes zu bestrafen.

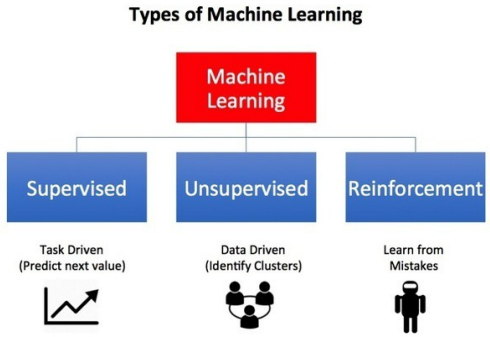


Abbildung 3: Die Bereiche des Machine Learnings
(Bhatt, 2018)

Dabei steht der Agent bei Reinforcement Learning sozusagen vor einer spielähnlichen Situation.
Im Rahmen einer unsicheren, potenziell komplexen, aber auch interaktiven Umgebung wird der Agent durch das Vorgehen des Trial-and-Errors und durch den Vorgang des Belohnens, beziehungsweise des Bestrafens, als Feedback auf seine Aktionen dazu befähigt, eine Strategie für das Erreichen eines Ziels zu erlernen.
Generell lassen sich mithilfe von Reinforcement Learning Klassifikation-, Entscheidungs- und Regelungsprobleme lösen, weshalb es in den unterschiedlichsten Bereichen angewandt wird. Beispiele hierfür sind unter anderem die individuell zugeschnittenen Produktvorschläge für Kunden basierend auf ihrem Einkaufsverhalten in Webshops, sowie verschiedene Maschinen zum Zusammenfassen von Texten oder auch das in der Wirtschaft sehr bedeutsame Topic Autonomes Fahren. Außerdem gelang es AlphaGo Zero mit Hilfe von Reinforcement Learning als erstes Computerprogramm den Weltmeister in dem chinesischen Spiel Go zu schlagen.
Um ein typisches Reinforcement Problem darzustellen, werden ein paar grundlegende Begriffe benötigt. Dazu gehört beispielsweise die Environment, welche die physische Welt darstellt, in welcher der Software-Agent arbeitet. Des Weiteren befindet sich der Agent immer in einem State, also der derzeitige Zustand oder die derzeitige Situation des Agenten. Auf seine Aktionen erhält der Agent dann auch immer ein Feedback, der sogenannte Reward – bei positivem Feedback – oder den Penalty – bei negativem Feedback. Die Policy ist die Methode die den Zustand des Agenten seinen einzelnen Aktionen zuordnet und der Value ist der zukünftige Reward, den der Agent durch das

Ausführen einer Aktion in einem bestimmten Zustand erhalten würde (Bhatt, 2018). Generell kann man die grundlegende Herangehensweise des Reinforcement Learnings wie folgt beschreiben: Als erstes wird die vorliegende Situation durch den Software-Agenten gründlich beobachtet, bevor er mithilfe der Policy eine mögliche Aktion aussucht und anschließend ausführt. Für diese ausgeführte Aktion erhält der Agent dann entweder einen Reward oder einen Penalty. Daraus kann er sich dann einen neuen Lernschritt ableiten und die Policy dementsprechend anpassen kann. Der Agent spielt alle möglichen Aktionen oder Situationen in seiner Environment wiederholt durch und lernt aus dann eine Strategie, bis eine optimale Policy gefunden ist.

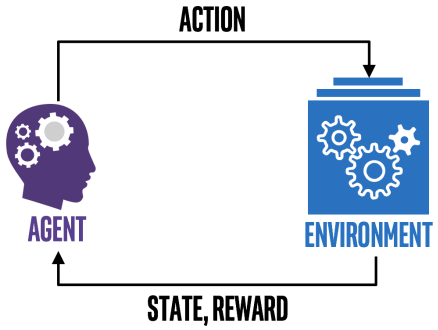


Abbildung 4: Reinforcement Learning
(Lee, 2019)

Vorteile des des Reinforcement Learnings gegenüber anderen maschinellen Methoden sind unter anderem, dass allgemein keine Ausgangsdaten oder menschliches Vorwissen benötigt wird, um Lösungen zu komplexen Problemen zu finden. Außerdem ist auch die aufwendige Erhebung und Bearbeitung von Trainingsmethoden hierbei nicht notwendig und die Lernmethode des Reinforcement Learnings kann für beliebige intellektuelle Aufgaben eingesetzt werden(Landwehr, 2020) .
Auf der anderen Seite ist jedoch auch zu sehen, dass bei Reinforcement Learning die größte Herausforderung darin besteht, die Simulationsumgebung vorzubereiten, welche wiederum stark abhängig von der vorliegenden Problem- oder Aufgabenstellung ist. Auch liegt eine weitere Herausforderung im Skalieren und Optimieren des Neural Networks, welches den Software-Agenten steuert. Neben System des Rewards oder des Penaltys gibt es keine andere Möglichkeit, um mit dem Netzwerk zu kommunizieren,

was wiederum zu folgeschweren Fehlern führen kann (Osinski und Budek, 2018).
Im Falle der Aufgabenstellung des InformatiCups 2022 geht es also darum, dass der Agent durch das Trial-and-Error-Verfahren die für ihn schnellstmögliche Strecke wählt und somit dann die Gesamtverspätung der Fahrgäste so gering wie möglich hält.

2.1.1 PPO - Proximal Policy Optimization

Die Proximal Policy Optimization, kurz PPO[1], ist eine Policy-Gradienten-Methode für Reinforcement Learning. Policy-Gradient-Methoden sind eine Art von Reinforcement-Learning-Techniken, die auf der Optimierung parametrisierter Strategien in Bezug auf die erwartete Rendite (der langfristige kumulative Reward) durch Gradientenabstieg beruhen. Im Gegensatz zu den traditionellen Reinforcement-Learning-Ansätzen sind sie nicht ganz so anfällig für Probleme, wie zum Beispiel dem Problem der Unlösbarkeit, das sich aus unsicheren Zustandsinformationen ergibt.
Bei PPO handelt es sich um eine Optimierung erster Ordnung, die ihre eigene Implementierung vereinfacht. Sie definiert das Wahrscheinlichkeitsverhältnis zwischen der neuen Policy und der alten Policy, was als r(0) bezeichnet wird. Dabei verfolgt PPO dann folgenden Ansatz: Anstatt eine harte Beschränkung aufzuerlegen, wird die Beschränkung als Strafe in der Zielfunktion formalisiert. Indem die Einschränkung nicht um jeden Preis vermieden wird, kann ein Optimierer erster Ordnung wie die Gradientenabstiegsmethode verwendet werden, um das Ziel zu optimieren. Auch wenn diese Einschränkung gelegentlich verletzt wird, ist der Schaden weitaus geringer und die Berechnung viel einfacher als bei den restlichen Policy-Gradient-Methoden, wie zum Beispiel der Trust Region Policy Optimization TRPO[2] (Austin, 2020).

[1] Proximal Policy Optimization
[2] Trust Region Policy Optimization

2.1.2 MCTS – Monte Carlo Tree Search

In der Computer Science gehört die Monte Carlo Tree Search, kurz MCTS[1] oder auch zu Deutsch Monte-Carlo-Baumsuche, zur Familie der Baumsuchalgorithmen.

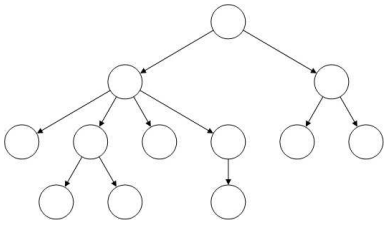


Abbildung 5: Der Baumsuchalgorithmus
(Stork, 2010)

Bei Baumsuchalgorithmen (Beispiel: s. Abbildung 5) handelt es sich um eine Methode, die verwendet wird, um innerhalb einer Runde eines Spiels jeden möglichen nächsten Spielzug zu suchen. Die verschiedenen möglichen Optionen an Spielzügen können dabei mithilfe einer Baumdarstellung visualisiert werden. Je mehr Runden innerhalb des Spiels, desto größer ist der Baum, dies ist nicht sehr optimal. Denn das Erzwingen eines exponentiell wachsenden Baums, um die endgültige Lösung für ein Problem zu finden, indem alle möglichen Optionen, das heißt in der Visualisierung alle untergeordneten Knoten, berücksichtigt werden, erfordert sehr viel Rechenleistung. Und viel benötigte Rechenleistung bedeutet wiederum einen sehr langen Findungsprozess der Lösung. Der ganze Entscheidungsprozess kann jedoch beschleunigt werden, indem man Richtlinien aufstellt, das heißt man rechnet einigen Knoten mehr Bedeutung zu als anderen. Dadurch wird ermöglicht, dass die untergeordneten Knoten der „bedeutsameren Knoten“ als erstes durchsucht werden, um eine passende Lösung zu finden(Sharma, 2018).
Und genau an dieser Stelle kommt die Monte Carlo Tree Search ins Spiel. Im Wesentlichen achtet MCTS mehr auf Knoten, welchen mehr Bedeutung zukommt, beziehungsweise welche vielversprechender sind. Dadurch wird vermieden, dass alle Möglichkeiten geprüft werden müssen(Wang, 2021).
Allgemein besteht MCTS aus vier grundlegenden Schritten, die immer wieder – idealerweise unendlich, was jedoch in der Praxis aufgrund von Limitationen von beispielsweise Ressourcen nicht möglich ist – wiederholt werden, um aus einem Set von Zügen den

[1] Monte Carlo Tree Search

besten als finale Lösung zu wählen. Die Schritte bestehen dabei aus: Auswahl (Selection), Erweiterung (Expansion), Simulation und Sicherung (Backpropagation) (s. Abbildung 6).

Bei der Selection geht es darum, einen Knoten im Baum auszuwählen, der die höchste Gewinnchance hat, damit die gesuchte Lösung am schnellsten gegenüber allen anderen Optionen im Baum erreicht wird.

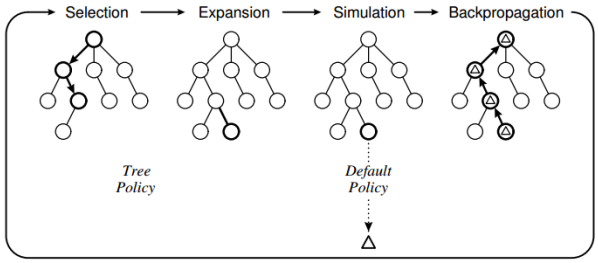


Abbildung 6: Monte Carlo Tree Search (Lang, n.d.)

Danach werden bei der Expansion die Optionen im Spiel erweitert und viele untergeordnete Knoten erstellt. Diese weiteren Unterknoten sind die zukünftigen möglichen Spielzüge. Die vorerst nicht weiter ausgebauten Knoten heißen Blätter. In einem weiteren Schritt geht es bei der Expansion darum, den richtigen Knoten von allen herauszufinden, das heißt den Pfad zu wählen, der am besten performt und zur richtigen Lösung entlang der Baumstruktur führt.
Jedoch wie kann der eine richtige Knoten gefunden werden?
Dazu wird Reinforcement Learning verwendet, um zufällige Entscheidungen im Spiel weiter unten von jedem weiteren Knoten zu treffen. Dabei wird jedem untergeordneten Knoten einen Reward gegeben – indem berechnet wird, wie nahe die Ausgabe ihrer zufälligen Entscheidung an der endgültigen Ausgabe lag, die wir benötigen, um das Spiel zu gewinnen. Allgemein wird die Simulation für jeden untergeordneten Knoten durchgeführt, gefolgt von ihren individuellen Belohnungen.
Schlussendlich müssen die Gesamtpunktzahlen der übergeordneten Knoten aufgrund der neuen Knoten und ihrer positiven oder negativen Bewertungen in der Umgebung aktualisiert werden. Die geschieht, indem der Baum einzeln nach oben zurückgegangen wird. Die neuen aktualisierten Bewertungen ändern den Zustand des Baums und können auch neue zukünftige Knoten des Auswahlprozesses ändern.
Nachdem alle Knoten in der Simulation-Phase geupdatet wurden, fängt die Schleife wieder mit dem Auswählen des besten Knoten im Baum an und fährt anschließend mit den folgenden Schritten fort, bis ein neuer Knoten weiter unten die gewollte finale

Lösung darstellt.
Allgemein lässt sich feststellen, dass MCTS gegenüber der Brute-Force, aus Millionen möglicher Wege den richtigen zu finden, eine wirtschaftlichere und effektivere Alternative darstellt (Sharma, 2018).
In unserem konkreten Fall des Entwerfens eines optimalen Fahrplans für ein gegebenes Schienennetzwerk, wird die Monte Carlo Tree Search dazu verwendet, um eine Strecke für jeden Zug im Netzwerk auszuwählen. Dabei soll es sich dann um genau die Strecke handeln, die es dem speziellen Zug - oder im Gesamten gesehen allen Zügen - ermöglicht so schnell und daraus folgend mit so wenig Verspätung wie möglich an seinem Zielbahnhof anzukommen.

2.2 Graph Neural Networks

In diesem Abschnitt werden wir uns mit dem anderen Teil unseres Algorithmus auseinandersetzen, dem Graph Neural Network (GNN[1]). Dabei wollen wir zuerst auf die Graphen im Allgemeinen eingehen und im Anschluss die Funktionsweise solcher GNNs näher betrachten.

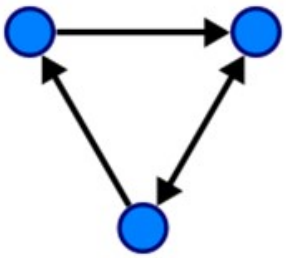


Abbildung 7: Directed Graphs (Scarselli et al., 2009)

Ein Graph besteht aus mehreren Punkten (Knoten oder Nodes), die miteinander in Verbindung stehen (Ecken oder Edges) und kann folgenderweise definiert werden G = (V, E), wobei V die Menge der Knoten und E die Kanten zwischen ihnen sind. Graphen befinden sich überall um uns herum und bekommen immer eine größere Bedeutung im Bereich des maschinellen Lernens. Der Grund dafür liegt darin, dass sich vieles

[1] Graph Neural Networks

in einem Graph abbilden lässt. So können zum Beispiel Bilder als Graph dargestellt werden, in dem jeder Pixel als Graphenknoten (Nodes) mit einer Kante (Edge) mit seinem Nachbar verbunden ist. Moleküle lassen sich ideal als Graphen abbilden. So würden die Atome die Nodes darstellen und die Verbindungen zwischen den Atomen die Edges. Zusätzlich kann jeder Verbindung Features hinzugefügt werden, wie zum Beispiel der Atomtyp, die Ladung oder der Bindungstyp an den Kanten. Sobald das Molekül dargestellt ist, kann man anhand dieses Graphen interessante Fragen beantworten (Scarselli et al., 2009).
In unserem Fall werden wir das Schienennetz als Graphen abbilden. Die Bahnhöfe bilden die Nodes ab und die Verbindungen zwischen den Bahnhöfen die Edges. Die Kapazität und die Länge der Strecke, sowie die Anzahl der Passagiere werden als Feature übergeben.
Ein Problem im Bereich des Deep Learnings ist es reale Gegebenheit, in all ihrer Komplexität, richtig darzustellen aus diesem Grund sind Graphen bestens dafür geeignet. In einem GNN sammeln die Nodes Informationen von ihren Nachbarn, welche über Edges miteinander verbunden sind. Auf diese Weise kann das GNN lernen: Informationen werden weitergegeben und in die Eigenschaften des jeweiligen Knotens aufgenommen.

2.3 Das Ziel und die Strategie der Lösung

Nachdem in den vorherigen Kapiteln das grundlegende theoretische Wissen vermittelt wurde, soll dieses Kapitel beschreiben, inwiefern diese theoretischen Methoden zusammenwirken, um unseren Lösungsansatz für die Aufgabenstellung des InformatiCups zu stellen.
Generell handelt es sich bei unserem Lösungsansatz um eine Graph-Neural-Network-unterstützte Monte Carlo Tree Search.

3 Softwarearchitektur

Dieses Kapitel dreht sich um die Softwarearchitektur, das heißt es wird unter anderem auf Fragen wie: Wie ist der Programmcode genau aufgebaut? Welche Packages wurden verwendet? Und wie liefert das Programm schlussendlich Ergebnisse? eingegangen. Außerdem wird auch das Training unserer Reinforcement-Learning-Lösung abgehandelt.

3.1 Die Architektur

Bevor in diesem Kapitel auf den grundlegenden Aufbau unseres Programmcodes eingegangen wird, muss erwähnt werden in welcher Programmiersprache der Code geschrieben wurde. Dabei handelt es sich bei unserer Lösung um die Programmiersprache Python. Grund für die Entscheidung für Programmiersprache Python war, dass diese zweifellos die beste Wahl im Bereich des Machine Learnings darstellt. Außerdem ist sie leicht verständlich, was die Datenvalidierung schnell und praktisch fehlerfrei macht und durch den Zugriff auf ein weit entwickeltes Bibliotheks-Ökosystem können komplexe Aufgaben ohne umfangreiches Programmieren ausgeführt werden.

3.1.1 Die Packages

Für unsere Lösung für das Erstellen des optimalen Fahrplan eines vorgelegten Schienennetzwerk haben wir in der Programmierung verschiedene Packages verwendet.
Primär haben wir in unserer Lösung das Python-Package Torch eingebaut. Torch ist eine Open-Source-Bibliothek für maschinelles Lernen in Python, die hauptsächlich

für Deep-Learning-Anwendung mit GPU[1]s und CPU[2]s verwendet wird. Torch wurde hauptsächlich vom Facebook AI Research-Team entwickelt.
Des Weiteren haben wir das Package NetworkX verwendet. NetworkX ist ein Softwarepaket, das für die Erstellung, Bearbeitung und Untersuchung der Struktur, Dynamik und Funktionen komplexer Netzwerke eingesetzt wird. Dabei wird es genutzt, um große komplexe Netzwerke zu untersuchen, die in Form von Graphen mit Knoten und Kanten dargestellt werden. Mit NetworkX können komplexe Netzwerke geladen und gespeichert werden. Außerdem können können viele Arten von zufälligen und klassischen Netzwerken generiert werden, die Netzwerkstruktur analysieren, Netzwerkmodelle erstellen, neue Netzwerkalgorithmen entwerfen und Netzwerke zeichnen.
Auch haben wir PyTorch Geometric (PyG[3]) genutzt. PyG ist eine auf PyTorch aufbauende Bibliothek zum einfach Schreiben und Trainieren von GNNs für eine Vielzahl von Anwendungen im Zusammenhang mit strukturierten Daten. Es besteht aus verschiedenen Methoden für Deep Learning auf Graphen und anderen unregelmäßigen Strukturen, auch bekannt als geometrisches Deep Learning. Darüber hinaus setzt es sich aus einfach zu bedienenden Mini-Batch-Loadern für den Betrieb auf vielen kleinen und einzelnen riesigen Graphen, Multi-GPU-Unterstützung, verteiltem Graph-Lernen über Quiver, einer großen Anzahl gängiger Benchmark-Datensätze (basierend auf einfach zu erstellenden Schnittstellen eigene), den GraphGym-Experimentmanager und hilfreiche Transformationen, sowohl zum Lernen auf beliebigen Graphen als auch auf 3D-Netzen oder Punktwolken zusammen.

[1] Graphics Processing Unit
[2] Central Processing Unit
[3] PyTorch Geometric

4 Schlussbemerkung

Aufgrund von Kompatibilitäs-Problemen ist es uns schlussendlich leider nicht gelungen, in dem gegebenen Zeitraum, unsere Lösung so ausgefeilt zu vervollständigen, dass sie einen optimalen Fahrplan für ein gegebenes Schienennetzwerk liefert. Dennoch wollten wir unsere Lösung für den InformatiCup abgeben, da wir von dem Potential unseres theoretischen Ansatzes überzeugt sind.

Literaturverzeichnis

Austin, D. (2020), 'Playing optimal heads-up poker with reinforcement learning', https://towardsdatascience.com/playing-optimal-heads-up-poker-with-reinforcement-learning-5fefe62618

Bhatt, S. (2018), 'Reinforcement learning 101', https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.

Landwehr, J. (2020), 'Was ist reinforcement learning?', https://it-talents.de/it-wissen/reinforcement-learning/.

Lang, E. (n.d.), 'Hauptseminar: Skat-ki auf basis von monte carlo tree search', https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2016-hsem/lang/bericht.pdf.

Lee, D. (2019), 'Reinforcement learning, part 1: A brief introduction', https://miro.medium.com/max/2000/0*WC4l7u90TsKs_eXj.png.

Osinski, B. und Budek, K. (2018), 'What is reinforcement learning? the complete guide', https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. und Monfardini, G. (2009), 'The graph neural network model', *IEEE Transactions on Neural Networks* **20**(1), 61–80.

Sharma, S. (2018), 'Monte carlo tree search', https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa.

Stork, T. (2010), 'Grundwissen informatik - baum', http://www.tobias-stork.de/gymnasium/informatik/grundwissen/Baum.html.

Vashisth, S. (2021), 'The 4 trends that prevail on the gartner hype cycle for ai, 2021', https://www.cdotrends.com/story/15913/4-trends-prevail-gartner-hype-cycle-ai-2021.

Wang, B. (2021), 'Monte carlo tree search: An introduction', https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168.

Ehrenwörtliche Erklärung

Hiermit erklären wir ehrenwörtlich, dass wir diese vorliegende Theoretische Ausarbeitung mit dem Thema *InformatiCup 2022 - Bahn-Simulator* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.
Die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren haben wir an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet.
Des Weiteren haben wir die vorliegende Arbeit bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Ravensburg, 16. Januar 2022

Pascal Knoll, Tom Zehle und Luisa Ibele