Hochschule Düsseldorf
University of Applied Sciences
HSD

Learning to Cooperate: Reinforcement Learning in the Iterated Prisoner's Dilemma

Dokumentation

von Jonah Gräfe

Düsseldorf, 16.02.2025

Studiengang:	B.Sc. Data Science, AI und Intelligente Systeme
Modul:	Advances in Intelligent Systems
Dozent:	Prof. Dr. Dennis Müller

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Thema und Motivation

Das Iterative Gefangenendilemma (IPD) ist ein klassisches Problem der Spieltheorie, das die Herausforderungen von Kooperation und Eigennutz modelliert. Zwei Spieler müssen unabhängig voneinander entscheiden, ob sie kooperieren oder defektieren. Während Kooperation zu einem besseren kollektiven Ergebnis führt, ist aus individueller Sicht Defektion oft vorteilhafter – ein klassisches Dilemma. In der Realität tauchen ähnliche Dilemmata in vielen Bereichen auf, etwa in der Wirtschaft, der Politik und der Evolutionsbiologie. Ein zentrales Forschungsthema ist, ob und unter welchen Bedingungen Akteure langfristig kooperative Strategien entwickeln können, um dem Dilemma zu entkommen.

Mit der zunehmenden Bedeutung von künstlicher Intelligenz (KI) und Reinforcement Learning (RL) stellt sich die Frage, wie sich autonome Agenten in einem solchen Szenario verhalten. Können sie durch Lernen langfristig Kooperation aufbauen, oder werden sie egoistische Strategien bevorzugen? Ziel dieses Projekts ist es, RL-Agenten im IPD zu trainieren und zu analysieren, ob sie fähig sind, Strategien zu entwickeln, die über bloßen Eigennutz hinausgehen. Damit trägt diese Arbeit zur Diskussion über das Potenzial von RL in sozialen und spieltheoretischen Kontexten bei.

1.2 Ziele der Arbeit

Das Ziel dieser Arbeit ist es, zu untersuchen, wie sich RL-Agenten im IPD verhalten und ob sie in der Lage sind, kooperative Strategien zu entwickeln. Dabei stehen folgende zentrale Forschungsfragen im Fokus:

1. Können RL-Agenten lernen, langfristig zu kooperieren?
    - Entwickeln sie Strategien, die über reines Eigeninteresse hinausgehen?
2. Welche Strategien entstehen im Laufe des Lernprozesses?
    - Verhalten sich die Agenten wie bekannte spieltheoretische Strategien (z. B. "Tit-for-Tat" oder "Always Defect")?
    - Zeigen sich neuartige, unerwartete Verhaltensmuster?

Um diese Fragen zu beantworten, werden verschiedene RL-Algorithmen auf das IPD angewendet, die Trainingsverläufe analysiert und die resultierenden Strategien miteinander verglichen. Durch die gewonnenen Erkenntnisse soll ein tieferes Verständnis dafür geschaffen werden, wie lernende Agenten spieltheoretische Herausforderungen bewältigen und ob sie sich aus dem klassischen Dilemma befreien können.

2 Hintergrund und theoretischer Rahmen

2.1 Das Iterierter Gefangenendilemma

Das Gefangenendilemma ist eines der bekanntesten Probleme der Spieltheorie und beschreibt eine Situation, in der zwei Spieler unabhängig voneinander entscheiden müssen, ob sie kooperieren oder defektieren. Die Auszahlung für jeden Spieler hängt sowohl von der eigenen Entscheidung als auch von der des Gegenübers ab. Die klassische Auszahlungsmatrix sieht dabei folgendermaßen aus:

	Spieler B kooperiert	Spieler B defektiert
Spieler A kooperiert	(R, R) Belohnung für Kooperation	(S, T) A verliert, B gewinnt
Spieler A defektiert	(T, S) A gewinnt, B verliert	(P, P) Bestrafung für gegenseitige Defektion

Tabelle 2.1: Allgemeine Auszahlungsmatrix

Dabei gilt üblicherweise:

- T (Temptation) $>$ R (Reward) $>$ P (Punishment) $>$ S (Sucker's payoff)
- $2R > T + S$, sodass sich gegenseitige Kooperation langfristig mehr lohnen würde als wechselseitige Ausnutzung.

Im einmaligen Gefangenendilemma ist die dominante Strategie, zu defektieren, da dies in jedem individuellen Fall die höhere Auszahlung sichert – unabhängig von der Entscheidung des Gegenspielers. Dies führt jedoch zu einem sozial suboptimalen Ergebnis.

Im iterierten Gefangenendilemma (IDG) wird das Spiel jedoch mehrfach hintereinander gespielt, sodass frühere Entscheidungen zukünftige Interaktionen beeinflussen können. Dadurch eröffnen sich neue Möglichkeiten für kooperative Strategien, bei denen Agenten versuchen, durch wechselseitige Zusammenarbeit langfristig höhere Erträge zu erzielen. Bekannte Strategien aus der Spieltheorie für das IDG sind beispielsweise: "Tit-for-Tat"(Spiele das, was dein Gegner in der vorherigen Runde getan hat). Älways Defect"(Immer defektieren, um kurzfristig die höchste Auszahlung zu sichern). "Grim Trigger"(Kooperiere, aber falls der Gegner einmal defektiert, defektiere für immer).

Die zentrale Forschungsfrage im IPD lautet daher: Ist es möglich, langfristige Kooperation zu etablieren, oder führt Eigennutz immer zu gegenseitiger Defektion? Diese Fragestellung ist besonders relevant für Reinforcement Learning-Agenten, da sie ihre Strategie durch wiederholte Interaktion und Belohnungsmechanismen erlernen.

2.2 Reinforcement Learning

Reinforcement Learning (RL) ist ein Teilbereich des maschinellen Lernens, bei dem ein Agent durch Interaktion mit einer Umgebung eine optimale Strategie erlernt. Der Lernprozess basiert auf einem Belohnungssystem: Der Agent führt Aktionen aus, erhält daraufhin Belohnungen oder Bestrafungen und passt sein Verhalten entsprechend an. RL-Probleme werden typischerweise als Markov-Entscheidungsprozesse (MDP) modelliert, bestehend aus:

- Zustand (State, S): Die aktuelle Situation der Umgebung.
- Aktion (Action, A): Eine Entscheidung, die der Agent treffen kann.
- Belohnung (Reward, R): Eine Rückmeldung, die die Qualitäten der gewählten Aktion bewertet.
- Übergangsmodell ($P(s'|s,a)$): Wahrscheinlichkeiten, dass der Zustand s' nach einer Aktion a im Zustand s entsteht.
- Policy ($\pi(s)$): Die Strategie des Agenten zur Auswahl von Aktionen.

Das Ziel ist es, eine Optimale Policy π^* zu lernen, die die kumulierte zukünftige Belohnung maximiert. Dafür gibt es verschiedene Methoden, darunter Q-Learning und Deep Q-Learning.

2.2.1 Q-Learning

Q-Learning ist ein wertbasierter RL-Algorithmus, der darauf abzielt, die Q-Werte für jede Zustands-Aktions-Kombination zu erlernen. Der Q-Wert $Q(s,a)$ repräsentiert die erwartete zukünftige Belohnung, wenn der Agent in Zustand s Aktion a wählt und danach der optimalen Strategie folgt. Die Aktualisierung der Q-Werte erfolgt iterativ mit der Bellman-Gleichung:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{2.1}$$

Hierbei sind:

- α die Lernrate, die bestimmt, wie stark neue Informationen alte Werte überschreiben.
- γ der Diskontfaktor, der die Gewichtung zukünftiger Belohnungen bestimmt.
- r die unmittelbare Belohnung nach der Aktion a.

Da Q-Learning tabellarisch arbeitet, ist es nur für kleine Zustandsräume geeignet, da die Q-Tabelle bei vielen möglichen Zuständen und Aktionen schnell zu groß wird. In komplexeren Umgebungen ist daher eine neuronale Netzarchitektur notwendig - hier kommt Deep Q-Learning (DQN) ins Spiel.

2.2.2 Deep Q-Learning

Deep Q-Learning (DQN) erweitert Q-Learning durch den Einsatz eines neuronalen Netzwerks zur Approximation der Q-Werte, anstatt eine explizite Tabelle zu speichern. Das Netz nimmt den Zustand s als Eingabe und gibt geschätzte Q-Werte für alle möglichen Aktionen a aus und die Gewichte des Netzwerks werden durch Gradientenabstieg aktualisiert. Anstatt aber das Netzwerk nach jedem Ereigniss zu trainieren, werden normalerweise vergangene Erfahrungen (s, a, r, s') in einem Speicher abgelegt, um so die Q-Werte in einem großen Schritt zu verbessern. Dazu wird zufällig aus diesem Speicher ein *Batch* (ein Teil der Erfahrungen) ausgewählt und die Q-Werte des Netzwerks aktualisiert (*experience replay*). Zudem existiert ein Zielnetzwerk, das die Q-Werte der optimalen Strategie repräsentiert und eine Kopie des originalen Netzwerks ist, welches aber seltener oder geringfügiger aktualisiert wird. Dies verhindert zu starke Schwankungen in den Q-Werten und stabilisiert das Training.

DQN ermöglicht das Lernen in hochdimensionalen Zustandsräumen, die tabellarische Methoden überfordern würden.

3 Methodik

3.1 Aufbau des Experiments

Um zu untersuchen, wie Reinforcement-Learning-Agenten im Iterierten Gefangenendilemma (IPD) agieren, wurde eine experimentelle Python-Umgebung implementiert. Diese Umgebung ermöglicht es, Spiele zu simulieren, RL-Agenten zu trainieren, und diese dann zu evaluieren. Für all diese Zwecke wurde die Rundenanzahl pro Spiel auf $n = 100$ gesetzt und die folgende Auszahlungsmatrix verwendet:

	Spieler B kooperiert	Spieler B defektiert
Spieler A kooperiert	(3, 3)	(5, 0)
Spieler A defektiert	(0, 5)	(1, 1)

Tabelle 3.1: Auszahlungsmatrix

3.2 Agenten und Strategien

Für das Training und die Evaluation der RL-Agent wurden folgende Basis-Strategien implementiert:

Strategie	**Beschreibung**
RandomAgent()	Entscheidet zufällig
AlwaysCooperateAgent()	Kooperiert immer
AlwaysDefectAgent()	Verrät immer
ProvocativeAgent()	Verrät nach zweimaligem Kooperieren
TitForTatAgent()	Imitiert den Gegner
TitForTwoTatsAgent()	Verrät, wenn der Gegner zweimal verrät
TwoTitsForTatAgent()	Verrät zweimal, wenn der Gegner verrät
TitForTatOppositeAgent()	Imitiert den Gegner umgekehrt
SpitefulAgent()	Verrät immer, wenn der Gegner verrät
GenerousTitForTatAgent()	Imitiert den Gegner, vergibt aber in 10% der Fälle
AdaptiveAgent()	Verrät, wenn der Gegner in den letzten 10 Runden zu mehr als 50% verraten hat
PavlovAgent()	Verrät, wenn die Entscheidungen in der vorherigen Runde unterschiedlich waren

GradualAgent()	Verrät so oft, wie der Gegner verrät
WinStayLoseShiftAgent()	Ändert die Strategie, wenn die letzte Belohnung < 1 war
SoftMajorityAgent()	Verrät, wenn der Gegner in mehr als 50% der Fälle verrät
SuspiciousTitForTatAgent()	Imitiert den Gegner und verrät in der ersten Runde
SuspiciousAdaptiveAgent()	Verrät, wenn der Gegner in den letzten 10 Runden zu mehr als 50% verraten hat, und verrät in der ersten Runde
SuspiciousGenerousTitForTatAgent()	Imitiert den Gegner, vergibt aber in 10% der Fälle und verrät in der ersten Runde
SuspiciousGradualAgent()	Verrät so oft, wie der Gegner verrät, und verrät in der ersten Runde
SuspiciousPavlovAgent()	Verrät, wenn die Entscheidungen in der vorherigen Runde unterschiedlich waren, und verrät in der ersten Runde
SuspiciousSoftMajorityAgent()	Verrät, wenn der Gegner in mehr als 50% der Fälle verrät, und verrät in der ersten Runde
SuspiciousTitForTwoTatsAgent()	Verrät, wenn der Gegner zweimal verrät, und verrät in der ersten Runde
SuspiciousTwoTitsForTatAgent()	Verrät zweimal, wenn der Gegner verrät, und verrät in der ersten Runde
SuspiciousWinStayLoseShiftAgent()	Ändert die Strategie, wenn die letzte Belohnung < 1 war, und verrät in der ersten Runde

Tabelle 3.2: Basis-Strategien

Zusätzlich gibt es den *RandomStrategies()* Agenten, der jede Epsiode zufällig eine Strategie aus den Basis-Strategien (siehe Tab. 3.2) auswählt und so seine Entscheidungen trifft.

Der Q-Learning Agent nutzt eine 2×2 Q-Tabelle zur Approximation der Q-Werte:

	Q-Werte für Kooperation	Q-Werte für Defektion
Gegner kooperierte	Q(0, 0)	Q(0, 1)
Gegner defektierte	Q(1, 0)	Q(1, 1)

Tabelle 3.3: Q-Tabelle

Weil die Q-Tabelle nur den letzten Gegnerzug betrachtet, kann der Q-Learning Agent nur vier Strategien entwickeln:

1. Immer kooperieren (wie *AlwaysCooperateAgent*) mit

$$Q(0,0) > Q(0,1) \wedge Q(1,0) > Q(1,1)$$

2. Immer defektieren (wie *AlwaysDefectAgent*) mit

$$Q(0,0) < Q(0,1) \wedge Q(1,0) < Q(1,1)$$

3. Den Gegner imitieren (wie *TitForTatAgent*) mit

$$Q(0,0) > Q(0,1) \wedge Q(1,0) < Q(1,1)$$

4. Gegenteil der Imitation (wie *TitForTatOppositeAgent*) mit

$$Q(0,0) < Q(0,1) \wedge Q(1,0) > Q(1,1)$$

Hierbei steuert die Lernrate $\alpha = 0.001$, wie stark neue Informationen in die Q-Tabelle einfließen, während der Discount-Faktor $\gamma = 0.95$ zukünftige Belohnungen mit einbezieht. Der Agent nutzt eine ϵ-greedy Strategie, bei der mit Wahrscheinlichkeit ϵ eine zufällige Aktion gewählt wird, um Exploration zu ermöglichen. ϵ startet bei 1.0 und wird mit 0.9995 pro Runde reduziert, bis ein Minimum von 0.00001 erreicht ist. Zur Entscheidungsfindung nutzt er entweder zufällige Exploration oder wählt die Aktion mit dem höchsten Q-Wert basierend auf der letzten Gegneraktion.

Der Deep Q-Learning Agent nutzt ein neuronales Netzwerk zur Approximation der Q-Werte und basiert auf einem Feedforward-Netzwerk mit drei voll verbundenen Schichten: Eine Eingabeschicht mit 64 Neuronen, eine versteckte Schicht mit 32 Neuronen und eine Ausgabeschicht mit 2 Neuronen, die die Q-Werte für die beiden möglichen Aktionen (Kooperation oder Defektion) liefert. ReLU-Aktivierungen sorgen für nicht-lineare Modellierung, während Xavier-Initialisierung eine stabile Gewichtsverteilung gewährleistet. Diese sorgt dafür, dass die Gewichte so initialisiert werden, dass der Informationsfluss durch das Netzwerk stabil bleibt. Die Gewichte W werden so initialisiert, dass

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{6}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right) \tag{3.1}$$

mit n_{in} und n_{out} als Anzahl der Neuronen der vorherigen und der aktuellen Schicht. Die Varianz der Gewichte bleibt also über alle Schichten hinweg konstant. Das Netz bekommt die letzten zehn Runden, also einen 20-dimensionalen Zustandsvektor, als Eingabe. Der Agent nutzt eine Replay-Memory (max. 20.000 Einträge) zur stabileren Lernprozessgestaltung und ein ϵ-greedy Explorationsverfahren, bei dem die Wahrscheinlichkeit für zufällige Aktionen mit zunehmendem Training abnimmt (ϵ sinkt von 1.0 auf 0.00001 mit einem Faktor von 0.9995). Das Lernen erfolgt über MSE-Loss und Adam-Optimierung mit einer Lernrate von 0.001. Während des Trainings werden zufällige 5 Batches aus

der Replay-Memory gezogen und die Q-Werte aktualisiert, wobei der jeweilige optimale Q-Wert (Ausgabe des Zielnetzwerks) mit $\gamma = 0.95$ diskontiert wird.

Die Belohnunsfunktion beider RL-Agenten ist die vorher beschriebene Auszahlungsmatrix (siehe Tab. 3.1). Beide RL-Agenten wurden in 10000 Episoden gegen eine zufällige aber feste Reihenfolge von Basis-Agenten trainiert.

3.3 Evaluierungsmethodik

Die Evaluierungsmethodik basiert auf einem Turnierformat, in dem jeder Agent gegen jeden Basis-Agenten 100 Spiele absolvieren muss. Jeder Agent spielt also $24 * 100 = 2400$ Spiele (es gibt 24 Basis-Agenten) und $2400 * 100 = 240000$ Runden. Dieses Format stellt sicher, dass für alle Agenten die selben Voraussetzungen herrschen und statistisch signifante Ergebnisse erzielt werden.

4 Ergebnisse

4.1 Daten und Beobachtungen

Über die $T = 10000$ Trainingsepisoden wurden folgende durchschnittliche Belohnungen pro Epsiode berechnet:

	$\frac{\sum r}{T}$
QLearningAgent($\alpha = 0.1$, $\gamma = 0.95$)	265
DeepQLearningAgent($\alpha = 0.001$, $\gamma = 0.95$)	287
RandomStrategies()	233

Tabelle 4.1: Trainigs-Ergebnisse

Es bildet sich also schon während des Trainings ab, dass der QLearning-Agent und der Deep QLearning-Agent langfristig bessere Entscheidungen treffen, als der *RandomStrategies()* Agent. Es lohnt sich also weniger jedes Spiel eine zufällige etablierte Strategie zu wählen. Nach dem Training wurden alle Agenten und Strategien in einem Tunier getestet:

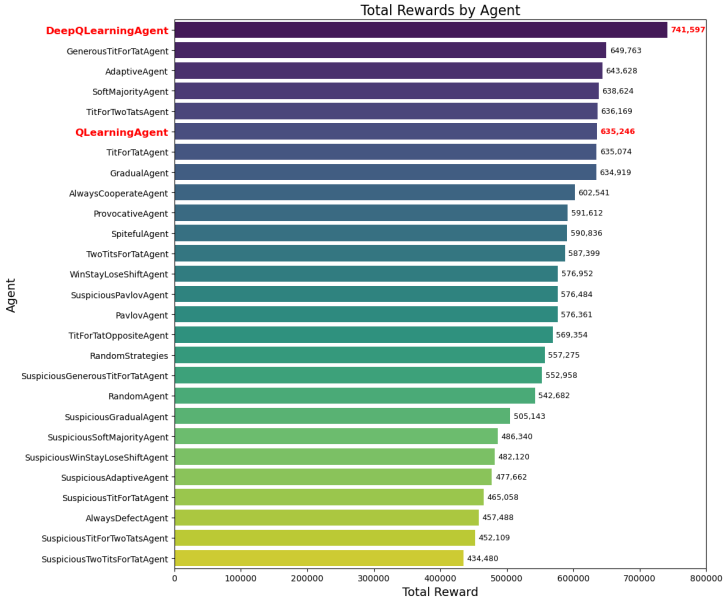

Abbildung 4.1: Gesamtbelohnungen der Agenten im Turnier

Bevor die Ergebnisse der RL-Agenten genauer analysiert werden, können einige Beobachtungen zusammengetragen werden:

- *Suspicious* Agenten schneiden insgesamt schlechter ab.
- Die beste Basis-Strategie ist *GenerousTitForTat*.
- Immer zu kooperieren ist besser als immer zu defektieren.

Diese Beobachtungen decken sich auch mit den Entdeckungen, die Robert Axelrod in seinem Buch *The Evolution of Cooperation*[1] gemacht hat.

4.1.1 QLearning-Ergebnisse

Aus dem Training ging folgende Q-Tabelle hervor:

	Q-Werte für Kooperation	Q-Werte für Defektion
Gegner kooperierte	52.7929238	2.61803697
Gegner defektierte	0.9977664	53.59111475

Tabelle 4.2: Q-Tabelle nach dem Training

Da $Q(0,0) > Q(0,1) \wedge Q(1,0) < Q(1,1)$ gilt, konvergierte der *QLearningAgent* offentsichtlich in Richtung des *TitForTatAgent*'s. Dies spiegelt sich auch in den Tunierergebnissen (siehe Abb. 4.1) wieder: Dort reiht sich der QLearning-Agent auch dort ein. Durch unterschiedliche Entscheidungen von Agenten, die auf Wahrscheinlichkeiten beruhen (*RandomAgent(), GenerousTitForTatAgent()*) gibt es minimale Abweichungen zwischen den Agenten. Wenn man diese Ergebnisse mit denen der anderen drei Strategien, die der *QLearningAgent()* entwickeln konnte vergleicht, fällt auf, dass die TitForTat-Strategie langfristig die besten Ergebnisse erzielt. Es gilt: TitForTatAgent() > AlwaysCooperateAgent() > TitForTatOppositeAgent() > AlwaysDefectAgent(). Die Verteilung von Kooperation und Defektion des *QLearningAgent*'s ist in Abb. 4.2 dargestellt:

[1] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

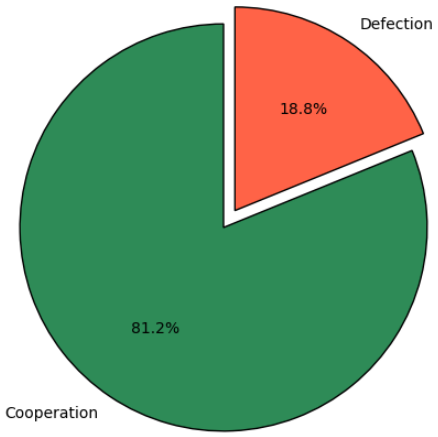

Abbildung 4.2: Verteilung von Kooperation und Defektion des *QLearningAgent*'s

4.1.2 Deep QLearning-Ergebnisse

Der *DeepQLearningAgent* geht als Sieger des Tuniers hervor mit durschnittlich $741597/2400 \approx 309$ Belohnungen pro Spiel. Die folgende Grafik zeigt die Verteilung von Kooperation und Defektion über alle Spiele im Turnier:

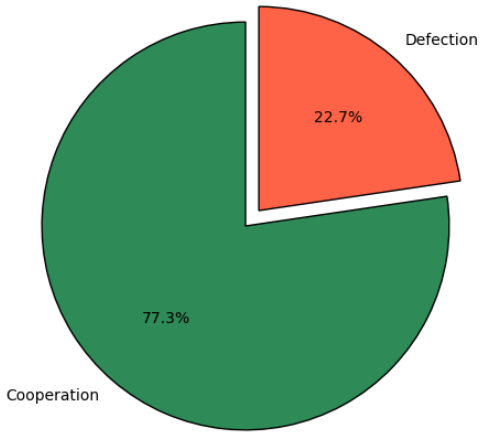

Abbildung 4.3: Verteilung von Kooperation und Defektion des *DeepQLearningAgent*'s

Damit defektiert der *DeepQLearningAgent* öfter als der *QLearningAgent*, aber auch hier zeigt sich wieder: Es lohnt sich häufig zu kooperieren.

Um besser zu verstehen wie der *DeepQLearningAgent* agiert, kann man sich anschauen, wie er seine Strategie je nach Gegner anpasst. Gegen die meisten Strategien setzt er auf langfristige Kooperation und defektiert nur sehr selten, es gibt aber auch einige Ausnahmen: Gegen den *SoftMajorityAgent()* wäre die beste Strategie abwechselnd zu koopieren und zu defektieren. Der *DeepQLearningAgent* erkennt dies und defektiert in fast 50% der Runden. Gegen den *ProvocativeAgent()* defektiert er fast immer, genauso wie gegen den *TitForTatOppositeAgent()*, was in beiden fällen auch sinnvolle Strategien sind. Gegen den *TitForTwoTatsAgent()* defektiert er ebenfalls in ungefähr 50% der Runden. Teilweise "verwechselt" der *DeepQLearningAgent* seine Gegner und defektiert z.B. nur alle 2 Runden gegen den *AlwaysCooperateAgent()*. Gegen den *AlwaysDefectAgent()* und den *SpitefulAgent()* hat er jedoch Probleme und kooperiert häufig.

Insgesamt muss man anerkennen, dass ein "perfektes" Ergebnis nicht möglich ist. Im Verhalten des *DeepQLearningAgent* erkennt man ebenfalls, dass er zuerst "testet", gegen welchen Gegner er spielt. Das ist notwenig, um die Strategie anzupassen, führt aber auch zum Verlust von Punkten. Beim *SpitefulAgent()* z.B. ist es aber schon zu spät, wenn man erkennt, dass dieser der Gegner ist.

Durch diese Mischung aus langfristiger Kooperation und Ausnutzen von Schwachstellen in der gegnerischen Strategie ist der *DeepQLearningAgent* in der Lage langfristig zu gewinnen und jede andere Strategie zu übertreffen.

5 Diskussion

5.1 Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass lernende Agenten wie der *QLearningAgent* und insbesondere der *DeepQLearningAgent* langfristig erfolgreicher sind als einfache, vorab definierte Strategien. Diese Beobachtung bestätigt die Annahme, dass eine adaptive Strategie vorteilhafter ist als eine statische. Dennoch gibt es verschiedene Aspekte, die näher betrachtet werden sollten. Wie in 3.2 erklärt konnte der *QLearningAgent* nur in vier Richtungen konvergieren. Das bedeutet natürlich auch, dass er nie besser als die beste Basis-Strategie sein kann. Da er sich aber aber für die beste für ihn mögliche Strategie "entschieden" hat, zeigt er das Potential langfristig gute Entscheidungen zu treffen und überlegene Strategien zu entwickeln. Die höhere Belohnung des *DeepQLearningAgent* ist insbesondere auf seine Fähigkeit zurückzuführen, gegnerische Muster zu identifizieren und entsprechend zu handeln. Allerdings wurde beobachtet, dass er in einigen Situationen "Fehlklassifikationen" vornimmt und beispielsweise gegen den *AlwaysDefectAgent()* fälschlicherweise oft kooperiert. Dies deutet darauf hin, dass sein Entscheidungsprozess nicht immer optimal ist und dass es Situationen gibt, in denen er durch z.B. einen zu starken Drang nach Kooperation Punkte verliert.

Ein zentrales Ergebnis ist, dass langfristige Kooperation gegenüber reiner Defektion überlegen ist. Dies spiegelt die grundlegenden Erkenntnisse aus der Spieltheorie wider, wonach Vertrauen, Zusammenarbeit, Vergebung und Großzügigkeit langfristig zu höheren Gewinnen führen können. Allerdings zeigt der *DeepQLearningAgent*, dass es sich lohnen kann, Schwächen in gegnerischen Strategien gezielt auszunutzen. Diese Mischung aus Kooperation und Opportunismus scheint der Schlüssel zu maximalem Erfolg zu sein.

5.2 Limitationen

Trotz der vielversprechenden Ergebnisse gibt es einige Limitationen:

- Die Anzahl der getesteten Spiele und Strategien ist begrenzt. Eine breitere Auswahl an Strategien könnte neue Herausforderungen für die Agenten darstellen.
- Die RL-Modelle basieren auf den spezifischen Hyperparametern α, γ und ϵ. Andere Parametrisierungen oder alternative Architekturen, insbesondere im Deep-Q-Learning, könnten zu unterschiedlichen Ergebnissen führen.
- Der Einfluss von mehrstufigen Gegneranalysen oder Gedächtnisstrategien wurde nicht untersucht. Ein RL-Agent mit Langzeitgedächtnis (z.B. LSTMs) könnte noch besser auf bestimmte Gegner reagieren.

6 Fazit

6.1 Zusammenfassung der Ergebnisse

Zusammenfassend zeigen die Ergebnisse, dass Reinforcement Learning eine effektive Methode ist, um erfolgreiche Strategien für wiederholte soziale Dilemmata zu entwickeln. Während der *QLearningAgent* bereits eine solide Strategie findet, hebt sich der *DeepQLearningAgent* durch seine anpassungsfähige und opportunistische Spielweise hervor. Dies unterstreicht die Stärke von Deep Learning in strategischen Entscheidungsszenarien. Trotz einzelner Schwächen zeigt sich, dass lernfähige Agenten klassische, statische Strategien langfristig übertreffen können.

Literatur

Axelrod, Robert. *The Evolution of Cooperation.* Basic Books, 1984.