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

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?
    - Gibt es bestimmte Bedingungen, unter denen Kooperation wahrscheinlicher wird?
2. Welche Strategien entstehen im Laufe des Lernprozesses?
    - Verhalten sich die Agenten wie bekannte spieltheoretische Strategien (z. B. "Tit-for-Tatöder Älways Defect")?
    - Zeigen sich neuartige, unerwartete Verhaltensmuster?
3. Wie beeinflussen verschiedene Lernparameter das Verhalten der Agenten?

- Welche Rolle spielen Faktoren wie Belohnungsfunktionen, Discount-Faktor oder Explorationsstrategien?
- Wie wirken sich unterschiedliche Trainingsumgebungen auf die Ergebnisse aus?

Welche Rolle spielen Faktoren wie Belohnungsfunktionen, Discount-Faktor oder Explorationsstrategien? 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 Qualitaten 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. Dies erlaubt das Lernen in hochdimensionalen Zustandsräumen, die tabellarische Methoden überfordern würden. Die Hauptbestandteile von DQN sind:

1. Neuronales Netz als Q-Funktion
   - Das Netz nimmt den Zustand s als Eingabe und gibt geschätzte Q-Werte für alle möglichen Aktionen a aus.
   - Die Gewichte des Netzwerks werden durch Gradientenabstieg und einen Mean-Squared-Error (MSE)-Loss aktualisiert.
2. Erfahrungsspeicher (Experience Replay)
   - Anstatt direkt mit den neuesten Erfahrungen zu trainieren, werden vergangene Erfahrungen (s, a, r, s') in einem Speicher abgelegt.
3. Zielnetzwerk (Target Network)
   - Zusätzlich zum Hauptnetzwerk existiert eine separate Kopie, die in regelmäßigen Abständen aktualisiert wird.
   - Dies verhindert zu starke Schwankungen in den Q-Werten und stabilisiert das Training.

Die Aktualisierungsregel für DQN basiert auf dem Mean-Squared-Error zwischen dem vorhergesagten und dem Ziel-Q-Wert:

$$L = \left(r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_{\text{current}}(s, a) \right)^2 \tag{2.2}$$

DQN ist besonders mächtig für komplexe Umgebungen, in denen eine tabellarische Q-Funktion nicht mehr praktikabel ist.

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

Der Q-Learning Agent nutzt eine Q-Tabelle zur Approximation der Q-Werte. Hierbei steuert die Lernrate $\alpha = 0.01$, wie stark neue Informationen in die Q-Tabelle einfließen, während der Discount-Faktor $\gamma = 0.5$ 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 0.995 pro Episode reduziert, bis ein Minimum von 0.001 erreicht ist. Der Agent passt seine Q-Werte nach jeder Runde an und berücksichtigt dabei eine modifizierte Belohnungsfunktion: 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

	Gegner kooperiert	Gegner defektiert
Q-Agent kooperiert	0.2	-1
Q-Agent defektiert	2	-0.5

Tabelle 3.3: Q-Learning Belohnungsfunktion

(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 0.001). Das Lernen erfolgt über MSE-Verlust und Adam-Optimierung mit einer Lernrate von 0.001. Die Belohnungsfunktion ist leicht modifiziert, und sieht folgendermaßen aus: Während des

	Gegner kooperiert	Gegner defektiert
Deep Q-Agent kooperiert	2 oder 2.5, wenn letzte beide Runden -1	-2
Deep Q-Agent defektiert	1	-1

Tabelle 3.4: Deep Q-Learning Belohnungsfunktion

Trainings werden zufällige Minibatches aus der Replay-Memory gezogen und die Q-Werte mit der Bellman-Gleichung aktualisiert, wobei zukünftige Belohnungen mit $\gamma = 0.99$ diskontiert werden. Der Agent speichert seine letzten Aktionen in einem Zustandsvektor und nutzt diese zur Entscheidungsfindung.

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

4.2 Wichtige Erkenntnisse

5 Diskussion

5.1 Interpretation der Ergebnisse

5.2 Vergleich mit Erwartungen

5.3 Limitationen

6 Fazit und Ausblick

6.1 Zusammenfassung der Ergebnisse

6.2 Ausblick