Ein Vergleich von feature-basiertem und bildbasiertem Reinforcement Learning anhand des Spiels *Snake*

Sven Koch
sven.koch@haw-hamburg.de

Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Zusammenfassung. Eine der beliebtesten Herausforderungen der letzten Jahre auf dem Gebiet der künstlichen Intelligenz ist die Entwicklung von Agenten, die in der Lage sind, das Spielen von klassischen Videospielen zu perfektionieren. Diese Arbeit diskutiert den Einsatz von Reinforcement Learning zum eigenständigen Erlernen des Spiels *Snake*. Die Schwierigkeit in der Umsetzung eines RL-Verfahrens für *Snake* besteht in dem extrem großen Zustandsraum des Spiels. Zur Lösung dieses Problems führt diese Arbeit einen feature-basierten Q-Learning Ansatz und einen bildbasierten Deep Q-Learning Ansatz ein. Zur Modellierung des feature-basierten Ansatzes werden geeignete Features beschrieben, mit denen das Spiel erfolgreich abstrahiert werden kann. Mit den beiden vorgestellten Ansätzen können RL-Agenten trainiert werden, die das Spielen von *Snake* beherrschen. Der Vergleich der beiden Ansätze zeigt, dass der feature-basierte Ansatz gegenüber dem bildbasierten Ansatz in Trainingszeit und Spielperformance deutlich überlegen ist. Der feature-basierte Ansatz profitiert besonders durch den verkleinerten Zustandsraum, was jedoch auch zu Nachteilen in der Erkennung von bestimmten Spielsituationen führt.

Schlüsselwörter: Reinforcement Learning · Q-Learning · Deep Q-Learning · Snake Game.

1 Einleitung

Videospiele und Spiele im Allgemeinen sind seit geraumer Zeit ein beliebtes Aufgaben- und Forschungsgebiet im Bereich der künstlichen Intelligenz. Einerseits stellen die Spiele eine reale Herausforderung für das intelligente Verhalten und das maschinelle Lernen dar und sind dabei anderseits einfach zu formalisieren. [4, 1, 5]

Spieleentwickler haben zudem ein jahrzehntelanges Interesse, Agenten mit intelligentem Verhalten (KI-Spieler) zu entwerfen, die ein Spiel nur auf der Grundlage der eigenen Spielerfahrung erlernen können. Mit dem klassischen Prinzip der dynamischen Programmierung ist diese Aufgabenstellung nur bei Problemen

mit einer geringen Anzahl an Zuständen lösbar. Oftmals kann für komplizierte Zusammenhänge nur ein intelligentes Verhalten mit zuvor festgelegten Regeln simuliert werden. [4]

Reinforcement Learning (RL) ist eine der interessantesten Technologien im Bereich des maschinellen Lernens, bei der durch iteratives Ausprobieren die optimale Aktion in jedem Zustand erlernt wird. Dieses Verfahren ist insbesondere auch zur Lösung von umfangreichen Problemen einsetzbar und beispielsweise in Anwendungen nützlich, die aufgrund ihrer hohen Komplexion nicht durch die dynamische Programmierung umsetzbar sind. [4]

Exemplarisch wird in dieser Arbeit anhand des Videospiels *Snake* die Leistung von zwei RL-Verfahren untersucht. Das Spiel *Snake* besitzt allgemein große Bekanntheit, ist einfach darstellbar und zugleich ausreichend komplex, um als geeigneter Vergleichsmaßstab zu dienen. [4]

Hierzu setzt sich diese Arbeit zweierlei Ziele. Zunächst soll ein möglichst leistungsfähiger RL-Agent entwickelt und trainiert werden, der gute Ergebnisse im Spiel *Snake* erreicht. Dazu führt diese Arbeit in die RL-Verfahren Q-Learning und Deep Q-Learning ein. Während das Q-Learning mit einer feature-basierten Repräsentation des Zustandsraums arbeitet, nutzt das Deep Q-Learning die vollständig verfügbaren bildbasierten Informationen zur Kontrolle des Agenten.

Damit einhergehend sollen als weiteres Ziel die Eigenschaften mit den verbundenen Vor- und Nachteilen des feature-basierten und bildbasierten Reinforcement Learnings erarbeitet werden.

1.1 Verwandte Arbeiten

Es existieren zahlreiche Arbeiten über die Forschung im Bereich der künstlichen Intelligenz für Spiele-Agenten. Die bekannteste Erfolgsgeschichte im Einsatz von Reinforcement Learning ist *TD-gammon* [9], ein RL-Agent für das Spiel Backgammon aus den 1990er-Jahren. Inzwischen konzentriert sich die Forschung nicht mehr nur auf Spiele, die sich in einfacher Form durch symbolische Darstellungen beschreiben lassen, wie z. B. Brett- und Kartenspiele, sondern auch auf grafikbasierte Videospiele. [4]

Eine hohe Bekanntheit auf diesem Gebiet hat die von DeepMind Technologies (2013) [5] veröffentlichte Arbeit erlangt, in der ein RL-Modell zum Erlernen einer erfolgreichen Kontrollstrategie für verschiedene klassische *Atari*-Spiele beschrieben wird. Durch den Einsatz eines Deep Q-Learning Verfahrens mit einem Convolutional Neural Network (CNN) können die hochdimensionalen Bildinformationen der Videospiele als direkte Eingabe für den RL-Agenten verwendet werden. Durch diese Eigenschaft ist die vorgestellte RL-Architektur generalisierbar und kann auf verschiedene Titel aus der *Atari*-Familie angewendet werden. [5, 6]

Ma et al. (2016) [4] sowie Sebastianelli et al. (2021) [7] beschreiben in ihren Arbeiten bereits den Einsatz von Reinforcement Learning zum Erlernen des Spiels *Snake*. Die beiden Ansätze betrachten jedoch nur feature-basierte Lösungen und verwenden nicht die gesamten verfügbaren Bildinformationen des *Snake*-Spiels.

2 *Snake*

Snake ist ein Videospiel, bei dem der Spieler eine Schlange auf einem begrenzten Spielfeld steuert (vgl. Abbildung 1). Das Ziel des Spiels ist es, zufällig erscheinende Äpfel zu fressen und dabei eine Kollision mit der eigenen Schlange oder der Spielfeldbegrenzung zu vermeiden. Mit jedem gefressenen Apfel wird die Schlange länger, sodass es immer schwieriger wird, dem eigenen Schlangenkörper auszuweichen. Der Spieler verliert, wenn die Schlange gegen den Spielfeldrand oder gegen sich selbst gesteuert wird.

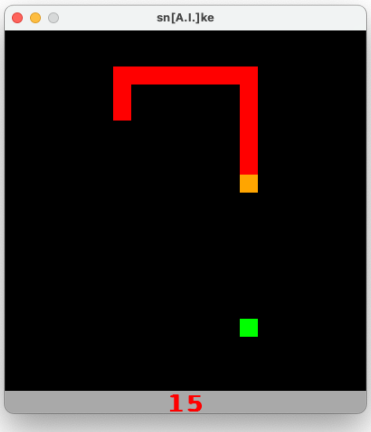


Abb. 1: Bildschirmfoto des Videospiels *Snake*. Der Schlangenkopf ist orange, der Schlangenkörper ist rot und der Apfel ist grün gefärbt.

2.1 Mathematische Analyse

Abstrakt lässt sich jeder Zug des Spiels *Snake* als Suche nach einem selbstmeidenden Pfad (engl. self-avoiding walk (SAW)) innerhalb eines $\mathbb{R}^2$-Gitters betrachten. Eine mathematische Beschreibung des *Snake*-Spiels kann wie folgt formuliert werden:

- Das Spielfeld der Größe $m \times n$ wird als Graph $G = (V, E)$ dargestellt, wobei $V = \{v_i\}$ die Menge der Knoten ist, wovon jeder Knoten einem Quadrat auf dem Spielfeld entspricht, und die Menge der Kanten $E = \{e_{ij} | v_i \text{ ist benachbart zu } v_j\}$. [4]
- Die Schlange wird als Pfad $\{u_1, \ldots, u_k\}$ modelliert, wobei u_1 und u_k der Kopf der Schlange bzw. das Ende sind. [4]

NP-Schwere Nach Viglietta (2013) [10] ist jedes Spiel mit sammelbaren Gegenständen, Ortstraversierungen und One-Way-Pfaden NP-schwer. Diese Eigenschaften sind auf das Spiel *Snake* zutreffend. Da sich die Schlange nicht selbst kreuzen darf, muss zum Erreichen eines Apfels stets ein One-Way-Pfad traversiert werden. Folglich ist es NP-schwer, in jedem Spielzug den kürzesten selbstmeidenden Pfad zu finden, wenn gleichzeitig keine weiteren Kenntnisse über die Abfolge und Position des Auftretens von Äpfeln bekannt sind. [4, 10]

2.2 Aktionsraum

Der Aktionsraum der Schlange umfasst vier mögliche Bewegungen, die den Himmelsrichtungen Nord, Ost, Süd und West entsprechen (vgl. Abbildung 2). Die Ausführung einer Aktion resultiert in der Fortbewegung um ein Quadrat auf dem Spielfeld. In jedem Spielzug muss eine Aktion getätigt werden. Abhängig der Position des Schlagenkörpers führt jeweils eine der vier Aktionen zu einer Bewegung in den eigenen Körper, was den Tod der Schlange und somit das Ende des Spiels bedeutet. [3]

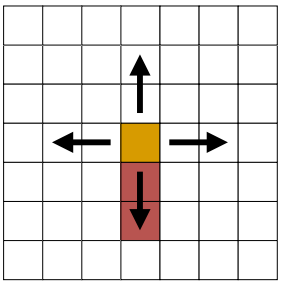


Abb. 2: Mögliche Aktionen im Spiel *Snake*

2.3 Realisierung

Das *Snake*-Spiel wurde in der Programmiersprache *Python* implementiert. Zur Umsetzung der grafischen Benutzeroberfläche (GUI) wurde auf die *Python*-Bibliothek *Pygame* zurückgegriffen.

Die Größe der Spielfläche ist auf 20×20 Felder festgelegt. Ein neues Spiel startet mit einer Schlange der Länge zwei an einer zufälligen Position.

3 Hintergrund zu Q-Learning und Deep Q-Learning

Der als Reinforcement Learning bekannte Teilbereich des maschinellen Lernens beschreibt eine Bandbreite von Ansätzen, bei denen ein Agent durch die direkte

Interaktion mit seiner Umgebung eine optimale Strategie (engl. policy) erlernt, um ein bestimmtes Ziel zu erreichen. [8]

Der RL-Agent beobachtet in einem iterativen Prozess die Umgebung und führt eine Aktion aus, mit der Einfluss auf den Zustand der Umgebung genommen wird. Von der Umgebung erhält der Agent ein Belohnungssignal, welches die Güte der Aktionen bewertet. Das Ziel des Agenten besteht in der Regel darin, eine optimale Strategie zu erlernen, indem durch geeignete Aktionen die erwartete kumulative Belohnung über die Zeit maximiert wird. Die Modellierung der Belohnungsfunktion bildet die Charakteristik des zu lösenden Optimierungsproblems ab und gibt an, welches Ziel angestrebt wird. [7, 8]

Ein Markov Decision Process (MDP) ist eine klassische Formulierung eines sequentiellen Entscheidungsproblems und bildet somit den Rahmen zur Darstellung von RL-Problemen. Ein MDP kann als ein Tupel $(S, A, R, P, \gamma, p_0)$ definiert werden, wobei S der Zustandsraum (mit $s \in S$ als beliebiger Zustand), A der Aktionsraum (mit $a \in A$ als beliebige Aktion), $R(\cdot|s, a)$ die Wahrscheinlichkeitsverteilung für die erwartete Belohnung im Zustand s bei Aktion a, $P(\cdot|s, a)$ die Wahrscheinlichkeitsverteilung für den Übergang in den Folgezustand aus dem Zustand s bei Aktion a, $\gamma \in [0, 1]$ der Diskontierungsfaktor und p_0 die Wahrscheinlichkeitsverteilung über die möglichen Startzustände ist. [7, 1]

Die Strategie des Agenten $\pi(\cdot) := S \to A$ ist eine Abbildung von Zuständen auf eine Wahrscheinlichkeitsverteilung über den gesamten Aktionsraum A. Im Einzelnen läuft die Interaktion zwischen dem Agenten und der Umwelt wie folgt ab: Zu jedem Zeitpunkt $t = 1, 2, \ldots$ erhält der Agent eine Repräsentation des Zustands $s_t \in S$ der Umgebung und wählt stochastisch eine Aktion $a_t \sim \pi(\cdot|s_t)$. Anschließend erhält der Agent eine Belohnung $r_{t+1} \sim R(\cdot|s_t, a_t)$ als Reaktion auf die ausgeübte Aktion. Die Umgebung wechselt in den neuen Zustand $s_{t+1} \sim P(\cdot|s_t, a_t)$. Mit den Indizes t und $t + 1$ werden die Werte des Systemzustands und die ausgewählten Aktionen während einer bestimmten Episode bezeichnet. Mit S_t und R_t werden die zeitdiskreten Zufallsvariablen, die mit $P(\cdot|s, a)$ bzw. $R(\cdot|s, a)$ assoziiert sind, bezeichnet. [7, 8]

Das Ziel des Agenten ist es, eine optimale Strategie zu finden, die die erwartete kumulative Belohnung über einen definierten Zeithorizont N maximiert. Sowohl in einem episodischen MDP (mit einem endlichen Zeithorizont N) als auch in einem kontinuierlichen MDP (mit $N \to \infty$) stellt die Abfolge von Zuständen, Aktionen und Belohnungen in einer Episode eine Trajektorie der angewendeten Strategie dar. Jede Trajektorie akkumuliert die gesammelten Belohnungen der Umgebung. Demzufolge kann die folgende Gleichung für die Berechnung der diskontierten Gesamtbelohnung angewendet werden

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,. \tag{1}$$

Es sei $Q^\pi(s, a)$ definiert als die Q-Wert Funktion einer beliebigen Strategie π. Diese Funktion entspricht der erwarteten Belohnung, die sich aus der Ausführung

einer Aktion a im Zustand s und der weiteren Anwendung der Strategie π ergibt, d. h.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right] . \tag{2}$$

Die optimale Strategie π^* maximiert $Q^{\pi}(s,a)$ und entspricht der folgenden Bellman-Gleichung

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in A} Q^*(s_{t+1}, a') \middle| S_t = s, A_t = a\right] . \tag{3}$$

Das Q-Learning ermöglicht die Berechnung der optimalen Q-Wert Funktion $Q^*(s,a)$ durch Approximation der Bellman-Gleichung (3) mittels einer ausreichend großen Anzahl an Stichproben. Der erwartete Q-Wert $Q(s_t, a_t)$ wird während des Trainings gemäß der Regel

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \tag{4}$$

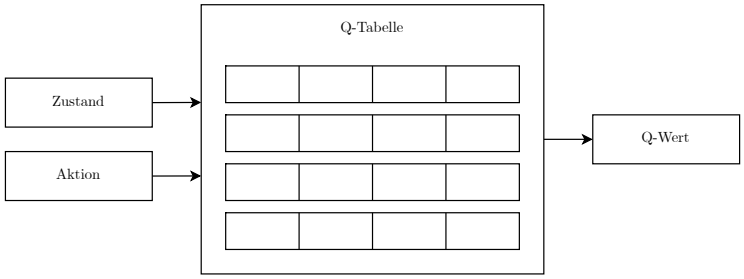
aktualisiert, wobei α die Lernrate und r_{t+1} das Belohnungssignal ist, das der Agent erhält, wenn die Aktion a_t im Zustand s_t ausführt wird. [7]

In der einfachsten Form speichert ein Q-Learning Algorithmus die erlernten Q-Werte in einer Tabelle (vgl. Abbildung 3a). Dazu ist für jede Zustands-Aktions-Kombination ein Eintrag in der Tabelle vorgesehen. Durch die Erweiterung mit künstlichen neuronalen Netzwerken ist das Deep Q-Learning entstanden, bei dem neuronale Netze zur Approximation der Q-Wert Funktion verwendet werden (vgl. Abbildung 3b). [7]

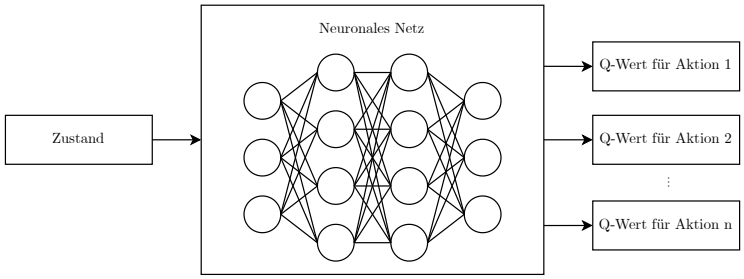
Durch das Speichern aller Q-Werte in einer Umsetzungstabelle (engl. lookup table) ist die Anzahl der möglichen Zustände und Aktionen beim einfachen Q-Learning begrenzt. Große Zustands- und Aktionsräume lassen die Größe dieser Tabelle stark anwachsen, wodurch ebenfalls der benötigte Speicherbedarf sowie die Trainingszeit deutlich ansteigt. Nach Möglichkeit kann der Zustandsraum durch einzelne adäquat gewählte Features abstrahiert werden. Die für den Agenten zur Verfügung stehenden Informationen werden dadurch auf eine kleinere Zustandsmenge reduziert. Bei der Wahl und Entwicklung dieser Features muss beachtet werden, dass keine wichtigen Daten aus der Gesamtinformationsmenge verloren gehen. [1]

Das Deep Q-Learning mit künstlichen neuronalen Netzen kann in der Regel größere Informationsmengen verarbeiten. Mit dieser Architektur sind unter anderem auch Agenten mit wertkontinuierlichen Zustands- und Aktionsräumen möglich. Durch den Einsatz eines neuronalen Netzwerks können während des Lernprozesses selbstständig informationstragende Merkmale aus der Menge der Gesamtinformationen extrahiert werden. [1]

Das Trainieren von RL-Agenten, die direkte hochdimensionale Beobachtungen aufnehmen, birgt die Gefahr von auftretenden Instabilitäten bei der Funktionsapproximation. Die neuronalen Netze tendieren dazu, alte Lernerfolge zu



(a) Q-Learning



(b) Deep Q-Learning

Abb. 3: Q-Learning vs. Deep Q-Learning

vergessen, da bei der Interaktion mit der Umgebung die Trainingsdaten stets in zeitlich geordneter Reihenfolge abgearbeitet werden und vergangene Situationen unter Umständen nicht erneut auftreten und trainiert werden. Zur Lösung dieses grundlegenden Instabilitätsproblems führt erstmals DeepMind Technologies [6] die Verwendung der beiden Techniken Experience-Replay und Target-Networks ein. [1]

Ein Experience-Replay Buffer speichert Transitionen der Form $(s_t, a_t, s_{t+1}, r_{t+1})$ in einem zyklischen Pufferspeicher und ermöglicht es dem RL-Agenten auf zuvor beobachtete Daten zurückzugreifen und erneut für das Training zu verwenden. Durch die zufällige Auswahl von Stichproben aus dem Pufferspeicher können die zeitlichen Korrelationen, die sich negativ auf das Training von neuronalen Netzen auswirken können, aufgehoben werden. Zudem wird damit die Anzahl der erforderlichen Interaktionen mit der Umgebung massiv reduziert. [1]

Die zweite Stabilisierungsmethode besteht in der Verwendung eines separaten Target-Netzwerks, das im Training zusätzlich zum bestehenden Policy-Netzwerk eingesetzt wird. Die Gewichte des Target-Netzwerks werden zunächst identischen zu den Gewichten des Policy-Netzwerks initialisiert. Das Target-Netzwerk wird aber für einen langen Zeitraum eingefroren. Anstatt den Fehler auf der Grundlage des eigenen Policy-Netzwerks mit schnell und stark schwankenden Schätzungen der Q-Werte berechnen zu müssen, wird stattdessen das eingefrorene Target-Netzwerk verwendet. Während des Trainings werden die Gewichte

des Target-Netzwerks nach einer festen Anzahl von Schritten mit den Gewichten des Policy-Netzwerks aktualisiert. [1]

Eine zusätzliche Herausforderung im Reinforcement Learning ist das maßgebliche Dilemma zwischen Exploration und Exploitation. Zur Auswahl der Aktion a_t muss stets ein Kompromiss zwischen dem Ausprobieren einer vermeintlich nicht optimalen Aktion (Exploration) oder der Ausnutzung der optimalen Aktion (Exploitation) gefunden werden. Durch zufällig gewählte Aktionen kann der Agent noch unbekannte Zustände der Umgebung erkunden, wohingegen nützliche Fortschritte auf der Grundlage bereits erlernter bestmöglicher Aktionen erzielt werden. [7, 1]

Off-Policy-Algorithmen, wie beispielsweise das Q-Learning, verwenden typischerweise die einfache ϵ-Greedy-Explorationsstrategie, bei der eine zufällige Aktion mit der Wahrscheinlichkeit $\epsilon \in [0, 1]$ ausgeführt wird und andernfalls die optimale Aktion ausgewählt wird. Indem ϵ mit der Zeit abnimmt, wird mit voranschreitendem Trainingsfortschritt immer häufiger die erlernte bestmögliche Aktion getätigt. [1]

4 Methodik

Eine der größten Schwierigkeiten bei der Umsetzung und Implementierung erfolgreicher RL-Algorithmen für Videospiele ist die enorme Größe ihrer Zustandsräume. [4]

Exemplarisch für das Spiel *Snake* sei ein Spielfeld der Größe $n \times n$ angenommen. Naiv betrachtet könnte jedes Feld auf dieser Spielfläche unter Verwendung der genauen Position der Schlange und des Apfels durch eine der vier Bedingungen parametrisiert werden: $\{$enthält den Apfel, enthält den Kopf der Schlange, enthält den Körper der Schlange, leer$\}$. Nach den Regeln der einfachen Kombinatorik würde die Größe des Zustandsraums $|S|$ die Bedingung $|S| > (n^2)^4$ erfüllen, was bereits für kleinere Werte für n enorm groß ist. [4]

Mit den feature-basierten und bildbasierten RL-Ansätzen existieren zwei Methoden, um das Videospiel trotz seiner Komplexität bewältigen zu können.

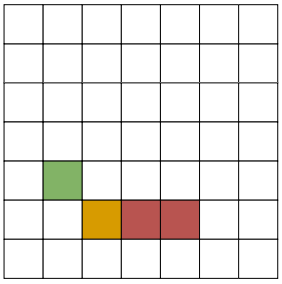
4.1 Feature-basierter Ansatz mit Q-Learning

Die zuvor erläuterte Problematik zusammenhängend mit dem sehr großen Zustandsraum des Spiels und der begrenzten maximalen Größe einer Q-Tabelle erfordert für das Q-Learning vorab eine Definition von qualifiziert gewählten Features, um die Informationsmenge zu reduzieren.

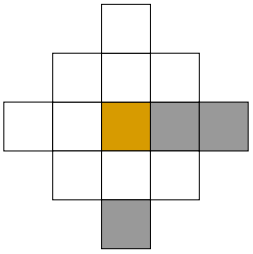
In den folgenden Absätzen wird die Modellierung des Zustandsraums, der Belohnungsfunktion und des Trainings für den feature-basierten Q-Learning Ansatz beschrieben.

Zustandsraum Zur Reduktion der Informationen des gesamten Spielfeldes wird ein begrenztes Sichtfeld definiert. Der Zustandsraum setzt sich aus diesem Sichtfeld und der relativen Position des Apfels zum Kopf der Schlange zusammen.

Der Sichtbereich des Agenten besteht aus zwölf Feldern, die um den Kopf der Schlange angeordnet sind. Ein Feld kann entweder frei oder von einem Hindernis belegt sein. Die Spielfeldbegrenzung und der Schlangenkörper werden dabei nicht unterschieden und gleichermaßen als Hindernis gewertet. Der Apfel stellt kein Hindernis dar. Die Abbildung 4 illustriert die Zusammenhänge des Sichtfeldes. Aus der Größe des Sichtbereichs ergeben sich $2^{12} = 4.096$ mögliche Kombinationen für Hindernisse im Sichtfeld. [3]



(a) Spielsituation mit drei Hindernissen im Sichtfeld der Schlange.



(b) Zwölf Felder großes Sichtfeld angeordnet um den Kopf der Schlange. Die Felder mit Hindernissen sind grau gefärbt.

Abb. 4: Sichtfeld des Q-Learning Agenten

Die Position des Apfels auf dem Spielfeld wird relativ zum Kopf der Schlange betrachtet. Der Apfel kann dabei in einer von acht verschiedenen Richtungen aus der Menge {Nord, Nordost, Ost, Südost, Süd, Südwest, West, Nordwest} zum Kopf liegen. Da in jedem Spielschritt genau ein Apfel auf dem Spielfeld existiert, gibt es acht mögliche Kombinationen für die relative Position des Apfels. [3]

Um den gesamten Zustandsraum zu beschreiben, werden Sichtfeld und relative Apfelposition kombiniert. Es ergeben sich somit $2^{12} \cdot 8 = 32.768$ mögliche Zustände. Hierbei handelt es sich jedoch nur um die rechnerische Anzahl der möglichen Zustände. In der Praxis können einige Zustandskombinationsmöglichkeiten nie erreicht werden. [3]

Belohnungsfunktion Im Erfolgsfall beim Erreichen eines Apfels erhält der Agent eine hohe positive Belohnung. Analog dazu wird im Falle einer Niederlage eine hohe negative Belohnung an den Agenten verteilt. Zusätzlich werden Belohnungen für das Annähern bzw. das Entfernen der Schlange an den Apfel gegeben. Die euklidische Distanz zwischen dem Kopf der Schlange und der Position des Apfels stellt somit ein weiteres Feature dar und hilft dabei, die benötige Trainingszeit weiter zu verkürzen. [3]

Die Zahlenwerte der einzelnen Belohnungen können der Tabelle 1 entnommen werden.

Tabelle 1: Belohnungen für den feature-basierten RL-Agenten

Ereignis	Apfel gefressen	gestorben	an Apfel angenähert	von Apfel entfernt
Belohnung	+120	−60	+1	−1

Training Während des Trainings kommt die ϵ-Greedy-Explorationsstrategie und ein ähnlicher Algorithmus zur schrittweisen Herabsetzung der Lernrate α zum Einsatz.

Zum Start des Trainings wird ϵ auf einen Wert von 0,9 initialisiert, damit der Agent zunächst seine Umwelt erforschen kann. Im Laufe des Lernens wird ϵ linear über die ersten Zweidrittel der gesamten Trainingssitzung auf einen Wert von 0,01 reduziert. Im letzten Drittel des Trainings bleibt ϵ konstant auf dem Minimalwert 0,01. [3]

Analog dazu sollte die Lernrate α ebenfalls abklingen, um die Q-Werte zu stabilisieren, sodass langsam zu einer optimalen Q-Tabelle konvergiert werden kann. Ein höherer Wert für α bedeutet, dass die aktuellen Belohnungen mit einer höheren Gewichtung in die Berechnung zur Aktualisierung der Q-Tabelle einbezogen werden. Die Lernrate wird zu Beginn auf einen Startwert von 0,3 festgelegt und während der ersten zwei Trainingsdrittel linear auf 0,1 abgesenkt. [3]

4.2 Bildbasierter Ansatz mit Deep Q-Learning

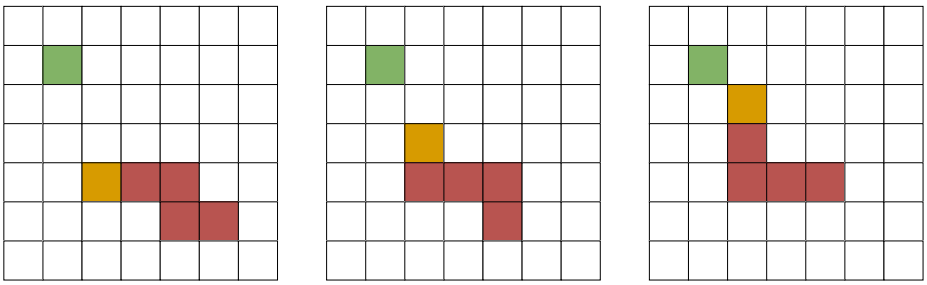
Für das Deep Q-Learning werden die gesamten visuellen Informationen der aktuellen Spielsituation verwendet, ohne dass im Voraus bestimmte Features definiert werden.

Nachfolgend wird die Gestaltung des Zustandsraums, der Belohnungsfunktion und des Trainingsablaufs für das bildbasierten Deep Q-Learning vorgestellt.

Zustandsraum Der Deep Q-Learning Agent nutzt die Bilddaten des *Snake*-Spiels als Eingabe. Die direkte Arbeit mit den Rohbildern im RGB-Farbraum ist jedoch rechenintensiv, weshalb ein einfacher Vorverarbeitungsschritt durchführt wird, um die Dimensionalität der Eingabe zu reduzieren. Dabei wird die RGB-Darstellung in Graustufen umgewandelt. Zudem wird den erzeugten Grauwertbildern ein weißer Rahmen mit der Breite von einem Pixel hinzugefügt, der die Spielfeldbegrenzung kennzeichnet. Bei einer Spielfläche der Größe 20×20 beträgt die Größe eines Bildes nach der Datenvorverarbeitung folglich 22×22 Pixel. [3, 6]

Durch ein einzelnes statisches Bild kann der Agent keine Kenntnisse über die Bewegungsrichtung der Schlange erlangen. Zur Erkennung der Bewegungsdyna-

mik wird daher ein Frame-Stack (vgl. Abbildung 5) gebildet, der die Bildinformationen des aktuellen und der vergangenen zwei Spielzüge enthält. [3, 5]



(a) Frame zum Zeitpunkt $t-2$

(b) Frame zum Zeitpunkt $t-1$

(c) Frame zum Zeitpunkt t

Abb. 5: Frame-Stack

Die endgültige Eingabe zur Repräsentation eines Zustands ist dementsprechend in der Form $(22, 22, 3)$.

Belohnung Mit dem bildbasierten Ansatz wird die Umwelt nur visuell beobachtet. Somit sind keine zusätzlichen Kenntnisse, wie z. B. die Entfernung zwischen Schlangenkopf und Apfel, vorhanden. Demzufolge kann die Spielumgebung nur Belohnungen für das Fressen eines Apfels oder den Tod der Schlange zurückgeben.

Die entsprechende Größenordnung der Belohnung für die jeweiligen Ereignisse können Tabelle 2 entnommen werden.

Tabelle 2: Belohnungen für den bildbasierten RL-Agenten

Ereignis	Apfel gefressen	gestorben
Belohnung	$+120$	-60

Training Die Architektur des neuronalen Netzes zur Approximation der Q-Wert Funktion ist in Abbildung 6 dargestellt. Das Netzwerk besteht aus drei hintereinander angeordneten Faltungsschichten (engl. convolutional layer), die speziell für die Verarbeitung von zweidimensionalen Bilddaten geeignet sind und zwei abschließenden Dense-Schichten. Die erste Schicht ist die Eingabeschicht und nimmt den Vektor der drei 22×22 Pixel großen Grauwertbilder des *Snake*-Spiels entgegen. Die folgenden Schichten nutzen die ReLU-Aktivierungsfunktion

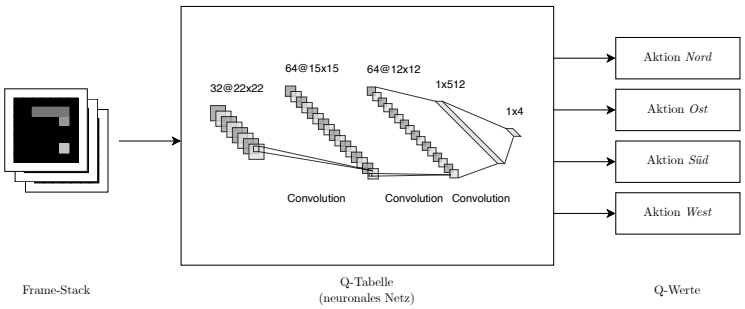


Abb. 6: Architektur des neuronalen Netzes für das Deep Q-Learning

und dienen zur Extraktion der informationsbehafteten Merkmale aus den Rohbilddaten. Die letzte Schicht ist die Ausgangsschicht, die mithilfe einer linearen Aktivierungsfunktion die beste Aktion für die jeweiligen Eingaben vorhersagt. Die Anzahl der Ausgangsneuronen korrespondiert mit den vier möglichen Bewegungen der Schlange. Die beste Aktion wird durch die Verwendung des größten der vier vorhergesagten Q-Werte ausgewählt. [2]

Die beschriebene Architektur wurde in *Python* mit *Keras* und *Tensorflow* umgesetzt. Das neuronale Netz wird mit dem Backpropagation-Verfahren trainiert, wobei der Adam Optimierungsalgorithmus und die Huber Loss-Funktion verwendet werden. [2]

Zur Vermeidung von Instabilitäten während des Trainings kommt ein Experience-Replay Buffer und ein separates Target-Netzwerk zum Einsatz. Der Experience-Replay Buffer speichert die letzten 100.000 Transitionen. Die Gewichte des Target-Netzwerks werden jeweils nach 1.000 Trainingsepisoden aktualisiert. [3]

Zudem wird die zuvor für den feature-basierten Q-Learning Algorithmus beschriebene ϵ-Greedy-Explorationsstrategie verwendet, um den Parameter ϵ über die Trainingszeit herabzusenken. [3]

5 Auswertung

Mit den beiden vorgestellten RL-Algorithmen ist es realisierbar, Agenten zu entwickeln, die das Spiel *Snake* erlernen. Zwischen den beiden Verfahren kommt es beim Training und in der Leistungsfähigkeit im Spiel jedoch zu größeren Abweichungen.

5.1 Lernkurven

Die Lernkurven für das Training des Q-Learning und des Deep Q-Learning Agenten sind in den Abbildungen 7 und 8 dargestellt.

In beiden Diagrammen ist der Trainingsfortschritt durch die mit der Zeit steigende Belohnung pro Episode deutlich erkennbar. Der Q-Learning Ansatz

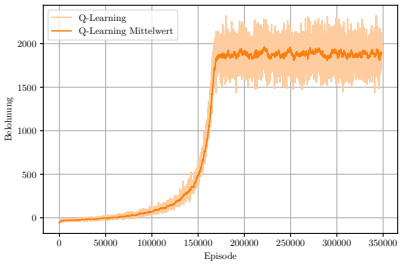


Abb. 7: Lernkurve des Q-Learnings

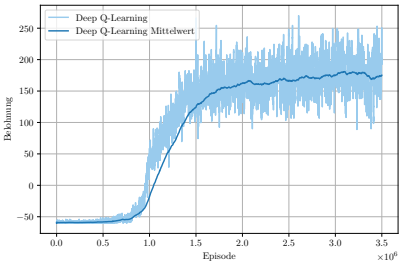


Abb. 8: Lernkurve des Deep Q-Learnings

erreicht nach ca. 170.000 Episoden eine Konvergenz bei einer mittleren Belohnung von 1.850. Das Training des Deep Q-Learnings konvergiert hingegen erst nach über zwei Millionen Trainingsepisoden bei einer durchschnittlichen Belohnung pro Episode von 170.

Ursächlich für den deutlich höheren Zeitbedarf des Deep Q-Learning Ansatzes ist der größere Zustandsvektor mit den Pixeldaten des *Snake*-Spiels. Zudem erhält der Deep Q-Learning Agent keine zusätzliche Belohnung abhängig zur Veränderung der Distanz zwischen Schlange und Apfel. Im bildbasierten Verfahren muss der Agent das Ziel des Spiels, also das Fressen von Äpfeln, ausschließlich durch die Exploration der Umgebung erlernen, wodurch mehr Trainingszeit nötig ist. Der feature-basierte Ansatz konvergiert darüber hinaus gegen eine deutlich höhere Belohnung pro Episode.

Die Höhe der Belohnungen der beiden RL-Ansätze ist jedoch nicht direkt miteinander vergleichbar. Zwar liegen die Werte für die Spielereignisse „gestorben“ und „Apfel gefressen“ in derselben Größenordnung (vgl. Tabelle 1 und 2), allerdings erhält der feature-basierte Ansatz zudem eine kleine positive Belohnung für das Annähern an den Apfel. Ein erfolgreicher Q-Learning Agent akkumuliert daher auch durch die Annäherung an den Apfel eine gewisse Belohnung.

Die große Diskrepanz zwischen den Lernkurven der beiden Agenten ist damit allerdings nicht zu erklären. Die offensichtliche Hauptursache ist, dass der feature-basierte Q-Learning Agent das Spiel besser beherrscht, weniger häufig stirbt und mehr Äpfel frisst. Diese Beurteilung lässt sich sowohl qualitativ als auch in einem quantitativen Leistungsvergleich feststellen.

5.2 Leistungsvergleich

Der Q-Learning-Agent zeigt im einfachen Spiel sehr gute Ergebnisse. Von Vorteil erweist sich dabei der durch die Features abstrahierte kompakte Zustandsraum. Durch diese Abstraktion wird eine sehr gute Lernperformance erzielt, wodurch bereits nach sehr kurzen Trainingszeiten (einige Minuten bis wenige Stunden) gut funktionierende Agenten hervorgebracht werden können. Für einen primitiveren Agenten wäre auch ein noch kleinerer Zustandsraum mit einem einge-

schränkteren Sichtfeld denkbar. Dieses Ergebnis zeigt, dass sich bereits mit einfach gewählten Features (Einschränkung des Sichtfeldes, relative Apfelposition) ein recht erfolgreicher Agent entwickeln lässt. [3]

Eine größere Schwäche des feature-basierten Ansatzes ist allerdings, dass der Agent aufgrund des beschränkten Sichtfeldes keine Bildung von Sackgassen mit dem eigenen Schlangenkörper erkennen kann (vgl. Abbildung 9). Eine einfache Vergrößerung des Sichtfeldes kann dieses Problematik nicht zuverlässig lösen, sondern würde bestenfalls nur zu einem Aufschub des Problems führen. Der Agent müsste langfristig die Informationen über die gesamte Spielfläche kennen, um die Bildung von Sackgassen vollständig und frühzeitig erkennen zu können. Über die Größe des Sichtfeldes kann jedoch nicht unbegrenzt skaliert werden, da mit einer Vergrößerung eine exponentielle Zunahme der möglichen Zustände einhergeht. Mit einer entsprechend großen Q-Tabelle sinkt die Lernperformance des Agenten drastisch. [3]

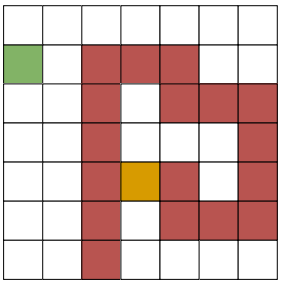


Abb. 9: Bildung einer Sackgasse. Der Agent erkennt mit seinem begrenzten Sichtfeld nur das Hindernis in westlicher Richtung, nicht aber in nördlicher Richtung. Um den kürzesten Weg zum Apfel zu verfolgen, wählt der Agent im nächsten Spielzug die Bewegung in nördliche Richtung und bildet somit eine unausweichliche Sackgasse mit dem eigenen Schlangenkörper.

Der Deep Q-Learning Agent ist hingegen in der Lage, die gesamten Spielinformationen als Eingabe zu verarbeiten. Im Training werden die bedeutenden Merkmale aus den Bilddaten selbstständig herausgearbeitet. Dadurch ist der Agent nicht länger abhängig von den manuell definierten Features, von deren Güte die Leistung des Agenten maßgeblich abhängt. [3]

Der Deep Q-Learning Ansatz liefert ebenfalls einen Agenten, der das Spiel *Snake* grundsätzlich beherrscht. Jedoch ist die Leistungsfähigkeit dieses Agenten im direkten Vergleich mit dem Q-Learning deutlich unterlegen. Der Deep Q-Learning Agent stirb schneller und frisst weniger Äpfel pro Spiel. Somit konnte der erwünschte Vorteil des bildbasierten Ansatzes mit den vollständigen Informationen zur frühzeitigen Erkennung und Vermeidung von Sackgassen nicht ausgenutzt werden. Aufgrund des großen Informationsgehalts der Bilddaten wer-

den bereits lange Trainingszeiten (mehrere Stunden bis einige Tage) benötigt, um einen Agenten zu trainieren, der das Spiel *Snake* nur grundlegend beherrscht.

Zur Quantifizierung dieser Beobachtungen wurden zwei Metriken eingeführt, mit denen die Leistungen der beiden RL-Verfahren im Spiel direkt miteinander verglichen werden können.

Dazu wurden jeweils 1.000 Spiele mit den Agenten der beiden RL-Algorithmen durchgeführt und dabei die durchschnittliche Anzahl der Spielzüge pro Spiel und die durchschnittliche Anzahl der gefressenen Äpfel pro Spiel protokolliert. Je besser der Agent das Spiel beherrscht, desto mehr Spielzüge können durchgeführt und desto mehr Äpfel können gefressen werden. Die Abbildungen 10 und 11 stellen die ermittelten Ergebnisse dar.

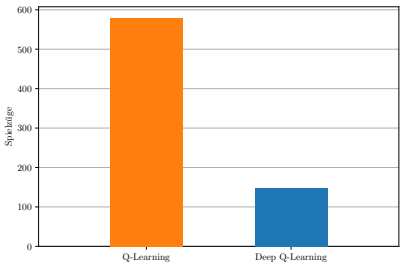


Abb. 10: Anzahl der durchschnittlich ausgeführten Spielzüge pro Spiel

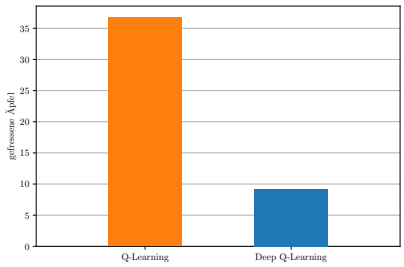


Abb. 11: Anzahl der durchschnittlich gefressenen Äpfel pro Spiel

Der feature-basierte Q-Learning Agent führt pro Spiel durchschnittlich 579 Züge aus und frisst dabei im Mittel 37 Äpfel. Der bildbasierte Deep Q-Learning Agent erreicht hingegen nur 148 durchschnittliche Spielzüge bzw. neun gefressene Äpfel in einem Spiel. Damit übersteigt die Leistung des feature-basierten Agenten die Leistung des bildbasierten Agenten ungefähr um das Vierfache.

Weiter ist zu beachten, dass für den feature-basierten Ansatz bei besserer Leistung eine wesentlich kürzere Trainingszeit notwendig war als für den bildbasierten Ansatz.

6 Fazit

In dieser Arbeit wurde ein Q-Learning-basierter und ein Deep Q-Learning-basierter Ansatz zum Erlernen des klassischen Videospiels *Snake* vorgestellt. Während der Zustandsraum für das Q-Learning durch geeignet gewählte Features abstrahiert und vereinfacht wird, nutzt das Deep Q-Learning die vollständigen bildbasierten Informationen des Spiels.

Mit beiden Ansätzen konnten RL-Agenten trainiert werden, die das Spiel *Snake* zumindest grundlegend beherrschen. Dabei übersteigen die Leistungen des

feature-basierten Ansatzes die Leistungen des bildbasierten Ansatzes deutlich. Der feature-basierte Ansatz profitiert durch den stark verkleinerten Zustandsraum besonders bei der Trainingsgeschwindigkeit.

Allerdings sind mit der Abstraktion des Zustandsraums auch Einschränkungen verbunden. Im Speziellen kann im untersuchten Spiel *Snake* die Bildung von Sackgassen ohne Informationen über die gesamte Umgebung nicht rechtzeitig erkannt werden. Theoretischen Überlegungen zufolge hätte ein bildbasiertes Verfahren mit vollständiger Kenntnis über die gesamte Spielfläche deutlich verbesserte Möglichkeiten. Durch die enorm langen Trainingszeiten, die für die bildbasierten Deep Q-Learning Agenten benötigt werden, konnten diese Vorteile in dieser Arbeit jedoch nicht ausspielt werden.

Einen Referenzwert für die Trainingsdauer von bildbasierten Deep Q-Learning Algorithmen liefert das *Atari*-Projekt von DeepMind Technologies. Das RL-Modell trainierte ungefähr 38 Tage, um Spielergebnisse zu erreichen, die menschliche Experten übertreffen. [6, 2]

Literaturverzeichnis

[1] Arulkumaran, Kai ; Deisenroth, Marc P. ; Brundage, Miles ; Bharath, Anil A.: Deep Reinforcement Learning: A Brief Survey. In: *IEEE Signal Processing Magazine* 34 (2017), Nr. 6, S. 26–38

[2] Chapman, Jacob ; Lechner, Mathias: *Keras Documentation: Deep Q-learning for atari breakout*. Mai 2020. – URL `https://keras.io/examples/rl/deep_q_network_breakout/`. – abgerufen am 24.01.2022

[3] Denecke, Eric ; Maass, Fynn L. ; Koch, Sven: *Q- und DeepQ-Learning Snake*. Feb 2021. – URL `http://textmining.ful.informatik.haw-hamburg.de/wiki/index.php/Q-_und_DeepQ-Learning_Snake`. – abgerufen am 16.12.2021

[4] Ma, Bowei ; Tang, Meng ; Zhang, Jun: *Exploration of Reinforcement Learning to SNAKE*. 2016

[5] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin: *Playing Atari with Deep Reinforcement Learning*. 2013

[6] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg u. a.: Human-level control through deep reinforcement learning. In: *nature* 518 (2015), Nr. 7540, S. 529–533

[7] Sebastianelli, Alessandro ; Tipaldi, Massimo ; Ullo, Silvia L. ; Glielmo, Luigi: A Deep Q-Learning based approach applied to the Snake game. In: *2021 29th Mediterranean Conference on Control and Automation (MED)*, 2021, S. 348–353

[8] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2018

[9] Tesauro, Gerald: Temporal difference learning and TD-Gammon. In: *Communications of the ACM* 38 (1995), Nr. 3, S. 58–68

[10] Viglietta, Giovanni: *Gaming is a hard job, but someone has to do it!* 2013

Erklärung zur selbstständigen Bearbeitung der Arbeit

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.

Reinbek	06.03.2022	
Ort	Datum	Unterschrift im Original