Carl von Ossietzky Universität Oldenburg

Wirtschaftsinformatik
Bachelorarbeit

Vergleich von verschiedenen Deep Reinforcement Learning Agenten am Beispiel des Videospiels Snake

---

Autor:
Lorenz Mumm

Erstgutachter:
Apl. Prof. Dr. Jürgen Sauer

Zweitgutachter:
M. Sc. Julius Möller

Abteilung Systemanalyse und -optimierung
Department für Informatik

Oldenburg, 8. Juli 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

KI	**K**ünstliche **I**entelligenz
RL	**R**einforcement **L**earning
DQN	**D**eep **Q**-**N**etwork
DQL	**D**eep **Q**-**L**earning
DDQN	**D**ouble **D**eep **Q**-**N**etwork
PPO	**P**roximal **P**olicy **O**ptimization
SAC	**S**oft **A**ctor **C**ritic
A2C	Advantage **A**ctor **C**ritic
Env.	**Env**ironment
Obs.	**Obs**ervation

Kapitel 1

Einleitung

Das Maschine Learning ist weltweit auf dem Vormarsch und große Unternehmen investieren riesige Beträge, um mit KI basierten Lösungen größere Effizienz zu erreichen. Auch der Bereich des Reinforcement Learning gerät dabei immer mehr in das Blickfeld von der Weltöffentlichkeit. Besonders im Gaming Bereich hat das Reinforcement Learning schon beeindruckende Resultate erbringen können, wie z.B. die KI AlphaGO, welche den amtierenden Weltmeister Lee Sedol im Spiel GO besiegt hat Chunxue u. a. (2019). In Anlehnung an die vielen neuen Reinforcement Learning Möglichkeiten, die in der letzten Zeit entwickelt wurden und vor dem Hintergrund der immer größer werdenden Beliebtheit von KI basierten Lösungsstrategien, soll es in dieser Bachelorarbeit darum gehen, einzelnen Reinforcement Learning Agenten, mittels statistischer Methoden, genauer zu untersuchen and den optimalen Agenten für ein entsprechendes Problem zu bestimmen.

1.1 Motivation

In den letzen Jahren erregte das Reinforcement Learning eine immer größere Aufmerksamkeit. Siege über die amtierenden Weltmeister in den Spielen GO oder Schach führten zu einer zumehmenden Beliebtheit des RL. Neue Verfahren, wie z.B. der Deep Q-Network Algorithmus auf dem Jahr 2015 Mnih u. a. (2015), der Proximal Policy Optimization (PPO) aus dem Jahr 2017 Schulman u. a. (2017) oder der Soft Actor Critic (SAC) aus dem Jahr 2018 Haarnoja u. a. (2018), haben ihr Übriges getan, um das RL auch in anderen Bereichen weiter anzusiedeln, wie z.B. der Finanzwelt, im autonomen Fahren, in der Steuerung von Robotern, in der Navigation im Web oder als Chatbot Lapan (2020).
Durch die jedoch große Menge an RL-Verfahren gerät man zunehmend in die problematische Situaion, sich für einen diskreten RL Ansatz zu entscheiden. Weiter erschwert wird dieser Auswahlprozess noch durch die Tatsache, dass die einzelnen Agenten jeweils untereinander große Unterschiede aufweisen. Auch existieren häufig mehrere Ausprägungen eines RL-Verfahrens, wie z.B. der Deep Q-Network Algo-

rithmus (DQN) und der Double Deep Q-Network Algorithmus (DDQN). Die Wahl des passenden Agenten kann großen Einfluss auf die Performance haben Bowei Ma (2016), deshalb soll in dieser Ausarbeitung eine Methodik, welche auf einem Vergleich berut, entwickelt werden, die den passenden Agenten für eine entsprechendes Problem bestimmt.
Aufgrund der Tatsache, dass die Durchführung der erwähnten Methodik an mehrere Umgebungen den Rahmen sprengen würde, wurde sich für ein spezifische Umgebung zum Testen entschieden. Die Wahl fiel dabei auf das Computerspiel Snake. Damit ergib sich die folgende Forschungsfrage:
Wie kann an einem Beispiel des Spiels Snake für eine nicht-triviale geringdimensionale Umgebung ein möglichst optimaler RL Agent ermittelt werden?

Basierend auf der Forschungsfrage ergibt sich ein Mehrwert für die Wissenschaft. Durch die Abnahme des Entscheidungsfindungsprozesses müssen Forscherinnen und Forscher wie auch Anwenderinnen und Anwender von RL-Verfahren nicht mehr unreflektiert irgendeinen RL Agenten auswählen, sondern können auf Grundlage der Methodik und der daraus hervorgehenden Daten den passenden Agenten bestimmen.

Mit der Wahl des Spiels Snake ist zusätzlich zu dem oben erwähnten Mehrwert noch ein weiterer in Erscheinung getreten. So interpretieren neue Forschungsanätze das Spiel Snake als ein dynamisches Pathfinding Problem. Mit dieser Art von Problemen sind auch unbemannte Drohne (UAV Drohnen) konfrontiert, welche beispielsweise Menschen in komplexen Katastrophensituationen, wie z.B. auf explodierten Rohölbohrplattformen oder Raffenerien, finden und retten sollen. Auch kann das Liefen von wichtigen Gütern, beispielsweise medizinischer Natur, in solche Gebiete kann durch die Forschung am Spiel Snake möglich gemacht werden Chunxue u. a. (2019). Somit kann der RL-Vergleich am Spiel Snake möglicherweise helfen, Menschen zu retten.

Kapitel 2

Grundlagen

2.1 Game of Snake

Snake (englisch für Schlange) zählt zu den meist bekannten Computerspielen unserer Zeit. Es zeichnet sich durch sein simples und einfach zuverstehendes Spielprinzip aus. In seiner ursprünglichen Form ist Snake als ein zweidimensionales rechteckiges Feld. Dieses Beschreibe das komplette Spielfeld, in welchem man sich als Snake bewegt. Häufig wird diese als einfacher grüner Punkt (Quadrat) dargestellt. Dieser stellt den Kopf der Snake dar. Neben dem Kopf der Snake befindet sich auf dem Spielfeld auch noch der sogenannte Apfel. Dieser wird häufig als roter Punkt (Quadrat) dargestellt.

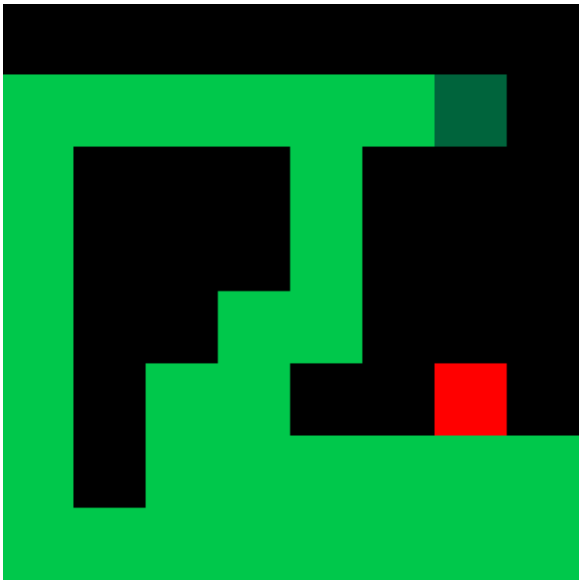

Abbildung 2.1: Game of Snake - Abbildung eines Snake Spiels in welchem der Apfel druch das rote und der Snake Kopf durch das dunkelgrüne Quadrat dargestellt wird. Die hellgrünen Quadrate stellen den Schwanz der Snake dar.

Ziel der Snake ist es nun Äpfel zu fressen. Dies geschied, wenn der Kopf der Snake auf das Feld des Apfels läuft. Danach verschwindet der Apfel und ein neuer

erscheint in einem noch freies Feld). Außerdem wächst, durch das Essen des Apfels, die Snake um ein Schwanzgglied. Diese Glieder folgen dabei in ihren Bewegungen den vorrangegangenen Schwanzgleid bishin zum Kopf. Dem Spieler ist es nur möglich den Kop der Snake zu steuern.
Bisher so weit so gut, doch es existiert noch einen Hacken an der Geschichte. Der Snake ist es nicht erlaubt die Wände oder sich selbst zu berühren, geschied dies, im Laufe des Spiels, trotzdem endet dieses sofort. Diese Einschränkung führt zu einem Ansteigen der Komplexität gegen Ende des Spiels. Ein Spiel gilt als gewonnen, wenn es der Snake gelungen ist, das komplette Spielfeld auszufüllen.

2.2 Reinforcement Learning

Das Reinforcement Learning (Bestärkendes Lernen) ist einer der drei großen Teilbereiche, die das Machine Learning zu bieten hat. Beben dem Reinforcement Learning zählen das supervised Learning (Überwachtes Lernen) und das unsupervised Learning (unüberwachtes Lernen) ebenfalls noch zum Machine Learning.
Einordnen lässt sich das Reinforcement Learning (RL) als Supervised Learning mit fehlenden Labels. Viele Aspekte des (SL), wie z.B. neuronale Netze zum Approximation einer Lösungsfunktion, Gradientenabstiegsverfahren und Backpropergation zur Erlernung von Datenrepräsentationen, werden auch im RL verwendet.
Auf Menschen wirkt das RL, im Vergleich zu den anderen Disziplinen des Machine Learnings, am nachvollziehbarsten. Dies liegt an der Lernstrategie die sich dieses Verfahren zu Nutze macht. Beim RL wird anhand eines “trial-and-error„ Verfarens gelernt. Ein gutes Beispiel für eine solche Art des Lernens ist die Erziehung eines Kindes. Wenn eben dieses Kind etwas gutes tut, dann wird es belohnt. Angetrieben von der Belohnung, versucht das Kind dieses Verhalten fortzusetzen. Entsprechend wird das Kind bestraft, wenn es etwas schlechtes tut. Schlechtes Verhalten kommt weniger häufig zum Vorschein, um Bestrafungen zu vermeiden.
Beim RL funktioniert es genau so. Das ist auch der Grund dafür, dass viele der Aufgaben des RL dem menschlichen Arbeitsspektrum sehr nahe sind. So wird das RL beispielsweise im Finanzhandel eingesetzt. Was früher Börsianer getan haben, erledigt heute der RL Agent. Auch im Bereich der Robotik ist das RL auf dem vormarsch. Wo früher noch komplexe Bewegungsabfolgen eines Roboterarms mühevoll programmiert werden mussten, da können wir heute bereits Roboter mit RL Agenten ausstatten, welche selbstständig die Bewegungsabfolgen meisten Lapan (2020).

2.2.1 Klassifikation von RL-Verfahren

Bevor ein tiefer technischer Einblick in das RL vorgenommen werden sollte, ist es ratsam sich erst einmal mit den einzelnen Klassifikationen des RL zu beschäftigen, um ein tiefergehendes Verständnis für die einzelnen RL Verfahren zu entwickeln. Alle RL Verfahren lassen sich, unter gewissen Gesichtspunkten, in Klassen einordnen, welche Aufschluss über die Implementierung, den Entscheidungsfindungsprozess und die Dateneffizens geben. Natürlich existieren viele Möglichkeiten RL-Verfahren zu klassifizieren aber vorerst beschränken wir uns auf die folgenen drei.

Model-free und Model-based

Die Unterscheidung in model-free (modelfrei) und in model-based (modelbasiert) gibt Aufschluss darüber, ob der Agent fähig ist, ein Modell des Zustandekommens der Belohnungen (Reward) zu entwickeln.
Model-free RL Verfahren sind nicht in der Lage das Zustandekommen der Belohnung vorherzusagen, vielmehr ordnen sie einfach die Beobachtung einer Aktion zu. Es werden keine zukünftigen Beobachtungen und/oder Belohnungen extrapoliert.
Ganz anderes sieht es da bei den model-based RL Verfahren aus. Diese versuchen eine oder mehrere zukünftige Beobachtungen und/oder Belohnungen zu ermitteln, um die beste Aktionsabfolge zu bestimmen. Dem Model-based RL Verfahren liegt also ein planungsprozess der nächsten Züge zugrunde Sutton und Barto (2018).

Beide Verfahrensklassen haben Vor- und Nachteile, so sind model-based Verfahren häufig in deterministischen Environments mit simplen Aufbau und strengen Regeln anfindbar. Die Bestimmung von Observations und/oder Rewards bei größerern Environments. wäre viel zu komplex und zu resourcenbindend.
Model-Free Algorithmen haben dagegen den Vorteil, dass das Trainieren leichter ist, aufgrund des wegfallenden Aufwandes, welcher mit der Bestimmung zukünftiger Observations und/oder Rewards einhergeht. Sie preformen zudem in großen Environments besser als model-based RL Verfahren. Des Weiteren sind model-free RL Verfahren universiell einsetzbar, im Gegensatz zu model-based Verfahren, welche ein Modell des Environment für das Planen benötigen Lapan (2020); Sutton und Barto (2018).

Policy-based und value-based Verfahren

Die Einordnung in Policy- und wertebasierte Verfahren gibt Aufschluss über den Entscheidungsfindungsprozess des Verfahrens. Agenten, welche policy-based arbeiten, versuchen unmittelbar die berechnete Policy umzusetzen und sie zu optimieren. Policy-based RL Verfahren besitzen dafür meist ein eigenes Network (Policy-

Network), welches die Policy bestimmt. Gewöhnlich wird diese als eine Wahrscheinlichkeitsverteilung über alle Actions repräsentiert. Jede Action erhält damit einen Wert zwischen Null und Eins, welcher Aufschluss über die Qualität der Action im momentanen Zustand des Env. liefert.
Anders als bei den policy-based wird bei den value-based Verfahren nicht mit Wahrscheinlichkeiten gearbeitet. Die Policy und damit die Entscheidungsfindung, wird indirekt über das Bestimmen aller Values ermittelt. Es wird daher immer die Action gewählt, welche zum dem State führt, der über den größten Value verfügt Lapan (2020).

On-Policy und Off-Policy Verfahren

Eine Klassifikation in On-Policy und Off-Policy Verfahren hingegen, gibt Aufschluss über die Dateneffizenz. Einfach formuliert sind RL Verfahren, welche Off-Policy sind, in der Lage von älteren Daten zu lernen. Diese können vom Menschen oder von Agenten generiert worden sein. Es spielt keine Rolle, mit welcher Entscheidungsfindungsqualität die Daten erhoben worden sind. Sie können zu Begin, in der Mitte oder zum Ende des Lernprozesses ermittelt worden sein. Die Aktualität der Daten spielt daher keine Rolle für Off-Policy-Verfahren.
On-Policy-Verfahren sind dagegen sehr wohl abhängig von aktuellen Daten, da sie versuchen die Policy indirekt oder direkt zu optimieren.

Auch hier besitzen beide Klassen ihre Vor- und Nachteile. So können beispielsweise Off-Policy Verfahren mit älteren Daten immer noch trainiert werden. Dies macht Off-Policy RL Verfahren dateneffizienter als On-Policy Verfahren. Meist ist es jedoch so, dass diese Art von Verfahren langsamer konvergieren.
On-Policy Verfahren konvergieren dagegen meist schneller. Sie benötigen aber dafür auch mehr Daten aus dem Environment, dessen Beschaffung aufwendig und teuer sein könnte. Die dateneffizienz nimmt ab Lapan (2020).

2.2.2 Vokabular

Um jedoch das RL tiefergehend zu verstehen, ist es erforderlich die gängigen Begrifflichkeiten zu erlernen und deren Bedeutung zu verstehen.

Agent

Im Zusammenhang mit dem RL hört man gewöhnlich von Agenten. Sie sind die zentralen Kapselungsobjekte, welche die eigentlichen Algorithmen, wie z.B. den Algorithmus des Q-Learning oder eines Proximal Policy Optimization, in eine feste Schnittstelle einbinden. Häufig sind auch Methoden in der Schnittstelle vorgegeben,

wie z.B. die choose_action oder die learn Methode. Je nach Implementierung können die Methoden gelegentlich auch andere aber ersichtliche Namen tragen, wie z.B. die choose_action Methode, welche auch häufig unter dem Namen „act“ anzutreffen ist. Bei dem Agenten handelt es sich gewöhnlich um die einzige Instanz, welche mit dem Environment (der Umgebung) interagiert. Zu dieser Interaktionen zählen das Entgegennehmen von Observations und Rewards, wie auch das Tätigen von Actions Lapan (2020).

Environment

Das Environment (Env.) bzw. die Umgebung ist vollständig außerhalb des Agenten angesiedelt. Es spannt das zu manipulierende Umfeld auf, inwelchem der Agent Interaktionen tätigen kann. An ein Environment werden unter anderem verschiedene Ansprüche gestellt, damit ein RL Agent mit ihm in Interaktion treten kann. Zu diesen Ansprüchen gehört unter anderem die Fähigkeit Observations und Rewards zu liefen, Actions zu verarbeiten Lapan (2020). Actions, welche im momentanen State des Env. nicht gestattet sind, müssen entsprechend behandelt werden. Dies wirft die Frage auf, ob es in dem Env. einen terminalen Zustand (häufig done oder terminal genannt) geben soll. Existiert ein solcher terminaler Zustand, so muss eine Routine für den Reset (Neustart) des Env. implementiert sein.

Action

Die Actions bzw. die Aktionen sind einer der drei Datenübermittlungswege. Bei ihnen handelt es sich um Handlungen welche im Env. ausgeführt werden. Actions können z.B. erlaubte Züge in Spielen, das Abbiegen im Autonomenfahren oder das Ausfüllen eines Antragen sein. Es wird ersichtlich, dass die Actions, welche ein RL Agent ausführen kann, prinzipiell nicht in der Komplexität beschrängt sind. Dennoch ist es hier gängige Praxis geworden, dass arbeitsteilig vorgegangen werden soll, da der Agent ansonsten zu viele Ressourcen in Anspruch nehmen würde.
Im RL wird zwischen diskreten und stetigen Aktionsraum unterschieden. Der diskrete Aktionsraum umfassen eine endliche Menge an sich gegenseitig ausschließenden Actions. Beispielhaft dafür wäre das Gehen an einer T-Kreuzung. Der Agent kann entweder Links, Rechts oder zurück gehen. Es ist ihm aber nicht möglich ein bisschen Rechts und viel Links zu gehen. Anders sicht das beim stetigen Aktionsraum aus. Dieser zeichnet sich durch stetige Werte aus. Hier ist das Steuern eines Autos beispielhaft. Um wie viel Grad muss der Agent das Steuer drehen, damit das Fahrzeug auf der Spur bleibt Lapan (2020)?

Observation

Die Oberservation (Obs.) bzw. die Beobachtung ist ein weiterer der drei Datenübermittlungswege, welche den Agenten mit dem Environment verbindet. Mathematisch, handelt es sich bei der Obs. um einen oder mehere Vektoren bzw. Matrizen. Die Obs. hat einen immensen Einfluss auf den Erfolg des Agenten und sollte daher klug gewählt werden. Je nach Anwendungsbereich fällt die Obs. sehr unterschiedlich aus. In der Finanzwelt könnte diese z.B. die neusten Börsenkurse einer oder mehrerer Aktien beinhalten oder in der Welt der Spiele könnten diese die aktuelle erreichte Punktezahl wiedergeben Lapan (2020). Es hat sch als Faustregel herausgestellt, dass man sich bei dem Designing der Obs. auf das wesentliche konzentieren sollte. Unnötige Informationen können den die Effizenz des Lernen mindern und den Resourcenverbrauch zudem steigen lassen.

Reward

Der Reward bzw. die Belohnung ist der letzte Datenübertragungsweg. Er ist neben der Action und der Obs. eines der wichtigsten Elemente des RL und von besonderer Bedeutung für den Lernerfolg. Bei dem Reward handelt es sich um eine einfache skalare Zahl, welche vom Env. übermittelt wird. Sie gibt an, wie gut oder schelcht eine ausgeführte Action im Env. war Lapan (2020). Um eine solche Einschätzung zu tätigen, ist es nötig eine Bewertungsfunktion zu Implementieren, welche den Reward bestimmt. Bei der Modellierung des Rewards kommt es voralledem darauf an, in welchen Zeitabständen dieser an den Agenten gesendet wird (sekündlich, minütlich, nur ein Mal). Aus Bequemlichkeitsgründen ist es jedoch gängige Praxis geworden, dass der Reward in fest definierten Zeitabständen erhoben und üermittelt wird Lapan (2020). Je nach Implementierung hat dies große Auswirkungen auf das zu erwartende Lernverhalten.

State

Der State bzw. der Zustand ist eine Widerspieglung der zum Zeitpunkt t vorherschenden Situation im Environments. Der State wird von der Obs. repräsentiert. Häufig findet der Begriff des States in diversen Implementierungen, wie auch in vielen Ausarbeitung zum Themengebiet des RL Anwendung Sutton und Barto (2018).

2.2.3 Funktionsweise

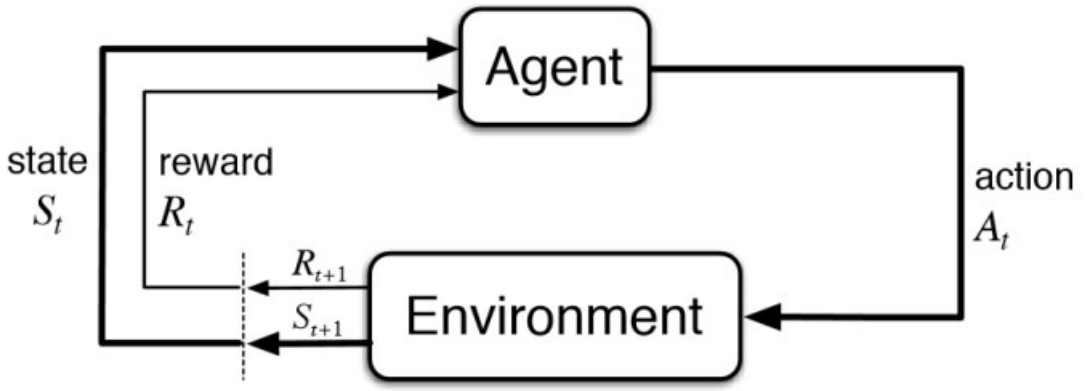

Abbildung 2.2: Reinforcement Learning schematische Funktionsweise - Der Agent erhält einen State S_t und falls t != 0 einen Reward R_t. Daraufhin wird vom Agenten eine Action A_t ermittelt, welche im Environment ausgeführt wird. Das Environment übermittelt den neu enstandenen State S_{t+1} und Reward R_{t+1} an den Agenten. Diese Prozedue wird wiederholt Sutton und Barto (2018).

Zu Beginn wird dem Agenten vom Environment der initialer State übermittelt. Auf Grundlage dieses Stat S_t wobei $t = 0$ ist, welcher inhaltlich aus der zuvor besprochenden Obs. 2.2.2 besteht, wird im Agenten ein Entscheidungsfindungsprozess angestoßen. Es wird eine Action A_t ermittelt, welcher der Agent an das Environment weiterleitet. Die vom Agenten ausgewählte Action A_t wird num im Env. ausgeführt. Dabei kann der Agent selbstständig das Env. manipuliert oder er kann die Action an das Env. weiterleitet. Es kümmert sich dann selbst um die Durchführung der Action.
Das manipulierte Environment befindet sich nun im neuen State S_{t+1}, welcher an den Agenten weitergeleitet wird. Des Weiteren wird noch einen Reward R_{t+1}, welcher vom Env. bestimmt wurde, an den Agenten übermittelt.
Mit dem neuen State S_{t+1}, kann der Agent wieder eine Action A_{t+1} bestimmen, die ausgeführt wird. Daraufhin werden wieder der neue State S_{t+2} und Reward R_{t+2} ermittelt und übertragen usw. Der Zyklus beginnt von neuem Sutton und Barto (2018).

2.3 Proximal Policy Optimization

Der Proximal Policy Optimization Algorithmus oder auch PPO abgekürzt wurde von den Open-AI-Team entwickelt. Im Jahr 2017 erschien das gleichnamige Paper, welches von John Schulman et al. veröffentlicht wurde. In diesem werden die besonderen Funktionsweise genauer erläutert wird Schulman u. a. (2017).

2.3.1 Actor-Critic Modell

Der PPO Algorithmus ist ein policy-based RL-Verfahren, welches, im Vergleich mit anderen gleichen Verfahren, einige Verbesserungen aufweist. Er ist eine Weiterentwicklung des Actor-Critic-Verfahrens und basiert intern auf zwei NN Lapan (2020), dem sogenanntes Actor-Network bzw. Policy-Network und das Critic-Network bzw. Value-Network. Beide NN können aus mehreren Schichten bestehen, jedoch sind Actor und Critic streng von einander getrennt und teilen keine gemeinsamen Parameter. Gelegentlich werden den beiden Netzen (Actor bzw. Critic) noch ein weiteres Netz vorgeschoben. In diesem Fall können Actor und Critic gemeinsame Parameter besitzen. Das Actor- bzw. Policy-Network ist für die Bestimmung der Policy zuständig. Anders als bei Value-based RL-Verfahren wird diese direkt bestimmt und kann auch direkt angepasst werden. Die Policy wird als eine Wahrscheinlichkeitsverteiling über alle möglichen Actions vom Actor-NN zurückgegeben Lapan (2020).
Das Critic- bzw. Value-Network evaluiert die Actions, welche vom Actor-Network bestimmt worden sind. Genauer gesagt, schätzt das Value-Network die sogenannte "Discounted Sum of Rewards" zu einem Zeitpunkt t, basierend auf dem momentanen State s, der dem Value-Network als Input dient. "Discounted Sum of Rewards" wird im späteren Verlauf noch weiter vorgestellt und erklärt.

2.3.2 PPO Training Objective Function

Nun da einige Grundlagen näher beleuchtet worden sind, ist das nächste Ziel die dem PPO zugrunde liegende mathematische Funktion zu verstehen, um im späteren eine eigene Implementierung des PPO durchführen zu können und um einen objektiveren Vergleich der zwei RL-Verfahren durchführen zukönnen.
Der PPO basiert auf den folgenden matematischen Formel, welche den Loss eines Updates bestimmt:

$$L_t^{\mathrm{PPO}}(\theta) = L_t^{\mathrm{CLIP\ +\ VF\ +\ S}}(\theta) = \hat{\mathbb{E}}_t[L_t^{\mathrm{CLIP}}(\theta) - c_1 L_t^{\mathrm{VF}} + c_2 S[\pi_\theta](s_t)] \tag{2.1}$$

Dabei besteht die Loss-Function aus drei unterschiedlichen Teilen. Zum einen aus dem Actor-Loss bzw. Policy-Loss bzw. Main Objective Function $L_t^{\mathrm{CLIP}}(\theta)$, zum anderen aus dem Critic-Loss bzw. Value-Loss L_t^{VF} und aus dem Entropy Bonus $S[\pi_\theta](s_t)$. Die Main Objective Function sei dabei durch folgenden Term gegeben Schulman u. a. (2017).

$$L_t^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t(s,a), \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t(s,a))] \tag{2.2}$$

Formelelemente

Um die dem PPO zugrundeliegende Update Methode besser zu verstehen folge eine Erklärung ihrer einzelnen mathematischen Elemente. Die einzelnen Erklärungen basieren auf den PPO Paper Schulman u. a. (2017).

Tabelle 2.1: Formelelemente

Symbol	Erklärung
θ	Theta beschreibt die Paramenter aus denen sich die Policy des PPO ergibt. Sie sind die Gewichte, welche das Policy-NN definiert.
π_θ	Die Policy bzw. Entscheidungsfindungsregeln sind eine Wahrscheinlichkeitsverteilung über alle möglichen Actions. Sutton und Barto (2018)
$L^{\text{CLIP}}(\theta)$	$L^{\text{CLIP}}(\theta)$ bezeichnetz den sogenannten Policy Loss, welche in Abhängigkeit zu der Policy π_θ steht. Dabei handelt es sich um einen Zahlenwert, welcher den Fehler über alle Parameter approzimiert. Dieser wird für das Lernen des Netzes benötigt.
t	Zeitpunkt
$\hat{\mathbb{E}}_t$	$\hat{\mathbb{E}}_t$ gibt an, dass es sich um einen Erwartung handelt, welche einem empirischen Durchschnitt zugrundeliegt.
$r_t(\theta)$	Quote zwischen alter Policy (nicht als Abhängigkeit angegeben, da sie nicht verändert werden kann) und aktueller Policy zum Zeitpunkt t. Daher auch Probability Ratio genannt.
$\hat{A}_t(s, a)$	erwarteter Vorteil bzw. Nachteil einer Action a, welche im State s ausgeführt wurde. welcher sich in Abhängigkeit von dem State s und der Action a befindet.
clip	Mathematische Funktion zum Beschneidung eines Eingabewertes. Clip setzt eine Ober- und Untergrenze fest. Sollte ein Wert der dieser Funktion übergeben wird sich nicht mehr in diesen Grenzen befinden, so wird der jeweilige Grenzwert zurückgegeben.
ϵ	Epsilon ist ein Hyperparameter, welcher die Ober- und Untergrenze der Clip Funktion festlegt. Gewöhnlich wird für ϵ ein Wert zwischen 0.1 und 0.2 gewählt.

γ	Gamma bzw. Abzinsfaktor ist ein Hyperparameter, der die Zeitpräferenz des Agenten kontrolliert. Gewöhnlich liegt Gamma γ zwischen 0.9 bis 0.99. Große Werte sorgen für ein weitsichtiges Lernen des Agenten wohingegen ein kleine Werte zu einem kurzfristigen Lernen führen Sutton und Barto (2018).

Advantage

Der erste behandelt den Advantage $\hat{A}_t(s, a)$. Dieser wird durch die Subtraktion der Discounted Sum of Rewards bzw. des Return R_t und dem Baseline Estimate $b(s_t)$ bzw. der Value function berechnet Mnih u. a. (2016); Schulman u. a. (2017).

$$\hat{A}_t(s, a) = R_t - b(s_t) \tag{2.3}$$

Der Advantage gibt ein Mass an, um wie viel besser oder schlechter eine Action war, basierend auf der Erwartung der Value function bzw. des Critics. Es wird also die Frage beantwortet, ob eine gewählte Action a im State s_t zum Zeitpunkt t besser oder schlechter als erwartet war Mnih u. a. (2016).

Return

Der Return R_t stellt dabei die Summe der Rewards in der gesamten Spieleepisode von dem Zeitpunkt t an dar. Diese kann ermittelt werden, da alle Rewards, durch das Sammeln von Daten, bereits bekannt sind. Des Weiteren werden die einzelnen Summanden mit einem Discount Factor γ multipliziert, um die Zeitpräferenz des Agenten besser zu steuern. Gamma liegt dabei gewöhnlich zwischen einem Wert von 0.9 bis 0.99. Kleine Werte für Gamma sorgen dafür, dass der Agent langfristig eher dazu tendiert Actions bzw. Aktionen zu wählen, welche unmittelbar zu positiven Reward führen. Entsprechend verhält es sich mit großen Werten für Gamma Sutton und Barto (2018).

Baseline Estimate

Der Baseline Estimate $b(s_t)$ oder auch die Value function ist eine Funktion, welche durch ein NN realisiert wird. Es handelt sich dabei um den Critic der Actor-Critic-Verfahrens. Die Value function versucht eine Schätzung des zu erwartenden Discounted Rewards R_t bzw. des Returns, vom aktuellen State s_t, zu bestimmen. Da es sich hierbei um die Ausgabe eines NN handelt, wird in der Ausgaben immer eine Varianz bzw. Rauschen vorhanden sein Mnih u. a. (2016).

Probability Ratio

Die Probability Ratio $r_t(\theta)$ ist der nächste Baustein zur Vervollständigung der PPO Main Objective Function.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2.4}$$

In normalen Policy Gradient Methoden bestehe ein Problem zwischen der effizienten Datennutzung und dem Updaten der Policy. Dieses Problem tritt z.B. im Zusammenhang mit dem Advantage Actor Critic (A2C) Aglorithmus auf und reklementiert das effiziente Sammeln von Daten. So ist des dem A2C nur möglich von Daten zum lernen, welche on-policy (unter der momentanen Policy) erzeugt wurden. Das Verwenden von Daten, welche unter einer älteren aber dennoch ähnlichen Policy gesammelt wurden, ist daher nicht zu empfehlen.
Der PPO bedient sich jedoch eines Tricks der Statistik, dem Importance-Sampling (IS, deutsch: Stichprobenentnahme nach Wichtigkeit). Unter der Zuhilfenahme des IS ist es dem PPO nun möglich, auch Daten zu nutzen, welche unter ähnlichen älteren Policy gesammelt wurden. Dies eröffnet die Möglichkeit der Implementierung eines Replay-Buffers bzw. Memorys. Des Weiteren können nun generierte Daten mehrfach für Updates der Policy genutzt werden, was die Menge an Daten, welche nötig ist, um ein gewisses Ergebnis zu erreichen, stark minimiert. Der PPO Algorithmus ist durch die Umstellung auf die Probability Ratio effizienter in der Nutzung von Daten geworden.

Surrogate Objectives

Der Name Surrogate (Ersatz) Objective Function ergibt sich aus der Tatsache, dass die Policy Gradient Objective Function des PPO nicht mit der logarithmierten Policy $\log_{\pi_\theta}(a_t|s_t)\hat{A}_t$ arbeitet, wie es die normale Policy Gradient Methode vorsieht, sondern mit dem Surrogate der Probability Ratio $r_t(\theta)$ 2.4 zwischen alter und neuer Policy. Intern beruht der PPO auf zwei sehr ähnlichen Objective Functions, wobei die erste dieser beiden

$$r_t(\theta)\hat{A}_t(s,a) \tag{2.5}$$

der normalen TRPO Objective Function entspricht, ohne die, durch den TRPO vorgesehende, KL-Penalty Schulman u. a. (2015, 2017). Die alleinige Nutzung dieser Objective Function hätte jedoch destruktiv große Policy Updates zufolge. Aus diesem Grund haben John Schulman et al. eine zweite Surrogate Objective Function,

dem PPO Verfahren hinzugefügt.

$$\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t(s,a) \tag{2.6}$$

Die einzige Veränderung im Vergleich zur ersten Objective Function 2.5 ist, dass eine Abgrenzung durch die Clip-Funktion eintritt. Sollte sich die Probability Ratio zuweit von 1 entfernen, so wird $r_t(\theta)$ entsprechende auf $1-\epsilon$ bzw. $1+\epsilon$ begrenzt. Das hat zufolge, dass der Loss $L_t^{\text{CLIP}}(\theta)$ ebenfalls begrenzt wird, sodass es zu keinen großen Policy Updates kommen kann. Es wird sich daher in einer Trust Region bewegt, da man sich nie allzu weit von der ursprünglichen Policy entfernt Schulman u. a. (2015, 2017).

Zusammenfassung der PPO Training Objective Function

$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t(s,a), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t(s,a))]$ Insgesamt ist der Actor-Loss ein Erwartungswert, welcher sich empirisch über eine Menge an gesammelten Daten ergibt. Dies wird durch $\hat{\mathbb{E}}_t$ impleziert. Dieser setzt sich aus vielen einzelnen Losses zusammen. Diese einzelnen Losses sind das Minimum der beiden Surrogate Objective Functions zusammen. Dies sorgt dafür, dass keine zu großen Losses die Policy zuweit von dem Entstehungspunkt der Daten wegführen (Trust Region) Schulman u. a. (2017).

Des Weiteren wird für den PPO Training Loss auch noch der Value-Loss benötigt. Dieser setzt sich folgendermaßen zusammen:

$$L_t^{\text{VF}} = (V_\theta(s_t) - V_t^{targ})^2 \text{ wobei } V_t^{targ} = r_t(\theta) \tag{2.7}$$

Der letzte Teil, welcher für die Bestimmung des PPO Training Loss benötigt wird, ist der Entropy Bonus. Dabei handelt es sich um die Entropie der Policy. Diese lässt sich normal bestimmen.
Es ergibt sich damit der bereits oben erwähnte 2.1 PPO Training Loss Schulman u. a. (2017):

$$L_t^{\text{PPO}}(\theta) = L_t^{\text{CLIP + VF + S}}(\theta) = \hat{\mathbb{E}}_t[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}} + c_2 S[\pi_\theta](s_t)]$$

2.4 Deep Q-Learning

Das Q-Learning ist eine weitere Art des RL, welches eine gesamte Familie an RL-Verfahrewn umfasst. Namensgebend für die Familie des Q-Learnings sind jedoch die sogenannten Q-Values, welche die Aktionswert einer Action a im State s angeben.

Deutlich wichtiger im weiteren Verlauf des Deep Q-Learnings sind jedoch die Zustandswerte, welche von der Zustandswertefunktion $V(s)$ ermittelt werden. Bei ihr handelt es sich, bezogen auf das Deep Q-Learning um eine NN, dass im Verlauf triniert wird.
Um das Q-Learning auch hier vollständig zu verstehen und um einen Eindruck des zu erwartenden Implementierungsaufwandes zu erhalten, ist es nötig die dem DQN zugrundeliegenden Formeln zu verstehen. Angestammtes Ziel des Q-Learnings ist es für alle State-Action-Paare (s, a) ein Q zu ermitteln Lapan (2020). Daher gilt:

$$Q(s,a) = \mathbb{E}_{s' \sim s}[r(s,a) + \gamma V(s')] = \sum_{s' \in S} p_{a,s \to s'}(r(s,a) + \gamma V(s')) \tag{2.8}$$

Der Aktionswert für den State s und die Action a setzt sich auf dem zu erwartenden unmittelbaren Reward $r(s, a)$ und der diskontierten (abgezinsten) langfristige Reward des Folgezustandes s' zusammen. Dabei ist γ der Abzinsfaktor und der langfristige Reward des Folgezustand wird durch $V(s')$ dargestellt.
Diese Aktionswerte $Q(s, a)$ können nun herangezogen werden um das Value-NN zu trainieren, da der Value V eines State s durch den maximalen Aktionswert, über alle Actions, definiert ist Lapan (2020).

$$V(s) = \max_{a \in A} Q(s,a) \tag{2.9}$$

Der Value eines States ist daher der maximale Aktionswert desselbigen. Das bedeutet, dass der Aktionswert eines beliebigen State dem Wert entspricht, welcher sich mit der besten Action des States erzielen lässt.
Problematisch ist jedoch die Bestimmung der Q-Values, da sich 2.8 schlecht in High-level-Languages wie Python, Java oder C++ implementieren lässt. In der Praxis wird daher ein rekursives Vorgehen dazu verwendet, um die Q-Values, über den Trainingsverlauf, zu erlernen Lapan (2020).

$$Q(s,a) = r(s,a) + \gamma \max_{a \in A} Q(s', a') \tag{2.10}$$

Bei der letzen Formel handelt es sich auch um die durchzuführende Prozedue, um das Value-NN zu trainieren Lapan (2020). Da es sich beim Q-Learning um value-based RL-Verfahren 2.2.1 handelt, muss kein weiteres NN aufgesetzt werden, wie es beispielsweise beim PPO der Fall ist. Im Vergleich zum voher vorgestellten PPO ist der DQL Algorithmus ein deutlich simpleres Verfahren. Die Tatsache, dass es sich bei dem DQL auch noch um eine off-policy Verfahren handelt, lässt unter anderem auch das Lernen von älteren Daten zu. Im Vergleich zum PPO ist der DQL-Agent dazu jedoch, in einem deutlich größeren Maß, imstande. Auch sehr weit zurückliegende

Daten können vom DQL-Algorithmus zum Lernen verwendet werden. Dies bedingt einige Details, die in der Implmentierung zum Tragen kommen.

2.5 Backpropagation und das Gradientenverfahren

Nachdem nun alle Agenten-Klassen vorgestellt sind, man sich vielleicht der eine oder andere Leser frage, wie denn nun das eigentliche Lernen vonstattengeht. Die dem Lernen zugrunde liegenden Verfahren sind das Backpropagation und das Gradient descent. Dabei wird häufig fälschlicherweise angenommen, dass sich hinter dem Begriff Backpropagation der komplette Lernprozess verbirgt. Dam ist jedoch nicht so. Der Backpropagation-Algorithmus oder auch häufig einfach nur Backprop genannt, ist der Algorithmus, welchr zuständig für die Bestimmung der Gradienten in einer Funktion. Häufig wird ebenfalls angenommen, dass Backprop nur für NN anwendbar sind, den ist jedoch nicht so. Prinzipiell können mit dem Backprob-Algorithmus die Gradienten einer jeden Funktion bestimmt werden, egal ob NN oder eine Aktivierungsfunktion, wie z.B. Sigmoid oder Tanh (Goodfellow u. a., 2018, S. 90ff.).

Das Gradientenverfahren oder im Englischen auch Gradient descent genannt, wird dafür eingesetzt um die eigentliche Optimierung des NN durchzuführen. Dafür werden jedoch die Gradienten benötigt, welche im Vorhinein durch den Backprop-Algorithmus bestimmt wurden. Jedes NN definiert je nach den Gewichten des NN eine mathematische Funktion. Diese steht in Abhängigkeit von den Inputs und berechnet auf deren Basis die Outputs bzw. Ergebnisse. Basierende auf dieser Funktion lässt sich eine zweite Funktion definieren, die Loss Function oder Kostenfunktion oder Verlustfunktion usw. Diese gibt den Fehler wieder und soll im Optimierungsverlauf minimiert werden, um optimale Ergebnisse zu erhalten. Diese Fehlerfuntion zu minimieren müssen die Gewichte des NN soweit angepasst werden, der die Fehlerfunktion geringe Werte ausgibt. ist dise für alle Daten, mit welchen das NN jemals konfrontiert wird geschafft, so ist das NN perfekt angepasst (Goodfellow u. a., 2018, S. 225ff.).

Ein näheres Eingehen auf die Bestimmung der Gradienten im Rahmen des Backpropagation-Algorithmus und auf die Anpassung der Gewicht im Rahmen des Gradientenverfarens wird der Übersichtlichkeit enfallen. Des Weiteren machen moderne Framework wie Facebooks PyTorch, Googles Tensorflow oder Microsofts CNTK das detalierte Wissen um dise Verfahren für anwendungsgetriebene Benutzer obsolet.

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel soll es thematisch über den momentanen Stand der bereits durchgeführten Forschung gehen. Dabei sollen drei ausgewählte Arbeiten vorgestellt und diskutiert werden.

3.1 Autonomous Agents in Snake Game via Deep Reinforcement Learning

In der Arbeit Autonomous Agents in Snake Game via Deep Reinforcement Learning wurden mehrere Optimierungen an einem DQN Agenten durchgeführt, um mit diesen eine größere Performance beim Sammeln von Äpfeln im Spiel Snake zu erzielen. Die Arbeitet wurde von Zhepei Wei et al. an verschiedenen Universitäten und Forschungsinstituten, wie z.B. College of Computer Science and Technology Jilin University (Changchun, China), Science and Engineering Nanyang Technological University (Singapore) verfasst und im Jahr 2018 veröffentlicht.

3.1.1 Vorstellung

Thematisch werden zwei Optimierungen in diesem Paper vorgestellt, welche auf einen Baseline DQN (Referenz DQN) angewendet worden sind.

- optimierte Reward-Funktion
- Dual Experience Replay Method

In der optimierten Rewardfunktion kommen drei Faktoren zum tragen. Der erste Faktor ist die Distanz. Um so weiter sich die Snake von dem Apfel entfernt, um so größer (negativ) wird der Reward. Nährt sich hingegen die Snake dem Apfen an, so wird der Reward immer größer (positiv).
Der zweite Faktor der Reward-Funktion ist der sogenannte Training Gap. Dieser stellt Erfahrungen dar, welche im Verlauf des Spiels nicht erlernt werden soll. Diese

Erfahrungen, welche direkt nach dem Essen eines Apfels generiert wurden, werden nicht zum lernen herangezogen. Der Agent würde durch diese Erfahrungen lernen, wie das Spiel die Äpfel zufallsbasiert neu verteilt, was nicht dem Trainingsziel entspricht. Durch das Entfernen dieser Erfahrungen wird das Pathfinding des Agenten verbessert.
Der dritte Faktor der Reward-Funktion ist die Timeout Strategy. Diese sorgt für eine Bestrafung, wenn der Agent über eine gewissen Anzahl an Schritten P keinen Apfel gefressen hat. Dies dient dazu den Agenten so zu trainieren, dass er den möglichst optimalsten Weg findet und um etwaige DeadLocks, wie das Gehen im Kreis, zu verhindern. Dabei werden die Rewards der letzen P Schritte (Erfahrungen) mit einem Malus belegt, der sich nach der Länge der Snake richtet. Insgesamt setzt sich die Reward-Funktion nun folgendermaßen zusammen:
Autonormal gilt der Distance Reward. Sollte sich das Spiel im Training Gap befinden, so wird der Reward des Training Gap gewählt. Überschreitet der Agent jedoch die Granze von P Schritten ohne einen Apfel gefressen zu haben, wird der Timeout Strategy Reward Malus auf die letzen P Schritte aufaddiert.

Die Dual Experience Replay Method ist die zweite vorgestellt Optimierung dieses Papers. Hierbei wird der Experience Replay Buffer in zwei Hälften geteilt. in MP_1 und MP_2. Diese beiden Buffer speichern beide die Erfahrungen aus den gespielten Spielen. Der Unterschied zwischen beiden besteht jedoch darin, dass in MP_1 nur reward-basiert gute Erfahrungen ($r > 0.5$) gespeichert werden. Alle Erfahrungen mit Rewards $r < 0.5$ werden in MP_2 gespeichert. Ähnlich der ϵ-greedy Strategie, wird das Verhältnis zwischen den Erfahrungen von MP_1 und MP_2 mit einem Linearen Zerfall gesteuert. Zu Beginn wird werden durchschnittlich 80% Erfahrungen aus MP_1 und 20% aus MP_2 gewählt. Durch stetiges decrementieren fällt dieses Verhältnis gegen Zerfallsende auf 50% MP_1 und 50% MP_2.
Durch diese Aufteilung soll ein schnellerer Lernerfolg gerade zu Beginn erzielt werden.

3.1.2 Diskussion

Die vorliegende Arbeit besitzt bereits viele gute Optimierungsstrategien, welche auf in dieser Bachelorarbeit Verwendung finden sollen. So ist die Idee der Anpassung der Reward-Funktion auf im Vorfeld der Bearbeitung der Verwandten Arbeiten als möglicher Optimierungsansatz infrage gekommen. Jedoch sehe ich davon ab, die Reward-Funktion genau so zu verwenden. Besonders der Timeout Strategy Reward Malus stellt ergibt keinen Sinn, da das Snake Env. so konzipiert wurde, dass die Spiel endet, sofern die Snake in P Schritten keine Apfel frisst.

Dennoch ist der Ansatz der distanzbasierten Reward-Funktion sowie die Einbeziehung der Training Gap wertvolle Erweiterungen, welche in die Optimierungsphase mit eingebunden werden. Ähnlich verhält es sich mit der Dual Experience Replay Method.
Kritisch hingegen ist die Auswahl des Agenten zu betrachten. Das paper bleibt einer Begründung für die Nutzung des DQN-Agenten, welchen die Verfasser selbst erstellt haben, schuldig. Auch wurde nicht tiefergehend auf die Wahl der Netzsturktur, wie auch auf die Observation, eingegangen. Vielmehr erklärt das Paper nur welche Komponenten das Netzwerk besitzt, mit einer kurzen Erwähnung der Funktion der Komponente. Bei der Observation setzen die Verfasser auf ein 240x240 Pixel große Screenshots der Spieloberfälche. Diese Method sorgt zwar für eine vollständige Erfassung des Spielgeschehens seitens des Agenten, jedoch auch für eine große rechnerische Belastung. Es ist zu beführchten, dass das Lernen deutlichlagsamen wird. Des weitern könnte durch den Einsatz von Convolutionalen Layern wichtige Informationen verloren gehen. Verstärkt wird dieser Verdacht zudem nochdurch die tatsache, dass die Verfasser Convolutionalen Layer mit großen Filtern und Strides verwenden.
Diese Tatsachen sollen bei dem Design der netzstruktur beachtet werden, damit kein Informationsverlust eintreten kann. Ein begründete Wahl der Netzsturktur sowie eine auf diese netzstruktur angepasste Observation sollen die Folge dieser Kritik sein. Abschließend werden solgende Elemente aus diesem Paper in dieser Bachelorarbeit Verwendung oder Beachtung finden:

- optimierte Reward-Funktion
- Dual Experience Replay Method
- differenzierte Wahl der Netzsturktur
- differenzierte Wahl der Observation

3.2 Deep Reinforcement Learning for Snake

Die Arbeit Deep Reinforcement Learning for Snake wurde von Anton Finnson und Victor Molnö verfasst. Auf der inhaltlichen Ebene beschäftigt sich das Paper mit dem Verfahren, der Optimierung von Q-Learning Agenten durch successives anpassen einzelner Hyperparameter.
Nachdem zunächst ein leichteres Spiel herangezogen wurde, um die Agenten zu testen, haben die Autoren mit dem Spiel Snake begonnen. Aufgrund schlechter Leistungen wurde der Hauptfokus auf einen DDQN gelegt. Dieser wurde zuerst mit einen fest vordefinierten Set an Hyperparametern getestet.

Ziel des Vorgehens ist es einen Agenten mit den optimalsten Hyperparametern zu bestimmen. Dafür wurden eine Reihe an Standardparametern festgelegt, welche als Basis für die folgenden einzelnen Anpassungen dienen. Basierend auf den Basisparametern wird nun immer ein Parameter verändert. Daraunter fallen ϵ_{min} 2.4 die Netzsturktur ϵ_{dec} der Replay Memory Buffer usw. Am Ende wurden die besten besten Parameter genutzt einen optimierten Agenten zu definieren, welche eine bessere Leistung als der Agent vorwies als der Standard Ageten.

3.3 UAV Autonomous Target Search Based onDeep Reinforcement Learning in Complex Disaster Scene

Kapitel 4

Anforderungen

In dem letzen Kapitel 2, wurden die Grundlagen für den weiteren Vergleich, der Deep Reinforcement Learning Algorithmen, gelegt. Jedoch ist ein solcher Vergleich eine sehr umpfangreiche Angelegenheit, welche, ohne Eingrenzung, eine große Menge an Arbeit und Resourcen binden würde. Daher wurde entschieden, dass der Vergleich exemplarisch an zwei RL Agenten unterschiedlicher Gattungen durchgeführt werden soll. Dies erlaubt eine Abstraktion des Vorgehens auch für andere Agenten.
Problematisch an einem solchen Vergleich ist die Objektivität. Wie kann also sicher gestellt werden, dass die Ergebnisse repräsentativ sind?
Die Antwort auf diese Frage sind feste Anforderungen bzw. Rahmenbedingungen, die bei den einzelnen, am Vergleich beteiligten, Teilen gelten müssen. Dabei gibt es verschiedene Anforderungen an unterschiedliche Systembereiche. In dieser Ausarbeitung existieren es vier Anforderungsbereiche, welche Anforderungen an die folgenden Bereiche stellen:

- Spielimplementierung
- Agenten
- statistische Datenerhebung
- Evaluation

4.1 Anforderungen an das Environment

Die Anforderungen, welche an eine Reinforcement Learning Environment gestellt werden, sind vielseitiger Natur. Sie stammen unter anderem aus der Softwaretechnik, aus dem RL und aus dem gesundem Menschenverstand.

4.1.1 Abtrennung

Bereits aus softwaretechnischen Gesichtspunkten ist die Abtrennung von unabhängigen System ein wichtiges Prinzip. Das Reinforcement Learning ist davon nicht ausge-

nommen. Das Environment, welches das Spiel Snake aufspannt, muss strickt von anderen Teilen der Softwarearschitektur getrennt werden. Dies begünstigt die Modularität und erleichtert nicht nur die Wartbarkeit, sondern auch die Anpassbarkeit des Systems auf Veränderungen, wie z.B. eine andere Reward-Funktion oder Observation.

4.1.2 Kommunikation

Damit keine vollkommende Abtrennung des Environments jegliches Interagieren unmöglich macht, sollen nach dem Vorbild aus 2.2.2 drei Kommunikationskanäle implementiert werden. Das Env. soll in der Lage sein, Actions zu empfangen und Observations und Rewards zu senden.

4.1.3 Funktionalität

Die Funktionalität des Environments ist durch das Implementieren der folgenden Methoden herzustellen.

- step
- reset
- render

Wobei die step Methode für die Ausführung, der vom Agent ausgewählten Action, zuständig ist, die reset Methode den momentaten Env. Fortschritt zurücksetzt und render Methode für das visuelle darstellen zuständig ist. Näheres dazu in 4.1.4.

4.1.4 Visualisierung

Um den Spielfortschritt besser evaluieren zu können und aus Gründen der Ästhetik soll eine graphische Oberfläche implementiert werden. Diese soll das Spielgeschehen graphisch wiedergeben.

4.1.5 Effizienz

Da die in 4.1.3 angesprochenen Methoden sehr häufig aufgerufen werden ist es intelligent, diese so effizient wie möglich, in bezug auf die durchzuführenden Operationen der Methoden, zu implementieren. Dies sollte zu schnelleren Lernerfolgen führen. Insgesamt solle die Spielumgebung so effizient wie möglich aufgesetzt werden.

4.1.6 Test

Um zu garantieren, dass das Env. voll funktionsfähig ist sollen dieses mit Tests überprüft werden. Dies dient der Sicherheit, da Fehler im Env. zu Fehlern im Entscheidungsfindungsprozess führen können, wie das Hide and Seek Environment von OpenAI zeigt Baker u. a. (2019).

4.2 Anforderungen an die Agenten

Neben dem Environment gelten für die Agenten ebenfalls spezielle Anforderungen, welche zum einem objektiven Vergleich führen sollen. Auch sollen Anforderungen an die Implementierung gestellt werden, um Standards einfzuühren.

4.2.1 Abtrennung

Auch beim Agenten spielt die Abtrennung vom restlichen System eine große Rolle. Es erleichtert den umgang mit eventuellen anderen Agenten. Ebenfalls wird dadurch die Wartbarkeit verbessert, da Fehler genauer lokalisiert werden können.

4.2.2 Funktionalität

Reinforcement Learning Agenten können ein großes Repertoire besitzen. Zu den wichtigsten und für die Funktionalität am maßgiblichsten sind die Methoden:

- act
- lern

Durch das Aufrufen der act Methode teilweise auch choose_action Methode genannt, wird der Entscheidungsfinddungsprozess angestoßen. Dieser kann je nach der Art des Agenten durch ein neuronales Netzwerk oder durch andere Methoden, wie z.B. durch das zufällige Wählen von einer actions, erfolgen.
Die lern Methode ist für den Lernprozess des Agenten zuständig.

4.2.3 Effizienz

Durch die Tatsache das sowohl die act als auch die lern Methode sehr häufig in einem Trainingsverlauf aufgerufen werden, ist es von Vorteil, diese, wie natürlich auch alle anderen Bestandteile des Agenten, sorgfälltig und so effizient wie möglich zu implementieren, um unnötigen Resourcenverbrauch zu verhindern.

4.2.4 Einzigartigkeit

Um den Vergleich der verschiedenen Systeme mit dem größtmöglichen Maß an objektivität zu vollziehen, ist es wichtig, dass die Einzigartigkeit eines Agenten nicht alleine durch die Art (PPO oder DQN) definiert wird, sondern auch durch die dem Agenten zugehörigen Parameter.

4.2.5 Test

Zur Prüfung der Funktionalität der einzelnen Agenten, sollen Tests implementiert werden, welche im besonderen das Lernen und die Aktionsbestimmung abdecken sollen. Dies dient zum Ausschluss von Fehlern.

4.3 Anforderungen an die Datenerhebung

Auch die statistische Datenerhebung soll durch Anforderungen konkretisiert und definiert werden. Um die Objektivität sicherzustellen, werden dafür Anforderungen bezüglich der Durchführung, der Datenspeicherung und den Spezifikationen der Datenerhebung erhoben.

4.3.1 Mehrfache Datenerhebung

Um die Valität der zu erzeugenden Trainingsdaten zu grarantieren, soll für jeden Agenten die Datenerhebung dreimalig durchgeführt werden. Dies trägt nicht nur zur Objektivität bei, sondern sichert das Ergebnis ebenfalls auch vor Schwankungen ab, welche beim Reinforcement Learning natürlicherweise auftreten, bedingt durch die zufälligen Spielverläufe des Environment.

4.3.2 Datenspeicherung

Damit ausden erzeugten Daten statistische Schlüsse gezogen werden können ist es wichtig, dass die erzeugten Spieldaten gespeichert werden. Da jedoch die Menge an Daten die schnell riesige Dimensionen annimmt, sollen stellvertretend nur die Daten ganzer Spiele gespeichert werden. Dies ist ein eingehbarer Kompromiss zwischen Vollständigkeit und effizientem Speicherplatzmanagement.
nach reiflichen Überlegungen sollen die folgenden Daten erhoben und abgespeichert werden:

Tabelle 4.1: Zuerhebende Daten

Daten	Erklärung

time	Die Uhrzeit. Dies dient später dem Geschwindigkeitsvergleich.
steps	Die in einem Spiel durchgeführten Züge. Diese geben in der Evaluation später Aufschluss über den Lernerfolge und weisen auf Lernfehler der Agenten, wie beispielsweise das Laufen im Kreis, hin.
apples	Die Anzahl der gefressenen Äpfel in einem Spiel. Maßgiblicher Evaluationsfaktor zur Einschätzung des Lernerfolges.
scores	Die gesammelten und aufsummierten Rewards, welche der Agent in einem Spiel gesammelt hat. Auch dieser Datenwert gibt Aufschluss über den Lernerfolg. Gleichzeitig lässt sich den scores auch noch die effizienz der Lösung und damit des Lernens entnehmen.
wins	Hat der Agent das Spiel gewonnen. Dieser Wert stellt die Endkontrolle des Agenten dar.
epsilon (nur beim Deep Q-Learning Algorithmus)	Gibt die Wahrscheinlichekit für das Ziehen einer zufälligen Aktion wieder.
Learning Rate (lr)	einmalig zuspeichernder skalarer Wert, welcher ein Maß für die Anpassung der Gewichte des bzw. der NN wiedergibt.
board_size	einmalige zuspeichernder vektorieller Wert. Dieser gibt die Feldgröße des Environments an. Dieser ist wichtig für die Evaluation der Robustheit der Agenten.
andere Hyperparamenter der Agenten	Die Agenten werden maßgiblich durch die ihnen zugrundeliegenden Paramenter gesteuert. Um einen Vergleich verschiedenen Agenten durchführen zu können, sind daher die Hyperparamenter von größter Wichtigkeit.

4.3.3 Variation der Datenerhebungsparameter

Um die Möglichkeit auszuschließen, dass die momentan ausgewählten Parameter, wie z.B. Feldgröße, Lernrate, Reward-Funktion usw., für einzelne Agenten vorteilhafte Bedingungen hervorbringen können, sollen die einzelnen Agenten unter verschiedenen Modifikationen getestet werden. Dazu soll die Datenerhebung unter der Anwendung einer Modifikation durchgeführt werden. Zu den Modifikationen zählen:

Variation der Reward-Funktion

Durch das Verändern der Reward-Funktion soll überprüft werden, ob die Agenten unterschiedlich gut auf eine andere Reward-Funktion reagieren. Sollte die Reward-Funktion eins $R_1()$ für den Agenten eins A_1 überdurchschnittlich bessere Resultate liefern, so wird dies erkannt und in der Evaluation berücksichtigt.

Variation der Netzstruktur

Jeder Agent ist abhängig von einer ganzen Reihe an Faktoren, welche über Erfolg und Misserfolg entscheiden. Die Netzstruktur ist dabei einer der besonders wichtigen Faktoren. Je nach Variation der Netzstruktur, kann dies die Qualität eines Agenten entscheidend beeinflussen. Es ist daher angebracht, die einzelnen Agenten unter verschiedenen Netzstrukturen laufen zu lassen, um zum einen die Robustheit des übergeordneten RL Algorithmus zu testen und zum anderen um sicherzustellen, dass die vorherschende Netzstruktur nicht nur für einen Agenten vorteilhaft ist.

Variation der Observation

Eine Überprüfung der Agenten, neben den bereits genannten Variationen, ist auch noch mit der Variation der Observation durchführbar. Diese Veränderung soll Aufschluss über die Qualität der Agenten bei unterschiedlichen Eingabeinformationen liefern. Auch wir so wieder die Wahrscheinlichekit gesenkt, dass gerade diese Observation für einen spezifischen Agenten vorteilhaft ist.

4.4 Anforderungen an die Evaluation

Bei der Evaluation soll der optimalste Agent bestimmt werden. Dabei stellt sich jedoch die Frage, was optimal im Sachzusammenhang bedeutet. Daher sollen die Agenten unter verschiedenen Gesichtspunkten evaluiert werden, um ein möglichst großes Spektrum an Kriterien abzudecken. Die einzelnen Kriterien lauten:

Tabelle 4.2: Evaluationskriterien

Kriterium	Erläuterung
Performance	Welcher Agent erreicht, nach einer festen Anzahl an absolvierten Spielen, das beste Ergebnis? Im Sachzusammenhang mit dem Spiel Snake bedeutet dies: Welcher Agent frisst die meisten Äpfel, nach dem er über eine feste Anzahl an Spielen trainiert wurde?

Effizienz	Welcher Agent löst das Spiel mit der größten Effizienz. Bezogen auf das Spiel Snake bedeutet dies, welches Agent ist in der Lage die Äpfel mit möglichst wenig Schritten zu erreichen und zu fressen? Hierbei werden beispielsweise Probleme, wie das im Kreis laufen sichtbar.
Robustheit	Welcher Agent ist in der Lage in einer modifizierten Umgebung das Spielziel am besten zu erreichen? In Bezug auf Snake bedeutet dies: Welcher Agent ist in der Lage auf einem größeren oder kleineren Spielfeld die meisten Äpfel zu fressen?
Trainingszeit	Welcher Agent schafft es ein festes Ziel in der geringstmöglichen Zeit zu erreichen oder welcher Agent ist als erstes in der Lage durchschnittlich 10 Äpfel zu fressen.
Spielzeit	Welcher Agent schafft es am längsten ein durchzuhalten bevor ein terminaler Zustand erreicht wird? Auf Snake bezogen: Welcher Agent schafft es am längsten am Leben zu bleiben.
Multiprocessing fähig (boolscher Wert)	Welche Agenten sind multiprocessing fähig und können damit schneller und ausgibiger trainiert werden.

Kapitel 5

Agenten

Ein zentraler Aspekt der eines Vergleiches von verschiedenen RL-Agenten ist die genaue Definition der einzelnen Agenten. Basierende auf den Grundlagen 2.2.2 soll in diesem Kapitel der Begriff vervollständigt und die zu vergleichenden Agenten sollen vorgestellt werden.
Erste statistische Erhebungen haben gezeigt, dass die ausgewählten Hyperparameter einen immensen Einfluss auf das Verhalten der Agenten haben. Bestätigt wird diese Aussage durch die Quelle Sutton und Barto (2018). Ein Vergleich zwischen DQN und PPO mit wahllos gewählten Hyperparametern ist folglich wenig aussagekräftig. Daher ist auch die Definition des Begriffs Agent, welcher nur zwischen DQN und PPO diffenenziert, unzureichend.

Angebracht wäre eine neuer erweiterte Definition des Begriffs Agent für diese Ausarbeitung. Diese soll um den entscheidenen Faktor der Hyperparameter erweitert werden. Ein Agent wird daher nicht mehr alleinig durch seine Art (Q-Learning oder Policy Gradient bzw. DQN oder PPO) definiert, sondern ebenfalls durch die ausgewählten Hyperparameter.Eine Analogie aus dem Tierreich sollte hier Klarheit verschaffen.

Im Tierreich gibt es Hunde und Katzen. Diese stellen die RL-Klassen, wie z.B. Q-Learning- oder Policy Gradient Verfahren, dar. Sieht man jedoch genauer hin, so unterscheiden sich die Hunde und Katzen durch ihre jeweiligen Rassen, wie z.B. Pudel und Dalmatiner bei den Hunden und Maine Coons und Norwegische Waldkatzen bei den Katzen. Diese stellen die Algorithmusklassen, wie z.B. DQN oder PPO, dar. Dennoch unterscheiden sich auch Hunde und Katzen der selben Rasse untereinander, nämlich in ihrer DNS. Diese stellt die letzte Differenzierungsebene der Agenten dar. Im Sachzusammenhang stellen die Hyperparameter und Attribute, wie beispielsweise die Netzstruktur, die DNS eines Agenten dar.
Soll nun also ein Vergleich zwischen verschiedenen Agenten vollzogen werden, so gilt es als erstes die einzelnen Agenten zu definieren, daher ihre RL-Klasse, Algorithmus-

klasse und Hyperparameter zu bestimmen.

5.1 Agenten

Im Folgenden werden die einzelnen Agenten, welche untereinander verglichen werden sollen, tabellarisch vorgestellt. Daher wird Aufschluss über Details, wie z.B. die RL-Klasse, Algorithmusklasse, Hyperparameter und die Netzstruktur gegeben.

Tabelle 5.1: Agenten

Agentenname	RL-Klasse	Algorithmusklasse	Hyperparameter
DQN_0.99_64_5e-6_2**12_5e-4	Q-Learning	DQN	• gamma (γ) = 0.99 • batch_size = 64 = 2^6 • epsilon_decrement = 5e−6 • max_mem_size = 2^{12} • lr = 5e−4
DQN_0.95_128_1e-5_2**13_1e-4	Q-Learning	DQN	• gamma (γ) = 0.95 • batch_size = 128 = 2^7 • epsilon_decrement = 1e−5 • max_mem_size = 2^{13} • lr = 1e−4

PPO_0.99_128_10_1e-3_1.5e-3_0.5_1e-3_128_2**11	Policy Gradient	PPO	• gamma (γ) = 0.99 • K_epochs = 10 • epsilion_clip = 0.2 • lr_actor = 1e−3 • lr_critic = 1.5e−3 • critic_loss_coefficient = 0.5 • entropy_coefficient = 0.001 • batch_size = 128 • max_mem_size = 2^{11}
PPO_0.95_128_8_0.5e-3_1e-3_0.5_1e-4_64_2**9	Policy Gradient	PPO	• gamma (γ) = 0.99 • K_epochs = 10 • epsilion_clip = 0.2 • lr_actor = 1e−3 • lr_critic = 1.5e−3 • critic_loss_coefficient = 0.5 • entropy_coefficient = 0.001 • batch_size = 128 • max_mem_size = 2^{11}

Literaturverzeichnis

[Baker u. a. 2019] Baker, Bowen ; Kanitscheider, Ingmar ; Markov, Todor M. ; Wu, Yi ; Powell, Glenn ; McGrew, Bob ; Mordatch, Igor: Emergent Tool Use From Multi-Agent Autocurricula. In: *CoRR* abs/1909.07528 (2019). – URL http://arxiv.org/abs/1909.07528

[Bowei Ma 2016] Bowei Ma, Jun Z.: *Exploration of Reinforcement Learning to SNAKE*. 2016. – URL http://cs229.stanford.edu/proj2016spr/report/060.pdf

[Chunxue u. a. 2019] Chunxue, Wu ; Ju, Bobo ; Wu, Yan ; Lin, Xiao ; Xiong, Naixue ; Xu, Guangquan ; Li, Hongyan ; Liang, Xuefeng: UAV Autonomous Target Search Based on Deep Reinforcement Learning in Complex Disaster Scene. In: *IEEE Access* PP (2019), 08, S. 1–1. – URL https://ieeexplore.ieee.org/abstract/document/8787847

[Goodfellow u. a. 2018] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning. Das umfassende Handbuch*. MITP Verlags GmbH, 2018. – URL https://www.ebook.de/de/product/31366940/ian_goodfellow_yoshua_bengio_aaron_courville_deep_learning_das_umfassende_handbuch.html. – ISBN 3958457002

[Haarnoja u. a. 2018] Haarnoja, Tuomas ; Zhou, Aurick ; Abbeel, Pieter ; Levine, Sergey: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: *CoRR* abs/1801.01290 (2018). – URL http://arxiv.org/abs/1801.01290

[Lapan 2020] Lapan, Maxim: *Deep Reinforcement Learning*. MITP Verlags GmbH, 2020. – URL https://www.ebook.de/de/product/37826629/maxim_lapan_deep_reinforcement_learning.html. – ISBN 3747500366

[Mnih u. a. 2016] Mnih, Volodymyr ; Badia, Adrià P. ; Mirza, Mehdi ; Graves, Alex ; Lillicrap, Timothy P. ; Harley, Tim ; Silver, David ; Kavukcuoglu, Koray: Asynchronous Methods for Deep Reinforcement Learning. In: *CoRR* abs/1602.01783 (2016). – URL http://arxiv.org/abs/1602.01783

[Mnih u. a. 2015] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg ; Petersen, Stig ; Beattie, Charles ; Sadik, Amir ; Antonoglou, Ioannis ; King, Helen ; Kumaran, Dharshan ; Wierstra, Daan ; Legg, Shane ; Hassabis, Demis: Human-level control through deep reinforcement learning. In: *Nature* 518 (2015), Februar, Nr. 7540, S. 529–533. – URL `http://dx.doi.org/10.1038/nature14236`. – ISSN 00280836

[Schulman u. a. 2015] Schulman, John ; Levine, Sergey ; Moritz, Philipp ; Jordan, Michael I. ; Abbeel, Pieter: Trust Region Policy Optimization. In: *CoRR* abs/1502.05477 (2015). – URL `http://arxiv.org/abs/1502.05477`

[Schulman u. a. 2017] Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal Policy Optimization Algorithms. In: *CoRR* abs/1707.06347 (2017). – URL `http://arxiv.org/abs/1707.06347`

[Sutton und Barto 2018] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. – URL `http://incompleteideas.net/book/bookdraft2018jan1.pdf`

Erklärung

Hiermit versichere ich, Lorenz Mumm, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Lorenz Mumm