Hochschule der Medien
Fakultät für Druck und Medien
Medieninformatik

Deep Reinforcement Learning

Wie wirkt sich die visuelle Augmentation des Environments auf die Generalisierung eines RL-Agenten in Procgen aus?

zur Erlangung des akademischen Grades
Bachelor of Science

Thema: Deep Reinforcement Learning

Autor: Nicolas Reinhart - nr034@hdm-stuttgart.de
MatNr. 30038

Version vom: 17. August 2020

1. Betreuer: Prof. Dr.-Ing. Johannes Maucher
2. Betreuer: M.Sc. Johannes Theodoridis

Abstract

Heutzutage sind für viele Reinforcement Learning Benchmark-Umgebungen, wie Gym [BCP+16] von openAI der Goldstandard. Das Team von Procgen, ebenfalls von openAI, ging noch einen Schritt weiter und implementiert für einige der Atari Spiele eine prozedurale Generierung der Level. Das allein führt bei einigen Algorithmen bereits zu besserer Sample Efficiency und einer besseren Generalisierung.

In dieser Arbeit wird untersucht, wie sich eine visuelle Augmentation der Environments im Training oder während der Evaluation auf den allgemeinen Erfolg, den Reward und die Generalisierung auswirkt. Experimente mit maskierten Informationen oder invertierter Semantik untersuchen darüberhinaus die Relevanz bestimmter visueller Informationen und die Generalisierung von Konzepten. Durchgeführt werden die Experimente im Chaser-Environment von Procgen. Dieses Spiel ist angelehnt an den Atari-Klassiker "Ms. Pac-Man". Wie im Original ist es das Ziel, alle Orbs einzusammeln und auf dem Weg nicht von den Geistern gefressen zu werden.

Die Experimente der vorliegenden Arbeit zur farblichen Änderungen des Environments zeigen, dass bspw. die Existenz einer gewissen Farbe für die Erkennung eines Objekts große Relevanz besitzt, wenn das Objekt diese Farbe im Training hat. So nimmt die Performance eines Agenten rapide ab, wenn die Farbe der Orbs in der Evaluation eine andere ist, verglichen mit der zu Trainingsbedingungen. Des Weiteren zeigt sich über die Arbeit hinweg immer wieder, dass die Änderungen der Evaluationsumgebung und die daraus resultierenden Änderungen der Pixel-Verteilung, welche der Agent als Input bekommt, ebenfalls fatale Auswirkungen auf die Performance des Agenten haben.

Experimente mit Änderung der Semantik von Objekten zwischen Training und Evaluation zeigen eindeutig, dass das der Architektur vorangestellte Convolutional Neural Network (CNN) stark auf die optische Erscheinung der Objekte fixiert ist. Das Vertauschen zweier Sprites stellt eine unüberwindbare Hürde für den Agenten dar. Die Feature Extraction des CNN scheint stark auf die Texturen und die Formen der jeweiligen Objekte angewiesen zu sein, um diese korrekt zu erkennen. So scheut sich der Agent, bei Experimenten mit invertierter Semantik der großen Orbs und der Geister, einen statischen Gegner einzusammeln - ein großen Orb, welcher ihn mit den Bewegungen eines Geistes verfolgt, wird hingegen gedankenlos eingesammelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALE ..... Atari Learning Environment
CNN ..... Convolutional Neural Network
DNN ..... Deep Neural Network
DRL ..... Deep Reinforcement Learning
FFN ..... Feed Forward Netzwerk
GPU ..... Graphical Processing Unit
IHPfPVG ..... Investigating Human Priors for Playing Video Games
KI ..... Künstliche Intelligenz
MDP ..... Markov Decision Process
MLP ..... Multi Layer Perceptron
MSE ..... Mean-Squared-Error
NN ..... Neuronales Netzwerk
PCG ..... Prozeduraler Content Generierung
POMDP ..... Partially Observable Markov Decision Process
RL ..... Reinforcement Learning
SLP ..... Single Layer Perceptron
SW ..... Sliding Window

1 Einleitung & Motivation

Die Forschung rund um Reinforcement Learning (RL) hat in den letzten Jahren stark an Interesse gewonnen. Eine Suche des Begriffs 'Reinforcement Learning' auf Google trends zeigt, dass sich die Häufigkeit der Anfragen dieses Begriffs seit 2016 knapp verdreifacht hat. Gegenwärtige Benchmark-Umgebungen wie openAI's Gym [BCP+16] helfen den Fortschritt in der Forschung weiter voranzutreiben. Ein Weiterer, häufig verwendeter Simulator ist das Atari Learning Environment (ALE) [BNVB13]. Das ALE bietet signifikante Forschungs-Aufgaben für RL, model-learning, model-based planning, imitation learning, transfer learning und intrinsic motivation [BNVB13]. Neben den Vorteilen, die eine solche Benchmark-Umgebung und die darin befindlichen Atari Spiele mit sich bringen, haben diese Umgebungen immer noch entscheidende Nachteile: Die Level in einigen dieser Umgebungen sind händisch erstellt worden. Somit gibt es nur eine begrenzte Anzahl an Szenarien, in denen trainiert werden kann. Dieser Fakt führt bei der Arbeit mit RL-Agenten oft zu schlechterer Generalisierung und instabilen Policies [ZWP18]. Das Framework von Procgen adressiert das Problem der Generalisierung durch prozedurale Erstellung der Spielumgebung. Der prozedurale Anteil steuert dabei Entscheidungen wie die Logik des Levellayouts, die Position und Spawn-Zeiten von Spielelementen, die Auswahl der Game-Assets und weitere [CHHS19].

Im Paper [DAP+18] untersuchen die Autoren, warum Menschen im Lösen komplexer Videospiele so gut sind. Genauer untersuchen sie die Rolle von Vorwissen, welches hilfreich scheint, um eine schnelle und effiziente Exploration eines Frames oder eines Levels zu ermöglichen. Die Arbeit der Autoren dient als grundlegende Inspiration für diese Arbeit. Über den Verlauf ihrer Experimente hinweg verändern sie die visuelle Darstellung ihres Spiels bis zu einem Punkt, an dem bspw. die Ähnlichkeit von gleichen Objekten vollständig maskiert ist. Die in der Arbeit [DAP+18] erstellten Level werden von Menschen und einem RL-Agenten gespielt und ihre Leistungen werden verglichen. Die Level für den Agenten sind aufgrund der Komplexität des gewählten Spiels vereinfacht.

In der vorliegenden Arbeit wird untersucht, wie sich visuelle Augmentation des Environments auf die durch Procgen geförderte Generalisierung und den allgemeinen Erfolg trainierter Agenten auswirkt. Hierfür wird ein Environment von Procgen mit dem Titel *Chaser* [CHHS19] [S.15 A.6] herangezogen. Dieses Spiel stellt ein Replika des Atari-Klassikers "Ms. Pac-Man" dar.

Die Experimente sind in drei verschiedene Sets eingeteilt. Das erste Set überprüft die Reproduzierbarkeit der Ergebnisse aus dem Procgen-Paper und untersucht die Auswirkung prozeduraler Level-Generierung auf die Performance eines Agenten.

Das zweite Set beschäftigt sich mit visuellen Veränderungen der Farbe der Orbs und des Hintergrundes (BG), aber auch mit der Maskierung von Information, die im Training gegeben war. So werden in diesem Set z.B. Agenten in der Evaluation mit

Herausforderungen konfrontiert wie: die Farbe der Orbs ist eine andere, als im Training oder die Orbs werden überhaupt nicht dargestellt.

Das dritte Set befasst sich ausschließlich mit der semantischen Invertierung von visueller Information unterschiedlicher Spielelemente. Hierbei werden bspw. in der Evaluation die Sprites von kleinen Orbs und Mauern oder großen Orbs und Gegnern vertauscht und somit die Semantik optisch invertiert. Ein Bild des Environments kann Abbildung 3 entnommen werden. Kleine Orbs sind hier in grün und große sind gelb. Die Gegner sind die grünen Blobs mit drei Stacheln. Die Mauern sind grau. Diese Abbildung zeigt das Spiel in seiner unveränderten Form.

Die ausgeführten Experimente finden in Anlehnung an die in [DAP+18] durchgeführten Experimenten statt. Besonders Änderungen der visuellen Semantik und die daraus resultierende Änderung der Erscheinung von Objekten, sollte die verwendete IMPALA-Architektur [ESM+18], aufgrund des vorangeschaltenen CNN, vor komplexe Aufgaben stellen.

Ziel der Arbeit Ziel dieser Arbeit ist die Untersuchung der Generalisierung des eingesetzten IMPALA-Netzwerks 3.7 im Zusammenspiel mit dem verwendeten PPO Algorithmus 3.6,in der Procgen-Umgebung [CHHS19]. Dies geschieht anhand unterschiedlicher visueller Augmentationen des Trainings- und/oder der Evaluations-Environments. Genauer wird die durch Procgen 4.1 geförderte Generalisierung in ungesehenen Leveln untersucht. Die Evaluation ist anhand vorab erstellter Fragen realisiert. Diese geschieht unter Verwendung zweier visueller Veränderungen des Agenten-Inputs, während des Trainings und/oder der Evaluation: farbliche Änderungen von Spielelementen wie bspw. der Orbs und visuell-semantische Invertierung von zwei unterschiedlichen Spielelementen.
Diese Experimente folgen drei Sets an Fragen, welche in Kapitel 5.1 ausführlich aufgelistet sind.

In den Experimenten des Unterkapitels 5.3 wird Data Augmentation in Form von farblicher Änderung oder optischer Maskierung mancher Spielelemente betrieben. Hiermit wird einerseits durch optische Maskierung untersucht, wie relevant gewisse Objekte des Environments, hier die kleinen Orbs, für einen erfolgreichen Abschluss einer Episode sind. Andererseits wird mittels farblicher Änderungen untersucht, ob während des Trainings eine Konzeptualisierung der kleinen Orbs stattfindet. Mit einem ausreichenden Konzept eines kleinen Orbs, sollten farbliche Änderungen des Orbs in der Evaluation zu keinem Verlust der Performance führen. Zudem wird untersucht, ob mehrere Level innerhalb einer kleinen Anzahl an Zeitschritten auswendig gelernt werden können, und wie relevant die visuelle Information der kleinen Orbs dabei ist.

Die Experimente in Unterkapitel 5.4 untersuchen, wie ein trainierter Agent in der Evaluation abschneidet, wenn Spielelemente, wie die großen Orbs und die Geister, auf visuelle Weise semantisch invertiert sind. Hierbei wird untersucht, ob während der

Feature Extraction auch nicht-optische Features, wie bspw. ein Bewegungsmuster, ein Indikator für ein Spielelement sein kann. Des Weiteren wird durch eine Invertierung der kleinen Orbs und der Mauern überprüft, ob ein Agent ein Grundverständnis des Spiels aufweist, oder ob er dadurch auf die Leistung eines Random-Agenten zurückfällt.

Das erste erwähnte Set an Fragen 5.1 beschäftigt sich mit der Reproduktion der Ergebnisse der Arbeit [CHHS19] und der generellen Auswirkung der prozeduralen Level-Generierung und schafft lediglich das Fundament der folgenden Experimente.

Kapitel 2 beschreibt die verwandten Arbeiten, welche für die vorliegende Arbeit als Inspiration dienen oder technisch relevant sind. Das dritte Kapitel (3) legt die theoretische Grundlage für die Arbeit dar. Hier wird unter anderem der PPO Algorithmus und die IMPALA-Architektur erläutert. Das Kapitel 4 befasst sich mit dem Setup der Experimente. Hier wird das Environment beschrieben und seine Eigenschaften erläutert. Kapitel 5 handelt von der Durchführung der Experimente. Wie zuvor erwähnt, sind die Experimente in Sets unterteilt. Diese Sets werden systematisch mit Experimenten untersucht. Das sechste Kapitel (6) gibt eine Ausblick über offene oder während der Bearbeitung des Themas aufgekommene Fragen und mögliche Abläufe für Experimente, um diese zu untersuchen. Kapitel 7 fasst die Arbeit abschließend zusammen und beschreibt die Erkenntnisse, welche aus den Experimenten gewonnen werden können.

2 Verwandte Arbeiten

Das Problem des Overfittings ist in der Welt des Machine Learnings weit verbreitet. RL hat die generelle Problematik, dass Agenten oft schlecht generalisieren und somit innerhalb ihrer Trainingsumgebung zwar gute Leistungen erbringen, außerhalb jedoch nicht ([CHHS19], [ZVMB18]). Diese Arbeit verwendet die Ansätze der prozeduralen Generierung und der Data Augmentation, um das Problem der Generalisierung zu untersuchen.

Die Idee von Procedural Content Generation (PCG) ist kein neuer Ansatz. Die Spieleindustrie macht sich die Eigenschaften von PCG bereits Ende der achtziger Jahre in "Dungeons & Dragons" zu nutze. Durch die Anwendung von PCG in der Domäne Künstlicher Intelligenz (KI), kann die Generalisierung und Sample Efficiency in einigen Environments bereits stark verbessert werden. So war openAI mit Hilfe von PCG in der Lage, eine ausreichend große Level-Anzahl und Diversität bereitzustellen, um das Spiel "Capture the Flag" zu meistern. Weiter schaffen die Autoren es ebenso, dass die Spieler eines Teams in Kooperation arbeiten [JCD+19]. Inzwischen ist die Verwendung von PCG zur künstlichen Erweiterung der Trainingsumgebung bzw. des Trainingsdatensets eine etablierte Alternative zum händischen Erweitern. Arbeiten wie [RT19] oder [RT20], beide von Sebastian Risi und Julian Togelius, zeigen die mögliche Anwendung für KI und Herrausforderungen, die dabei zu beachten sind. Die hier aufgeführten Arbeiten stützen die These, dass eine prozedurale Erstellung des Trainingsenvironments positive Auswirkungen auf die Generalisierung hat. Die Experimente in 5.2 und 5.2.1, sowie einige Experimente in Unterkapitel 5.3 liefern empirische Beweise für die positive Auswirkung.

Data Augmentation wird typischerweise im Supervised Learning eingesetzt. Arbeiten wie [GRM+18] zeigen, welche Erfolge mittels künstlicher visueller Erweiterung eines bereits vorhandenen Datensatzes erzielt werden können. Die Autoren erweitern das *ImageNet* [DDS+09] zu ihrem eigenen *Stylized ImageNet*. In ihrer Arbeit zeigen sie, dass das eingesetzte CNN, im Fall des ImageNets, Texturen mit größerem Fokus als Formen lernen. Um diese Einseitigkeit zu verbessern, fügt ihre Optimierung dem zugrunde liegenden Datensatz bspw. Bilder hinzu, bei denen Texturen anderer Bilder auf die vorhandenen Bilder multipliziert sind. Was danach vom eigentlichen Bild übrig ist, sind lediglich die Form und die Kontraststufen. Das Training auf den erweiterten Daten resultiert in einer "[...] improved object detection performance and previously unseen robustness towards a wide range of image distortions, highlighting advantages of a shape-based representation" [GRM+18][S. 1] [1]. Diese Arbeit ist nur ein Beispiel für die Empfindlichkeit von CNNs gegenüber optischer Änderung. Ein weiteres Beispiel liefert die Arbeit [RW19]. Hier untersuchen die Autoren anhand spezifischer, optischer

[1] Übersetzung des Verfassers: verbesserten Leistungen bei Bilderkennung und bisher ungesehen Robustheit bezüglich einer weiten Spanne an verschiedenen Bildverzerrungen, was auf die Vorteile von formbasierter Repräsentation hinweist.

Stimuli, welche Muster potentiell besser erkannt werden können, als andere. Auch die Verwendung von Data Augmentation in RL ist nicht neu. Die Arbeit [RGY+20] untersucht die Auswirkungen verschiedener Arten von visuellen Augmentationen und erzielt mit ihrem Ansatz in den 16 Environments von Procgen eine bessere Performance um bis zu ca. 40%. Auch ältere Arbeiten, wie [ZWP18], sind ein Beleg für die Relevanz von Data Augmentation in RL.

Die in dieser Arbeit durchgeführten Experimente sind grundlegend durch die Arbeit [DAP+18] und die darin behandelten Experimente inspiriert. In der Arbeit werden die Auswirkungen von Vorwissen des Menschen auf die Performance, die der Mensch in einem Spiel erbringen kann, gemessen und mit einem RL-Agenten verglichen. Darüber hinaus testen sie die Kenntnis grundlegender Interaktionen mit Gegenständen, wie bspw. einer Leiter. So wird in dieser Arbeit das Sprite der Leiter durch ein einfarbiges Bild ersetzt, welches dieselben x- und y-Dimensionen wie das Bild der Leiter hat. Hierdurch soll die Semantik und Identität des Objekts maskiert werden. Ebenso haben sie die Semantik von zwei Objekten vertauscht. Das Bild der Leiter wird dann bspw. durch ein Bild aus mehreren kleinen Flammen ersetzt. Die Herausforderungen, die dem Menschen gestellt wurden, wurden in abgewandelter Form ebenfalls dem RL-Agenten gestellt. Hier ist die These, dass visuelle Änderungen, die Menschen vor eine schwierige Aufgabe stellen, für einen RL-Agenten, welcher mit der Situation im Training konfrontiert wurde, keine Rolle spielen, da er über keinerlei Vorwissen verfügt. Die von den Autoren aufgestellte These stellt sich nach ihrer Auswertung als wahr heraus. Die Experimente der Arbeit [DAP+18] bieten Potential herauszufinden, wie relevant gewisse visuelle Stimuli für den Agenten sind. So lässt sich bspw. mittels visueller Maskierung mancher Spielelemente testen, wie relevant diese Information für den Abschluss eines Environments sind.

3 Deep Reinforcement Learning - Theorie

Das wissenschaftliche Feld des maschinellen Lernens bietet viele verschiedene Ansätze einem Agenten beizubringen, wie er eine gegebene Aufgabe bewältigt. Dieses Feld lässt sich in drei Gruppen einteilen: Supervised Learning, Unsupervised Learning und *Reinforcement Learning* (RL). RL differenziert sich von den anderen beiden Feldern durch die fehlenden Trainingsdaten. In diesem Feld werden die Trainingsdaten mittels Interaktionen in einem Environment gesampled. RL ist ein facettenreiches Forschungsfeld. Es vereint Anwendungen aus bspw. der Kontrolltheorie, dem maschinellen Lernen und weiteren Feldern in Einem. RL ist inspiriert durch den Ansatz des verstärkenden Lernens der Verhaltenspsychologie. Im Folgenden werden die für diese Arbeit benötigten Grundlagen eingeführt.

3.1 Markov Decision Process

Der *Markov Decision Process* (MDP) [Bel57] stellt ein mathematisches Framework für das Problem des Lernens durch Interaktion dar. Das Framework stellt die Rahmenbedingungen, in einem diskreten, stochastischen und sequenziellen Environment zielorientierte Entscheidungen zu treffen. Hierzu wird die gesamte Interaktion zwischen dem Agenten und dem Environment auf drei Signale reduziert. Ein *State* ist hierbei der Zustand des Environments und beschreibt alle Übergangswahrscheinlichkeiten. Der Agent fällt seine Entscheidung auf Basis dieses States. Mittels *Actions* kann der Agent mit dem Environment interagieren und dessen State verändern. Für die Ausführung einer Action erhält der Agent einen *Reward* und den nächsten State. Der Reward stellt die Belohnung dar und kann sowohl positiv als auch negativ sein. Das Ziel des Agenten ist es, möglichst zielführende Actions auszuwählen und damit seinen Reward zu maximieren.

Dieser Prozess ist eine Erweiterung des *Markov Reward Process* um die Fähigkeit Entscheidungen zu treffen. MDPs bieten die Möglichkeit Entscheidungen selbst in Situationen zu treffen, in denen der Ausgang nicht zu 100 % von den Aktionen des Agenten abhängt, sondern durch Zufall verfälscht wird. Dem Agenten steht dabei die Teilmenge $A(s) \subset A$, die Menge aller möglichen Aktionen in State $s \in S$ zur Verfügung. Eine generelle Formulierung eines MDP deckt die meisten RL-Probleme ab und besteht aus dem folgenden Tupel $\langle S, A, P, R, \gamma \rangle$:

- S ein Set aus States
- A ein Set aus Actions
- P Übergangswahrscheinlichkeiten $P^a_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = S, a_t = A]$
- R eine Reward-Funktion $R^a_s = \mathbb{E}[R_{t+1} | s_t = S, a_t = A]$

- γ ein Diskontierungsfaktor $\gamma \in [0, 1]$
  - $\gamma = 0$ => Monte-Carlo (Aktionswahl nur anhand des aktuellen States)
  - $\gamma = 1$ => $TD(1)$ (Aktionswahl anhand aller möglichen Folgestates)

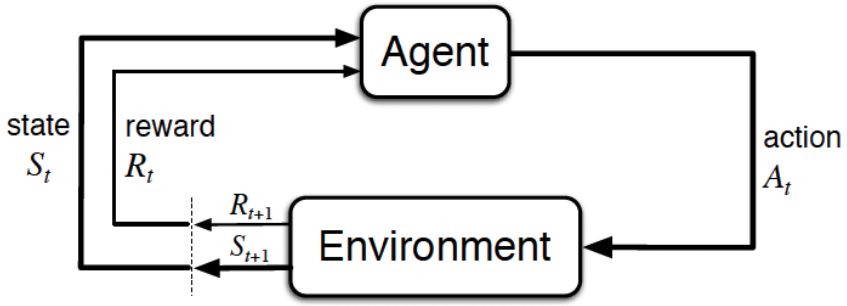


Abbildung 1: Visualisierung des RL Prozesses mittels eines MDP [SB11] [S.38]

Abbildung 1 beschreibt die Interaktionen in einem MDP. Der Agent interagiert mit dem Environment für eine diskrete Anzahl an Schritten $t = 1, 2, 3, ...$ Am Anfang eines jeden Schrittes t wählt der Agent eine Aktion a_t und erhält daraufhin einen nummerischen Reward $R_{t+1} \in R \subset \mathbb{R}$ und den Folgestate s_{t+1}.

Während man im Markov Reward Prozess versucht die Action-Value Function $Q_*(a)$ einer jeden Action zu schätzen, berechnet man in einem MDP die State-Action Value Function $Q_*(s, a)$ einer jeden Action in jedem State oder man schätzt den Value eines States $V_*(s)$. Bei der Berechnung des Values mittels $V_*(s)$, wird eine optimale Auswahl der Actions vorausgesetzt. Ein Sternchen im Index steht hier repräsentativ für die optimale Policy. Der Index der Funktionen V_π und Q_π referenziert folgend auf die aktuell verfolgte Policy. Die State-Value Function V kann dabei als Dekomposition, in Form eines Erwartungswerts des aktuellen Rewards R_{t+1} und des diskontinuierten Values des Folge-States $V(s_{t+1})$, formuliert werden.

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|s_t = S]$$

Die State-Action-Value Function Q kann auf ähnliche Weise formuliert werden. Diese Funktion schätzt, wie gut eine Action a unter der Prämisse ist, dass der Agent sich im State s_t befindet.

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})|s_t = S, a_t = A]$$

Das Verhalten des Agenten kann, innerhalb des durch den MDP gegebenen Rahmens, als Policy $\pi(a|s)$ definiert werden, wobei π eine Wahrscheinlichkeitsverteilung über alle States mit allen Actions darstellt. Diese Verteilung kann wie folgt definiert werden:

$$\pi(a|s) = \mathbb{P}_t[a_t = A|s_t = S].$$

Das beste Verhalten für ein gegebenes Problem zu finden, bedeutet analog die Policy π zu finden, die den Erwartungswert des akkumulierten diskontinuierten Rewards, über die Schritte t hinweg, maximiert. Der Return G_t ist der totale, diskontinuierte Reward und wird durch die folgende Gleichung definiert:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ [2]} .$$

3.2 Deep Learning

Im Gegensatz zu einem *Single Layer Perceptron* (SLP), enthält ein *Multi Layer Perceptron* (MLP) verborgene Schichten zwischen der Eingabe- und der Ausgabeschicht. Die verborgenen Schichten sind die sogenannten Hidden-Layers, von denen es beliebig viele hintereinander geben kann. Die Anordnung von mehreren Hidden-Layers von Neuronen, ist inspiriert vom biologischen Informations-Verarbeitungs-Prozess des menschlichen Gehirns. Da die verborgenen Layers nicht direkt mit der Ausgabe zusammenhängen, werden sie Hidden-Layer genannt [GBC16][S. 164-165].

Ein Deep Neural Network (DNN) zielt darauf ab, Schicht für Schicht, aus einfachen Inputs wie bspw. einem Vektor aus Zahlen oder einem Pixel-Stream, informative Features zu extrahieren. Abbildung 2 zeigt eine abstrakte Darstellung eines DNN. Was ein einfaches MLP von einem vollwertigen DNN differenziert ist die Extraktion der Features (orange in Abb. 2), die im Deep Learning dem Netzwerk überlassen ist. Diese Features müssen somit nicht manuell extrahiert werden. Der blaue Teil der Abbildung (2) ist applikations-spezifisch und könnte statt einem Classifier bspw. durch ein Regressionsmodell ersetzt werden. Die automatisierte Feature Extraction ermöglicht es eine Architektur auf unterschiedliche Probleme anzuwenden. Das ist möglich, da kein domänen-spezifisches Vorwissen mehr nötig ist, um dem applikations-spezifischen Teil des Netzwerks informative Features bereitzustellen. Die Verbindungen zwischen den Neuronen der jeweiligen Layer sind, wie bei einem SLP oder einem MLP, gerichtet und gewichtet. In der Feature Extraction befinden sich typischerweise Layer wie Convolutional-, Pooling-, RNN-, LSTM- und Attention-Layers. Der Teil des Classifier wird oft mit einem einfachen SLP oder einem mehrschichtigen MLP realisiert, bestehend aus Dense-Layers oder Fully-Connected-Layers.

Viele Problemstellungen, wie das Verarbeiten eines Pixel-Streams, sind sehr komplex. Ein Frame eines Spiels von 60 auf 60 Pixeln, aufgeteilt in drei Farbkanäle, ergibt eine Anzahl von 10.800 Datenpunkten, die verarbeitet werden müssen. Anders als bei einem einfachem SLP oder einem MLP werden beim Deep Learning Entscheidungen nicht direkt abhängig vom Input getroffen. Hier werden über mehrere Abstraktions-Layer hinweg, nicht-lineare Anpassungen an den Daten vorgenommen. Eine ausführliche Motivation für tiefe Architekturen gibt die Arbeit [B+09] von Yoshua Bengio.

[2] Sutton und Barto, vgl. [SB11] [S.44]

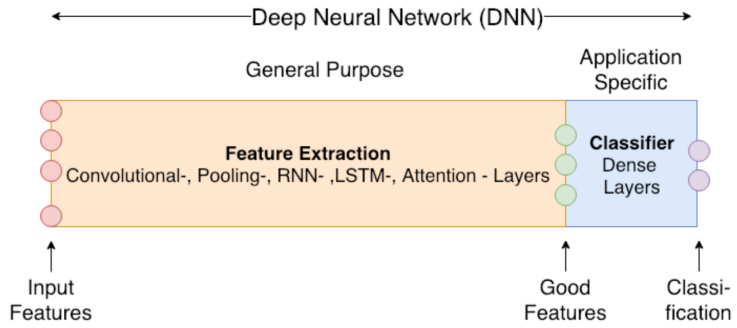


Abbildung 2: Abstrakte Darstellung eines DNN [Mau20].

Aufgrund der großen Anzahl an Neuronen sind die tiefen Neural Networks (NN) in der Lage komplexe Problemstellungen, wie aus einem Pixel-Stream eine Action abzuleiten, die in dieser Situation Reward-steigernd erscheint, auszuwählen. Im Bereich Deep Q-Learning wird bspw. die Approximation der State-Action-Value Function $Q(s, a)$, mit einem NN realisiert. Das State-Action-Value NN approximiert dann bspw. im Training den erwarteten durchschnittlichen Reward des übergebenen States-Action Tupels (s, a) und wird typischerweise anhand des TD-Errors 3.4.1 oder des Mean-Squared-Errors (MSE) [3] zum tatsächlich erhaltenen Return hin korrigiert.

3.2.1 Deep Feedforward Netzwerke

Deep *Feedforward Networks* (FFN) sind die Basis des Deep Learnings. Ziel einer solch tiefen Architektur ist es, eine Funktion zu approximieren. So mapped ein solches Netzwerk, im Falle eines Classifiers, ein Eingabebild x zu einem Label y. Das Mapping von Bild zu Label ist definiert durch $y = f(x, w)$, wobei w die Gewichte darstellt, die es zu lernen gilt. Diese werden optimiert, um die bestmögliche Approximation von $f(x)$ zu ermöglichen. Der Name der FFN definiert den Informationsfluss. So haben FFNs, im Vergleich zu bspw. rekurrenten Netzwerken, ausschließlich vorwärts gerichtete Verbindungen - daher die Bezeichnung feedforward. Diese Grundarchitektur stellt die Basis vieler Anwendungen, wie CNNs dar. CNNs sind lediglich eine spezielle Form eines FFNs. Solche Netzwerke werden durch azyklische Graphen beschrieben, welche definieren, wie die approximierende Funktion zusammengesetzt ist. Hat man bspw. ein Netzwerk mit drei Schichten, so besteht dieses aus drei verschiedenen Funktionen, die wie folgt miteinander verkettet sind:

$$f(x) = f^3(f^2(f^1(x)))$$

In diesem Fall stellt f^1 den ersten Layer dar, f^2 den zweiten, usw. Die gesamte Länge der Verkettung gibt die Tiefe des Netzwerks an. Der letzte Layer ist mit dem Output

[3] Der MSE ist die Summe, der quadrierte Differenz zwischen tatsächlichen Wert $x(i)$ und der Vorhersage $\hat{x}(i)$, geteilt durch die Anzahl an Datenpunkten. $MSE = \frac{1}{n} \sum_{i=1}^{n} (x(i) - \hat{x}(i))^2$

des Netzwerks verbunden. Ein repräsentatives Bild der Verkettung in Form eines MLP der Tiefe $L = 3$ kann dem Anhang entnommen werden (7). Die Eingabeschicht wird hierbei nicht zur Tiefe des Netzwerks gezählt.

Während des Trainings werden die Parameter des gesamten Netzwerks so verändert, dass sich die Funktion $f(x)$ an die optimale Approximation $f_*(x)$ annähert. Jedes Element des Trainings x muss also im Output-Layer möglichst genau das korrekte Label y produzieren. Wie die Layer zwischen Input und Output korrigiert werden, um näher an $f_*(x)$ zu gelangen, wird über den angewandten Lernalgorithmus definiert.

Eine tiefe FFN-Architektur könnte wie folgt sein: Am Netzwerk liegt der Input, bspw. ein eindimensionaler Vektor, an. Dieser Vektor wird dann durch eine Serie an Hidden-Layers, bis hin zum Output-Layer transformiert. Jeder Hidden-Layer besteht aus mehreren Neuronen, die vollständig mit den Neuronen des vorherigen und nachfolgenden Layers verbunden sind. Neuronen in Hidden-Layers teilen hier keine Verbindungen untereinander und agieren unabhängig voneinander. Der eben beschriebene Teil der Architektur übernimmt die Feature Extraction. Der letzte vollständig verbundene Layer entspricht dem Output-Layer. Im Fall von Klassifikationen würden im Output-Layer die Scores für die jeweiligen Klassen stehen, bspw. 10 Outputs bei 10 verschiedenen Klassen.

3.3 Convolutional Neural Networks

CNNs sind seit mehreren Jahren etabliert im Einsatz für Objekt-Detektion, Objekt-Erkennung, Sprach-Erkennung und weitere. Diese Netzwerke sind eine spezialisierte Form der in Unterkapitel 3.2.1 vorgestellten FFNs. Ebenso wie FFNs bestehen CNNs oft aus sehr vielen Schichten an Neuronen, mit Gewichten, die es zu lernen gilt. Der größte Unterschied zwischen FFNs und CNNs besteht in der Verwendung der Convolutional-Layer bzw. in der Anwendung von Convolutional Filtering. Diese Layer sind darauf ausgelegt lokale Informationen, bspw. aus benachbarten Pixeln aus Bildern oder umgebenden Wörtern in einem Text in ein Feature zu extrahieren. Aufbauend auf der Annahme, dass die umgebende Region für eine betrachtete Stelle relevant ist, teilen sich die Neuronen dieser Region die Gewichte zum nachfolgenden Neuron. Durch die many-to-one-Beziehung sind CNNs in der Lage, komplexe Inputs wie ein Bild zu verarbeiten. CNNs lernen damit automatisch häufig vorkommende Merkmale der Trainingsdaten, welche für den applikations-spezifischen Teil der Architektur relevant sind.

Die Bilder des Datensatzes *CIFAR-10* [4] [KNH] sind bspw. $32 * 32$ Pixel groß und haben drei Farbkanäle (RGB). Damit hat der ersten Convolutional-Layer einen Input von $32 * 32 * 3 = 3.072$ Datenpunkten. Im Fall dieser Arbeit wird bereits mit doppelt so großen Frames, ebenfalls mit drei Farbkanälen gearbeitet ($64 * 64 * 3 = 12.288$). Auch diese Größe ist noch weit von einem brauchbaren Bild entfernt, wie Abbildun-

[4] Der Datensatz von CIFAR-10 beinhaltet 60.000, $32 * 32$ Farbbilder, eingeteilt in 10 Klassen.

gen 3 und 4 zeigen. Eine einfache FFN-Architektur benötigt für ein solches Bild, je nach Aufbau, sehr viele Gewichte und ist deshalb ungeeignet, um direkt mit solchem Input zu interagieren. Convolutional-Layer verbinden Regionen von Neuronen des vorangegangenen Layers mit einem Neuron des Convolutional-Layers. Diese Layer bieten durch die Feature Extraction aus Regionen eine effiziente Reduktion des vorangegangenen Layers. Eine Veranschaulichung dieses Konzepts bezüglich des Netzwerks, bei zweidimensionalem Input, ist im Anhang gegeben (7).

CNNs für die Farbbild-Verarbeitung haben im Unterschied zu FFNs, ebenso wie ihr Input, einen dreidimensionalen Aufbau - die Neuronen werden in Höhe, Breite und Tiefe organisiert. Die Breite und die Höhe stehen hierbei für die räumlichen Informationen und die Tiefe für die Anzahl der Farbkanäle des anliegenden Bildes. So bestehen Convolutional-Layer in diesem Fall nicht aus einfachen Neuronen-Schicht, sondern aus einem dreidimensionalen Volumen an Neuronen.

Typischerweise bestehen CNNs aus drei verschiedenen Layern: Convolutional-Layer, Pooling-Layer und Fully-Connected-Layer. Die einzelnen Layer sind folgend detaillierter erläutert.

Convolutional: Diese Layer berechnen den Output aus den lokalen Regionen des Inputs. Sei $(r \times c)$ der zweidimensionale Input, bspw. ein Schwarz-Weiß-Bild, wobei r die X- und c die Y-Koordinate eines Pixels angibt. So gibt $r * c$ die Größe des Bildes an. Weiter sei $(a \times b)$ ein Filter, mit der *Kernel Size* $a * b$, wobei der Filter kleiner als der Input ist. Dieser Filter wird von oben links, bis unten rechts über den Input geschoben. Hierbei wird bei jeder Iteration das Skalarprodukt zwischen den jeweiligen Koeffizienten der Region des Inputs und den Koeffizienten des Filters errechnet. Dieses Skalarprodukt wird nach Anwendung der Aktivierungsfunktion g in den folgenden Layer geschrieben. Ist der geschriebene Wert positiv bedeutet dies, dass das Feature, welches durch die verwendeten Filter repräsentiert wird, vorhanden ist. Ist der Wert 0 kommt dieses Feature nicht an dieser Stelle des Inputs vor. Der *Stride* entscheidet dabei wie weit der Filter nach jeder Operation bewegt wird. Bei einem Stride von $s = 1$ ergeben sich $(r - a + 1) * (c - b + 1)$ mögliche Positionen für den Filter. Allgemein ist der Output im zweidimensionalen Fall von der Größe $[(r - a + s) \times (c - b + s)]$. Ein Bild dieses Ablaufs kann dem Anhang entnommen werden (7). Dieses Bild zeigt einen Input der Größe $[10 \times 10]$ und einen Filter der Größe $[3 \times 3]$. Mit einem Stride von $s = 1$ ergibt sich als Ausgabe dieser Operationen ein Layer der Größe $[8 \times 8]$ ($[(10 - 3 + 1) \times (10 - 3 + 1)]$). Der Stride des Filters kann auch größer als 1 sein. Zudem gibt es die Option, dass man dem Bild ein sogenanntes *Padding* hinzufügt oder nicht. Für das Padding gibt es wiederum selbst verschiedene Optionen für die Realisierung. Im Falle der Anwendung von Padding errechnet sich die Größe der Ausgabe bei der Größe des Padding p und bei quadratischem Input und Filter wie folgt:

$$p = \frac{r - a + 2p}{s} + 1$$

Pooling: Ein Pooling-Layer komprimiert die Daten entlang der räumlichen Achsen. Ähnlich wie beim Convolutional-Layer gibt es beim Pooling-Layer eine Art Filter. Dieser Filter verschiebt sich um den Wert des Strides. Bei diesem Filter werden die betroffenen Elemente jedoch nicht aufsummiert, sondern die Koeffizienten, welche der Filter abdeckt, werden mit dem Max-Operator auf einen reduziert. So wird bspw. aus der Ausgabe eines Layers mit $[32 \times 32 \times 10]$, mittels eines Filters der Größe $[2 \times 2]$ und einem Stride von 2 die Ausgabe des Pooling-Layers mit $[16 \times 16 \times 10]$ erzeugt.

Fully-Connected: Dieser Layer errechnet die Scores für die jeweiligen Klassen. Im Fall von zehn Klassen ist das Resultat ein Volumen der Maße $[1 \times 1 \times 10]$ [5]. Bis hier sind die räumlichen Informationen des Bildes vollständig transformiert und übrig bleibt ein quasi eindimensionaler Vektor mit den zehn Klassen-Scores des CIFAR-10 Datensatzes.

Activation: Die Funktion g realisiert eine elementweise Aktivierungsfunktion auf den oben beschriebenen Skalarprodukten.

Mögliche nicht-lineare Aktivierungsfunktionen der Neuronen:

- Sigmoid: $g(x) = \frac{1}{1+e^{-x}}$
- Hyperbolischer Tangens: $g(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
- ReLU: $g(x) = max(0, x)$
- etc.

3.4 Deep Reinforcement Learning

In Deep Reinforcement Learning werden die Policy oder die State-Value Function oder beide zusammen mit einem tiefen NN realisiert. Das hat den Vorteil, dass bspw. das Mapping der State-Action-Value Function Q beim Q-Learning nicht mehr in einer zweidimensionalen Tabelle gespeichert wird. Hier übernimmt das Mapping von State-Action Tupel zu Value ein tiefes NN. Das spart einerseits Anforderungen an die Hardware und kann andererseits schneller eine Antwort liefern, da keine mit der Problemgröße wachsende Tabelle durchsucht werden muss.

on-policy vs. off-policy Folgend wird der Unterschied zwischen *on-policy*- und *off-policy*-Verfahren erläutert. Beide Verfahren, sowohl *on-policy*, wie auch *off-policy* aktualisieren ihre Schätzungen der Q-Werte anhand des Tupels (s', a'). Der Folgestate ist s' und die dann gewählte Action ist a'. Die Verfahren unterscheiden sich in der Annahme für die Wahl der folgenden Action a' des Tupels.

on-policy: Diese Verfahren gehen davon aus, dass die Action a' mit der aktuell verfolgten Policy π_t, zum Zeitpunkt t ausgewählt wird. SARSA ist bspw. ein on-policy-Verfahren.

[5] Die Ausgabe des vorangehenden Layers wird typischerweise zu einem eindimensionalen Vektor gestaucht. So ergibt sich aus einen Layer der Größe $[2 \times 2]$ ein Vektor der Größe $[1 \times 4]$.

off-policy: Bei diesem Verfahren wird davon ausgegangen, dass die Action-Wahl *greedy*[6] getroffen wird. Aus diesem Grund ist Q-Learning ein off-policy Verfahren.

In on-policy geschehen Updates also anhand einer Action-Wahl der aktuell verfolgten Policy. In off-policy wird davon ausgegangen, dass die Action-Wahl einer greedy Policy folgt und somit nicht der aktuell verfolgten Policy entspricht.

3.4.1 Function Approximation

Im Rahmen dieser Arbeit werden sowohl die Policy, als auch die Value Function mit einem tiefen NN approximiert.

Value Function Approximation Die State-Value Function V mapped einen State s auf einen Value. Das NN ist mit dem Vektor w parametrisiert. $V(s, w)$ approximiert damit die tatsächliche State-Value Function $V_*(s)$. Typischerweise ist die Dimensionalität des Vektors $|w|$ geringer, als die Anzahl an möglichen States $|S|$. Die Reduktion des Problems von der Anzahl an möglichen States auf die Anzahl an verwendeten Gewichten ermöglicht das agieren in komplexen kontinuierlichen Environments. Das Netzwerk wird in Richtung des approximierten erwarteten Returns $(R_{t+1} + \gamma V(S_{t+1})$, bekannt als das *TD Target*, korrigiert. Die Größe eines Updates von V wird mittels des Hyperparameters α kontrolliert.

$$V(s_t) \leftarrow V(s_t) + \alpha(R_{t+1} + \gamma V_{t+1} - V(s_t))$$

Ein Update der State-Action-Value Function Q erfolgt analog.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Der Estimator der State-Value Function ist wie folgt definiert:

$$V_w(s) = \mathbb{E}_\pi[\sum_{k=0} \gamma^k R_{t+k+1} | s_t = S]\ ^7.$$

Der Iterator k ist limitiert über die Anzahl an Schritten, die vorausgeschaut werden. Bei $TD(\lambda)$ gibt λ diese Anzahl vor. Da man am Ende einer Episode den tatsächlich erhaltenen Reward vom Environment bekommt, ist dieser Teil ein Supervised Learning Problem und kann auch ebenso optimiert werden.

Policy Approximation Policy Gradient Methoden zielen darauf ab, eine Policy direkt aus der Interaktion mit dem Environment zu modellieren und optimieren. Typischerweise wird die Policy ebenfalls als parametrisierte Funktion $\pi_\theta(a|s)$ mit den Parametern θ formuliert. Der resultierende Value der Objectiv Function ist somit abhängig von der verfolgten Policy π. Die Reward- bzw. Objectiv Function wird definiert durch folgende Gleichung:

[6] Eine greedy Wahl der Action nimmt immer die Action, welche den höchsten totalen geschätzten Reward verspricht.

[7] Der Exponent k steht für die Anzahl der Episoden.

$$J(\theta) = \sum_{s \in S} d^{\pi_\theta}(s) V^{\pi_\theta}(s) = \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \ ,$$

wobei $d^{\pi}(s)$ die Verteilung der makrov'schen Kette für die Policy π_θ darstellt. Mittels des Gradienten Aufstiegsverfahrens wird θ in die Richtung gelenkt, die der Gradient $\bigtriangledown_\theta J(\theta)$ vorschlägt, um die erwarteten Rewards zu maximieren.

Das Policy Gradient Theorem legt das theoretische Fundament für die Umformung des Gradienten $\bigtriangledown J(\theta)$ zu einem Estimator $\mathbb{E}$. [8]

$$\bigtriangledown J(\theta) = \mathbb{E}_{\pi_\theta}[\bigtriangledown log \pi_\theta(a|s,\theta) Q^{\pi_\theta}(s,a)]$$

π_θ ist eine stochastische Policy und Q^{π_θ} ist der State-Action Value Funciton Estimator.

Advantage Um die Varianz im Training zu verringern, wird statt der lediglich value-based Methoden, häufig die Advantage Function eingesetzt und statt der Value Function im Policy Gradient verwendet [SWD+17]. Der Advantage ist definiert wie folgt:

$$A(s,a) = Q_b(s,a) - V_w(s)$$

$Q(s,a)$ entspricht dem Q-Value von Action a im State s, $V(s)$ dem Value des States s. Ein Nachteil dieser Praxis ist, dass man Q und Vmit zwei unterschiedlichen Parameter-Sets trainieren muss. Dies geht aus den Indizes der vorangegangenen Gleichung hervor. Um die Notwendigkeit von zwei unterschiedlichen Parametrisierungen zu vermeiden, wird häufig die Umformung angewandt, welche den Advantage Function Estimator A durch einen Estimator des TD-Errors ersetzt. Für die tatsächliche Value Function $V^{\pi_\theta}(s)$ definiert sich der TD-Error δ^{π_θ} folgendermaßen:

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

Die Definition von $Q(s)$ (3.1) entspricht dem aktuellen Reward, plus dem geschätzten, erwarteten und diskontinuierten Value des nächsten States s'. Der erste Teil des TD-Errors $(r + \gamma V^{\pi_\theta}(s'))$, bedingt durch den aktuellen State s und die Action a, entspricht somit der State-Action-Value Function Q. Ersetzt man nun die State-Action-Value Function mit dem ersten Teil des TD-Errors, für die Schätzung der Advantage Function, erhält man folgende Gleichung:

$$\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|s,a] = \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s)$$

Somit kann man den TD-Error verwenden, um den Policy-Gradienten zu schätzen und man umgeht damit die Notwendigkeit eines extra Netzwerks bzw. Parameter-Sets für Q. Da $V^{\pi_\theta}(s)$ den Value des aktuellen States repräsentiert, kann dieser Teil außerhalb des Estimators sein. Nun kann der TD Error verwendet werden, um den Policy Gradienten zu berechnen.

$$\delta_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\bigtriangledown_\theta log \pi_\theta(s,a) \delta^{\pi_\theta}]$$

[8] Sutton und Barto geben in Sektion 13.2 [SB11]S. 268 den Beweis für den episodischen Fall.

3.5 Actor Critic

Die Ansätze in model-free RL lassen sich grob in drei Gruppen einteilen.

- Value-basierte Methoden (bspw. Q-Learning und Deep Q-Learning): In diesen Methoden lernt der Agent eine Value Function V, welche jedem State oder jedem State-Action Tupel einen Value zuweist. Diese funktionieren zuverlässig in einer Umgebung mit einer endlichen Menge an Actions bzw. States.

- Policy-basierte Methode (bspw. REINFORCE Policy Gradient): Hier lernt der Agent direkt eine Policy π, ohne dabei eine State-Value Function zu nutzen. Diese Methoden sind besonders stark in Bereichen, in denen das Environment einen stochastischen oder kontinuierlichen Action Space aufweist.

- Hybrid Methoden (Actor Critic): Diese Methodik vereint die zuvor erwähnten Ansätze zu einem, welcher sowohl eine Value Function einsetzt als auch eine Policy verwendet. Aufgeteilt auf zwei NNs approximiert der Critic eine Value Function, welche den Aktor korrigiert. Diese Methodik vereint die Vorteile der beiden vorherigen Ansätze. Das Training eines Actor Critic Modells generiert seine Samples typischerweise on-policy (3.4). Im Fall einer synchronisierten Batched *Advantage Actor Critic* (A2C) Implementierung generiert der Aktor eine Anzahl an Samples T, wobei T deutlich kleiner als die Episodenlänge ist. Daraufhin evaluiert der Critic die Samples, aktualisiert seine Schätzungen für die Values der jeweiligen States und aktualisiert die Policy des Aktors.

In dieser Arbeit wird das Actor Critic Modell verwendet.

3.6 Proximal Policy Optimization

Der Proximal Policy Optimization (PPO) [SWD+17] Algorithmus ist ein on-policy (3.4) Policy Gradient Algorithmus. Im folgenden Unterkapitel wird der in dieser Arbeit verwendete Algorithmus zur Optimierung der Policy vorgestellt.

Unabhängig von den in dieser Arbeit erwähnten Problemen in RL (4.1 Overfitting auf große Datensätze, 5.2.1 Sparse-Reward-Problem), haben on-policy Policy Gradient Methoden oft zwei weitere generelle Probleme. Einerseits besteht das Problem, dass generierte Samples bzw. Batches immer abhängig von der aktuell verfolgten Policy sind. Dadurch erlebt der Agent sich konstant ändernde State- und Reward-Verteilungen, was zu Instabilität im Training führen kann. Andererseits besteht in RL eine große Sensitivität gegenüber Hyperparameter Tuning.

Darüberhinaus besteht in RL allgemein die Gefahr zu großer Policy-Updates. Wird bspw. ein Batch, anhand dessen die Policy optimiert werden soll, unter einer zufällig schlechten Policy gesampled, so kann es passieren, dass sich die Policy mit einem Update sehr weit von der aktuell verfolgten Policy π_θ entfernt. Auch wenn die Policy

bisher zu keinen schlechten Ergebnissen geführt hat, werden nun ggf. sehr viele gute Eigenschaften der bestehenden Policy revidiert. Daraufhin wird der nächste Batch unter noch schlechteren Umständen gesampled. Solch eine falsche Korrektur der Policy zu kompensieren, kann viele Zeitschritte im Training kosten und ggf. erholt sich die Policy nicht mehr vom ersten Batch, der die Policy in diese Richtung verbessert bzw. verschlechtert hat. Das PPO-Paper [SWD+17] zeigt in Abschnitt 6.1 Experimente mit unbeschränkten Policies. Diese Experimente liefern einen empirischen Beweis für Schwierigkeiten mit unbeschränkten Updates.

3.6.1 Hintergrund

Der PPO Algorithmus ist maßgeblich durch den Trust Region Policy Optimization (TRPO) Algorithmus [SLA+15] inspiriert.

Trust Region Policy Optimization Um die Stabilität im Training zu verbessern, verhindert TRPO unter anderem die zuvor erwähnten, zu großen Policy Updates. TRPO löst dieses Problem mittels eines sogenannten *KL-Divergence Constraint*, in der Objectiv Function. Grundsätzlich versucht der Algorithmus Folgendes zu maximieren:

$$L^{TRPO}(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]].$$

Der Teil der Gleichung links ($(\pi_\theta/\pi_{\theta_{old}})A_t$) ist ein Verhältnis zwischen der neuen und der alten Policy, unter Betrachtung desselben State-Action Tupels, multipliziert mit dem Advantage Estimate. Dieser Teil stellt die unbeschränkte Policy-Optimierung. Der Hyperparameter β, stellt den sogenannten Penalty Coefficient dar. Dieser entscheidet wie stark das KL-Divergence Constraint beachtet wird. Je größer β, desto kleiner ist das resultierende Update. Der Term $KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]$ [9] errechnet die Divergenz zwischen der alten und der neuen Policy, in anderen Worten die Entfernung zwischen den zwei Policies. Dies geschieht unter Betrachtung des States s_t und allen möglichen Actions.

3.6.2 PPOs Clipped Objectiv Funciton

Sei $r_t(\theta)$ das Verhältnis der Wahrscheinlichkeitsverteilungen $\pi_\theta/\pi_{\theta_{old}}$, sodass $r_t(\theta_{old}) = 1$, so beschreibt $r_t(\theta)A_t$ das unbeschränkte TRPO Objectiv.

$$L^{CPI}(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t] = \mathbb{E}_t[r_t(\theta)A_t]$$

CPI referenziert hier auf *Conservative Policy Iteration* der Arbeit [KL02]. Das unbeschränkte TRPO Objectiv wurde in dieser Arbeit erstmalig vorgestellt. Somit ist $r_t(\theta)$ größer als 1, wenn eine Action unter der neuen Policy wahrscheinlicher ist, als unter der

[9] Kullback-Leibler-Divergence: $D_{KL}(\pi_1||\pi_2)[s] = \sum_{a\in A}\pi_1(a|s)log\frac{\pi_1(a|s)}{\pi_2(a|s)}$

alten Policy. Im umgekehrten Fall ist $r_t(\theta)$ kleiner als 1. Wie in der letzten Problematik 3.6.1im zweiten Absatz bereits erklärt, kann auch dieses Objectiv zu sehr großen Policy Updates führen. Das Update kann so groß sein, dass die Policy es eventuell nicht mehr kompensieren kann, falls es ein schlechtes Update war.

PPO greift das Objectiv von TRPO auf, doch beschränkt es sich nicht auf die aufwendig zu berechnende KL-Divergence, sondern clipped es lediglich um den Wert $1 \pm \epsilon$. ϵ ist ein Hyperparameter, welcher typischerweise $0,2$ ist. Zusammen mit dem Min-Operator kann nun das eigentliche PPO-Objectiv vorgestellt werden.

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1+\epsilon, 1-\epsilon)A_t)]$$

Diese Objectiv Function nimmt das Minimum aus zwei Termen. Der erste Term im Min-Operator ($r_t(\theta)A_t$) ist das unbeschränkte TRPO Objectiv, wie in 3.6.1 vorgestellt. Der zweite Term beschränkt das Verhältnis $r_t(\theta)$ auf eine Differenz von $1 \pm \epsilon$ und multipliziert dann mit dem Advantage Estimate, wie in 3.6.3 definiert. Die daraus resultierende Policy π_θ kann sich somit, innerhalb eines Updates, nicht weiter als ϵ von der alten Policy $\pi_{\theta_{old}}$ entfernen.

Der PPO Algorithmus vereinfacht somit das KL-Divergence Constraint zu einer Clip-Operation. Zudem ist eine Schwäche des TRPO, dass es kein β gibt, welches über mehrere Environments hinweg erfolgreich ist. Selbst in einem einzigen Environment kann ein statisches β zu einer schlechten Performance führen. Dieses Problem besteht bei PPO nicht.

3.6.3 PPO Actor Critic Style

In dieser Arbeit wird der PPO Algorithmus zusammen mit dem Actor Critic Model verwendet. Folgend wird der *Loss*-Term vorgestellt, welche PPO einsetzt. Im PPO-Paper teilen sich die Netzwerke für die Value Function und die State-Action-Value Function einen großen Teil der Parameter. Genauer teilen sie sich die Object Detection und Feature Extraction.

In dieser Arbeit sind die Object Detection und Feature Extraction durch das IMPALA-Netzwerk (3.7) gegeben. Die Architektur teilt sich im Output-Layer in zwei Heads auf. Einen für die Value Function und einen für die Policy. Aufgrund dessen, dass sich die NNs einen großen Teil der Netzwerk-Parameter teilen, kann die gesamte Architektur mittels einer Loss-Term optimiert werden.

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

Der Term $L_t^{CLIP}(\theta)$ stellt dabei das Clipped Objectiv dar, wie in 3.6.2 definiert. $L_t^{VF}(\theta)$ stellt einen Loss-Term dar, welche wie folgt definiert ist:

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{targ})^2 \ .$$

$V_\theta(s_t)$ ist eine Value Function, wie sie im Abschnitt 3.4.1 definiert ist. V_t^{targ} ist der tatsächliche Return eines Batches. Dieser Term $L_t^{VF}(\theta)$ ist somit ein Squared-Error Loss.

$S[\pi_\theta](s_t)$ ist ein Term, welcher die Entropie der Gleichung erhöht. Durch den Entropie-Term betreibt der Agent im Training anfänglich viel Exploration, bis die anderen Terme der Gleichung ausreichend Gewicht erlangen. Die Koeffizienten c_1 und c_2 sind Hyperparameter zur Gewichtung der jeweiligen Terme. Optimiert wird dann mit einem stochastischen Gradienten-Abstiegsverfahren.

Der PPO Algorithmus nutzt eine Implementierung, wie sie in [MBM+16] dargestellt ist. In dieser Implementierung sampled die Policy für T Schritte, wobei T deutlich kleiner als die Episodenlänge ist. Anhand dieses Segments wird der Agent aktualisiert. Diese Art des Samplings benötigt jedoch eine Anpassung des Advantage Estimators, da dieser ansonsten Teile der *Trajectory*. Eine Trajectory τ_H ist ein Pfad des Agenten durch den State Space des Environments bis zum Horizont H beachten würde, welche über T hinaus liegen. Das kleine t spezifiziert hier den aktuellen Zeitschritt in $[0, T]$ und das große T spezifiziert die Länge des Segments.

$$A_t = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

Im Fall von $\lambda = 1$ kann die vorangegangene Gleichung zur folgenden Gleichung vereinfacht werden. δ_t entspricht hierbei dem TD-Error zum Zeitpunkt t, wie in 3.4.1 definiert.

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

Pseudocode PPO Die Implementierung des PPO, welcher mit Segmenten von Trajectories konstanter Länge arbeitet, ist folgend als Pseudocode gegeben. Dieser Code geht von einem Fall aus, in welchem man mehrere Aktoren parallel laufen lässt und dann über die Segmente aller Aktoren optimiert. In dieser Arbeit wird jedoch nur mit einem Aktor gearbeitet ($N = 1$).

Algorithm 1 Pseudocode Implementierung des PPO, Actor Critic

```
while i < iterations do
  while actor < N do
    run policy π_θold in environment for T steps
    compute advantage estimates A_1, ..., A_T
  end while
  optimize surrogate objectiv L(θ) with K epochs and minibatches of size M ≤ NT

  θ_old ← θ
end while
```

3.7 IMPALA Architektur

Die IMPALA-Architektur wird im Paper [ESM+18] erstmalig vorgestellt. Die Abkürzung IMPALA steht für **Imp**ortance Weighted **A**ctor-**L**earner **A**rchitecture. Der wichtigste theoretische Beitrag dieses Papers besteht in der Vorstellung einer Möglichkeit zur asynchronen Separation von Aktoren und Lernern. Das ermöglicht es mehreren Aktoren simultan auf unterschiedlichen Maschinen, mit unterschiedlichen Versionen derselben Policy zu trainieren. Nach einer gewissen Anzahl von Episoden schickt der Aktor seine Samples an den Lerner. Der Lerner muss daraufhin, während des Updates, in Betracht ziehen, dass die Samples der Aktoren (ggf. alle) mit unterschiedlichen Policies generiert wurden. Aus dieser Asynchronität folgt, dass der Aktor einen Batch mit einer Policy generiert, welche seit mehreren Versionen obsolet ist. Da ein Update nicht notwendigerweise anhand der aktuellsten Policy geschieht, handelt es sich um ein off-policy Verfahren (3.4). Das in der Arbeit [ESM+18] beschriebene off-policy Actor-Critic Verfahren wird von den Autoren *V-trace* genannt. Eine genaue Definition ist in Kapitel 4 der Arbeit [ESM+18] gegeben.

Wie bereits in 3.6.3 erwähnt, wird in dieser Arbeit nur mit einem Aktor trainiert. Was aus der Arbeit [ESM+18] übernommen wird, ist die Netzwerk-Architektur mit geteiltem Object Recognition Layer und der Trennung in zwei unterschiedliche Heads. Im Vergleich zum Paper ist die Architektur für diese Arbeit mit dem Faktor 4 skaliert. Das bedeutet, dass die Architektur vier mal breiter ist, als die im Paper. Die Tiefe des Netzwerks wird durch die Skalierung nicht verändert. Ein Bild des Object Recognition Layers kann dem Anhang entnommen werden (7).

4 Experimente - Setup

Alle Experimente wurden mit der Implementierung von baselines (openAI [ope19]) realisiert. Das eingesetzte Set an Hyperparametern und die verwendete Hardware kann dem Anhang entnommen werden (Hyperparameter 7, Hardware 7)

4.1 Procgen Benchmark Umgebung

Ein generelles Problem in RL sind die zugrunde liegenden Daten bzw. die verwendeten Environments. Eine Studie von Zhan, Vinyal, Munos und Bengio [ZVMB18] zeigt, dass Agenten selbst auf großen Trainings-Sets overfitten können. Dieses Problem wird durch die prozedurale Generierung in Procgen [CHHS19] adressiert. So wird die Fähigkeit des Agenten zu Generalisieren ein wichtiger Bestandteil einer erfolgreichen Policy, wobei dieser durch den theoretisch endlosen, prozedural erzeugten Level-Stream gefördert wird. Weiter fördert Procgen durch die hohe Diversität der einzelnen Level ebenfalls die Sample Efficiency. Aus den zuvor beschriebenen Umständen sind die Environments gut geeignet, um Sample Efficiency und Generalisierung zu messen.

Die Procgen Benchmark Umgebung von openAI bietet 16 verschiedene Spiele. Inspiriert von Gym[BCP+16], ebenfalls von openAI, sind alle Spiele im Retro-Stil. Insgesamt zielt Procgen auf die folgenden Punkte ab: experimenteller Komfort, hohe Diversität innerhalb von Environments und hohe Diversität zwischen Environments [ope20]. Procgen steht open source zur Verfügung und wurde im Dezember 2019 veröffentlicht.

4.2 Chaser

Folgend wird das gewählte Spiel Chaser für die Experimente vorgestellt. Ein Bild des Spiels kann Abbildung 3 entnommen werden. Ein Blick auf den Input des Agenten bietet die Abbildung 4. Dieser ist auf $64 * 64$ Pixel herunter skaliert. Das Spiel Chaser ist von dem Atari-Klassier "Ms. Pac-Man" inspiriert. Der Agent spielt den *Jäger*. Ziel des Jägers ist es, alle auf der Karte befindlichen grünen Orbs einzusammeln. Bei Er-

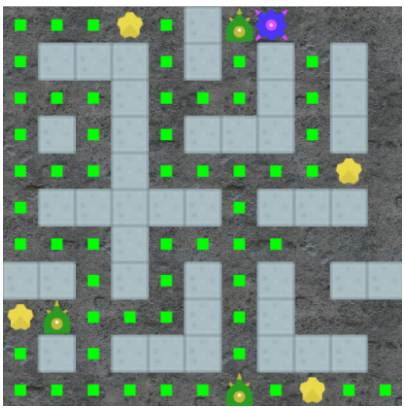


Abbildung 3: Chaser - unskaliert.

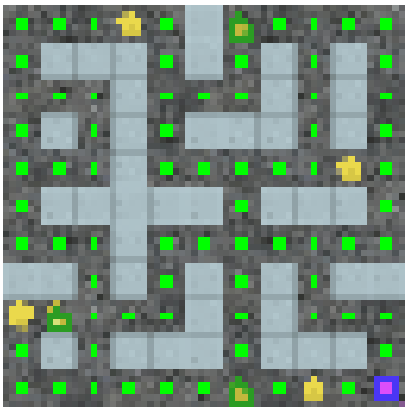


Abbildung 4: Chaser - herunterskaliert.

füllung des Ziels bekommt der Agent einen hohen extra Reward von 10 Punkten. Zum Vergleich: ein normaler grüner Orb gibt nur einen Reward von 0.04. Unterdessen muss der Agent den drei Geistern ausweichen, die sich im Spiel befinden.

Die Geister können sich in vier verschiedenen States befinden. Diese werden folgend erläutert.

- *idle-State*: Dieser State ist ein Übergangsstate, in welchem die Geister auf dem Spielfeld erscheinen. Nach t_1 Sekunden wird der Geist automatisch in den tödlich-State überführt. Dieser State ist optisch durch ein eigenes Sprite differenziert. Geister in diesem State bewegen sich nicht und können dem Jäger noch nicht schade.

- *tödlich-State*: Eine Berührung des Jägers mit einem Geist in diesem State beendet die Episode. Die Geister laufen in diesem State zufällig und ziellos.

- *verletzlich-State*: Geister können vom Jäger in diesem State gefressen werden. Sammelt der Jäger einen großen Orb ein, überführt er die Geister damit in diesen State. Dieser State ist optisch durch ein eigenes Sprite differenziert. Dieser State führt nach t_2 Sekunden wieder in den tödlich-State, wenn der Geist nicht gefressen wird.

- *killing-State*: Ist ein Geist zwei oder weniger Felder vom Jäger entfernt, so wird der Geist in diesen State überführt. In diesem State läuft der Geist nicht mehr in zufällige Richthungen, sondern zielstrebig auf den Jäger zu.

Der Action Space des Agenten ist diskret und beläuft sich auf 15 Aktionen. Dem Agenten steht dabei die folgende Menge an Actions zur Verfügung:

$$M = \{\leftarrow, \uparrow, \rightarrow, \downarrow, \leftarrow + \uparrow,\ \uparrow + \rightarrow,\ \rightarrow + \downarrow,\ \downarrow + \leftarrow,\ 'stehen\ bleiben', W, A, S, D, Q, E\}.$$

Diese beinhalten 'stehen bleiben' und 'laufen' in vier Richtungen, mit der Option eine Richtungstaste mit einer daneben liegenden zu kombinieren ($\leftarrow + \uparrow$, $\uparrow + \rightarrow$, $\rightarrow + \downarrow$, $\downarrow + \leftarrow$). Die Kombination aus zwei Richtungstasten führt jeweils zu einer eigenen Action. Bspw. führen die Richtungen 'hoch' $\uparrow$ und 'links' $\leftarrow$ zusammen zur Action 'hoch-links' $\uparrow\leftarrow$. Bewegt sich der Agent bspw. aktuell nach oben und die Action 'hoch-links' ($\leftarrow\uparrow$) wird ausgeführt, so läuft der Agent so lange 'hoch', bis sich die nächste Möglichkeit nach links zu laufen ergibt. Dann läuft er automatisch 'links'. Der Action Space beinhaltet nicht nur die vier Richtungstasten und die Kombinationen nebeneinander liegender Richtungen, sondern auch die Tasten W, A, S, D, Q und E. Hierbei sind die Tasten W, A, S und D gleich belegt wie die Tasten $\uparrow, \leftarrow, \downarrow$ *und* $\rightarrow$ (in der selben Reihenfolge). Die Tasten Q und E sind gleich belegt wie die Kombinationen $\leftarrow + \uparrow$, $\uparrow + \rightarrow$. Durch die Redundanz im Action Space mit den Buchstaben-Tasten,

lässt sich die Anzahl der möglichen Aktionen von 15 auf 9 reduzieren. Dabei werden die Tasten W, A, S, D, Q und E deaktiviert. Aus Gründen der Vergleichbarkeit zu anderen Arbeiten, wird in dieser Arbeit durchweg mit 15 Actions gearbeitet.

Das Spiel kann in drei verschiedenen Modi gespielt werden. Diese States sind: Easy, Hard und Exploration

	Easy	Hard	Exploration
totaler Orb Reward	0,04	-1,04	1,04
extra Orb Reward	0	-1	1
Fertigstellungsbonus	10	10	10
Spielfeldgröße	11x11	13x13	19x19

Tabelle 1: Übersicht über die unterschiedlichen Reward-Settings. Vergabe der Rewards für Orbs ist abhängig vom gewählten Modus.

Diese unterscheiden sich in der Größe des Spielfelds und im Reward-Systems. Die Vergabe der Rewards durch das Environment ist abhängig vom gewählten Modus und lässt sich in Tabelle 1 ablesen. Diese Arbeit verwendet ausschließlich den Modus *Easy*. In jedem Modus sind drei Gegner auf dem Spielfeld. Diese erscheinen jedes Spiel an einer zufälligen Position in unterschiedlichen Quadranten des Spielfeldes. Das Fressen eines Geistes bringt keinen Reward, ebensowenig wie das Gefressen werden.

Der State Space des Environments ist insofern variabel, dass die PCG nicht immer Level mit der selben Anzahl an freien Feldern hervorbringt. Bezogen auf die möglichen Pixel-Verteilungen ergibt sich ein theoretischer State Space von 68,7 Mrd. [10] möglichen States. Jedoch ist diese Zahl nicht die tatsächliche Größe des State Spaces, da die Berechnung der 68,7 Mrd. auf der Annahme stützt, dass jedes Pixel jede mögliche Farbe annehmen kann. Durch den Algorithmus der Level-Erstellung und die Verwendung von Sprites wird die Zahl deutlich reduziert.

Das Layout des Levels wird mit prozeduraler Generierung erstellt. Mit Hilfe des Kruskal Algorithmus [Kru56] [S.48-50] wird ein Spanning Tree über das Spielfeld gelegt, welcher alle Felder miteinander verbindet. Daraufhin werden so lange Wände entfernt, bis keine toten Enden mehr unter den Pfaden sind. Das Environment ist deterministisch, bis auf die Geister im tödlich-State. In diesem State bewegen sich die Geister zufällig durch das Level. Formal gesehen entspricht das Environment somit einem POMDP [Ast65].

Aufgrund des hohen Rewards von 10 für den letzten eingesammelten Orb, kann davon ausgegangen werden, dass ein Agent mit einem durchschnittlichen Reward ≥ 10 in der Evaluation erfolgreich besteht. Erreicht ein Agent in der Evaluation eines Checkpoint bspw. einen durchschnittlichen Reward ≥ 10, so kann man sagen, dass der Agent in der Evaluation besteht.

[10] 256 Farben pro Farbkanal, drei Farbkanäle, bei einer Bildgröße von 64 auf 64 Pixel. ($256^3 * 64 * 64 = 68.719.476.736$)

5 Experimente - Durchführung

Motivation Die in Kapitel 2 erwähnten Experimente der Arbeit [DAP+18] bieten unausgeschöpftes Potential zur Evaluation der Generalisierung eines RL-Agenten. So kann bspw. durch die Maskierung der Information der Orbs getestet werden, wie wichtig diese Information für den erfolgreichen Abschluss eines Levels ist. Darüber hinaus kann man damit ermitteln, ob der Agent etwas über die zugrunde liegende Level-Struktur bzw. den Algorithmus der die Level erstellt, lernt oder das gesamte Konzept des Spiels verstanden hat. Darüberhinaus wird untersucht, ob die Orbs dem Agenten nach einer gewissen Trainings-Dauer keinerlei wichtige Information mehr bieten.

Viele DRL-Agenten, die im Rahmen von Arcade-Spielen eingesetzt werden, bekommen den Bildschirm-Output des jeweiligen Spiels in Form eines ein- oder zweidimensionalen Arrays als Input. Somit ist alles was der Agent bisher an Stimuli erhalten hat lediglich ein Array mit ein oder zwei Dimensionen. Nimmt man nun in einer Evaluation Änderungen am Environment vor, kann das radikale Auswirkungen auf die Performance des Agenten haben.

5.1 Zu beantwortende Fragen

Unter den bisher beschriebenen Umständen wurden für diese Arbeit folgende Sets an Fragen entwickelt, die über den Verlauf der Experimente beantwortet werden.
Das erste Set reproduziert die Ergebnisse aus dem Procgen-Paper [CHHS19] und untersucht die Auswirkung von prozeduraler Level-Erstellung auf die Performance von Agenten. Dieses Set dient hauptsächlich der Bestätigung der Korrektheit der Implementierung.
Das zweite Set befasst sich mit visuellen Veränderungen der Farbe der Orbs und des Hintergrundes, aber auch mit Maskierungen gewisser visueller Information.
Das dritte Set befasst sich ausschließlich mit semantischen Veränderungen, wie das Tauschen von zwei Sprites.

Set 1 - Reproduktion und Generalisierung

- Lassen sich die Ergebnisse aus dem Paper [CHHS19] mit den eingesetzten Mitteln reproduzieren?
- Wie wirkt sich die prozedurale Erstellung und die daraus folgende hohe Diversität der Level auf die Generalisierung und die Sample Efficiency aus?

Set 2 - Farbliche Änderung

- Wie wirkt sich das Fehlen visueller Orb-Information auf die Performance aus?

- Wurde das Konzept eines Orbs verstanden? Wirken sich Änderungen der Farbe der Orbs negativ aus?
- Wie viele Level können mit der verwendeten Architektur auswendig gelernt werden? Welche Rolle spielt die visuelle Orb-Information dabei?

Set 3 - Semantische Invertierung

- Wie wirkt sich das Tauschen der Sprites von kleinen Orbs und Mauern aus?
- Wie wirkt sich das Tauschen der Sprites von großen Orbs und Geistern aus?

Das genau Setup bezüglich Dauer des Trainings oder Anzahl der trainierten Level etc. ist tabellarisch bei den jeweiligen Experimenten definiert. Generell wird die Leistung der Agenten mit einer "one-shot"-Evaluation bewertet. Das bedeutet, dass die Episode, während der Evaluation eines Levels, keine Zeitbeschränkung hat und kein Level ein zweites Mal dran kommt. Weiter wird standardmäßig auf 50 unbekannten Leveln trainiert. Änderungen an diesem Vorhaben werden in den jeweiligen Experimenten erläutert. Die Abkürzung SW steht folgend für Sliding Window.

Farbliche Distanz Fortan definiert der Begriff der *farblichen Distanz* die Entfernung zweier Farben zueinander, in einem dreidimensionalen Farbraum. Diese Distanz misst also die absolute Differenz der jeweiligen RGB-Farben. So haben die Farben Rot (r:255, g0:, b:0) und die Farbe Blau (r:0, g0:, b:255) eine farbliche Distanz von 510. Von rot zu blau muss der Rot-Wert der Farbe auf 0 und der Blau-Wert auf 255 gesetzt werden ($|-255| + |255| = 510$). So ergibt sich für die Farben Schwarz und Weiß eine farbliche Distanz von 765.

5.2 Reproduktion der Ergebnisse von Procgen

Bevor die Experimente der jeweiligen Sets evaluiert werden können, muss die Frage beantwortet sein, ob die verwendeten Implementierungen in der Lage sind, die Ergebnisse aus dem Paper zu reproduzieren, welches dem Environment zugrunde liegt. Im Paper [CHHS19] [S.7-8] wird unter dem Punkt 3.3 eine Untersuchung vorgestellt, deren RL-Agent für eine Dauer von 200 Mio. Zeitschritten in 500 Leveln trainiert wird. Dieser Agent wurde ebenfalls mit der in dieser Arbeit verwendeten IMPALA-Architektur [ESM+18] trainiert.

Das beschriebene Experiment wird in einem kleineren Rahmen, unter denselben Bedingungen durchgeführt. Genauer wird ein Agent für 80 Mio. Zeitschritte in 200 Leveln trainiert. Dies entspricht einer Skalierung der ursprünglichen Dauer und Größe des Experiments von $\frac{2}{5}$. Einen Blick auf den Input des Agenten dieses Experiments liefert Abbildung 4.

Training

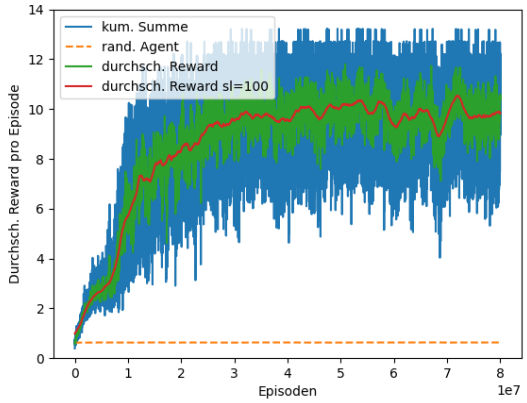


Abbildung 5: Blau: kumulierter Reward; Grün: durchschnittlicher kumulierter Reward; Rot: Grün mit SW 100; Orange: rand. Agent.

Evaluation

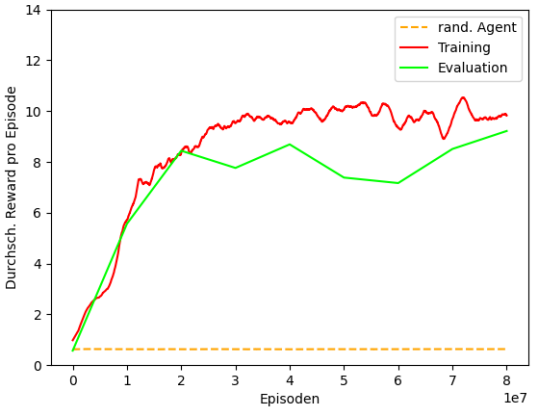


Abbildung 6: Evaluation: one-shot, 50 Level, 8 Checkpoints des Trainings.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training	80 Mio	200	normal	r:0, g:255, b:0
Evaluation	-	50	normal	r:0, g:255, b:0

Tabelle 2: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 1.

Tabelle 2 zeigt die geltenden Rahmenbedingungen während des Trainings. Die linke Abbildung (5) zeigt eine detaillierte Übersicht über das Training des Agenten. In blau wird der kumulierte Reward angegeben. Der grüne Graph gibt den durchschnittlichen kumulierten Reward an und der rote Graph zeigt den durchschnittlichen kumulierten Reward mit einem SW von 100. Die rechte Abbildung (6) zeigt eine Evaluation von acht Checkpoints über den Verlauf des Trainings. Gemessen wird die one-shot Performance des Agenten in 50 ungesehenen Leveln pro Checkpoint. Jeder Checkpoint wird in denselben 50 Leveln evaluiert. Die Evaluation findet, abgesehen von den ungesehenen Leveln, unter Trainingsbedingungen statt.

Diskussion: Ein Vergleich zwischen der Abbildung 6 und der Abbildung 7 zeigt, dass die Ergebnisse nicht nur reproduzierbar sind, sondern mit den eingesetzten Mitteln sogar übertroffen werden können. Im Paper von Procgen endet der Trainingsgraph von Chaser nach 500 Leveln, bei 200 Mio. Zeitschritten, bei einem Reward von etwa $7,2$ (vgl. 7). Beim Reproduktions-Experiment wird nach 200 Leveln bei 80 Mio. Zeitschritten, ein Reward von etwa $9,8$ erreicht. Eine mögliche Erklärung für diese Diskrepanz ist die fehlende Angabe zum eingesetzten Skalierungsfaktor der IMPALA-Architektur für dieses Experiment. Der Vergleich der Ergebnisse in Abbildung 6 mit der Abbildung 7 suggeriert jedoch, dass die Autoren an dieser Stelle eventuell einen kleineren Skalierungsfaktor, bspw. 2 statt 4, für die Architektur verwenden. Eine weitere mög-

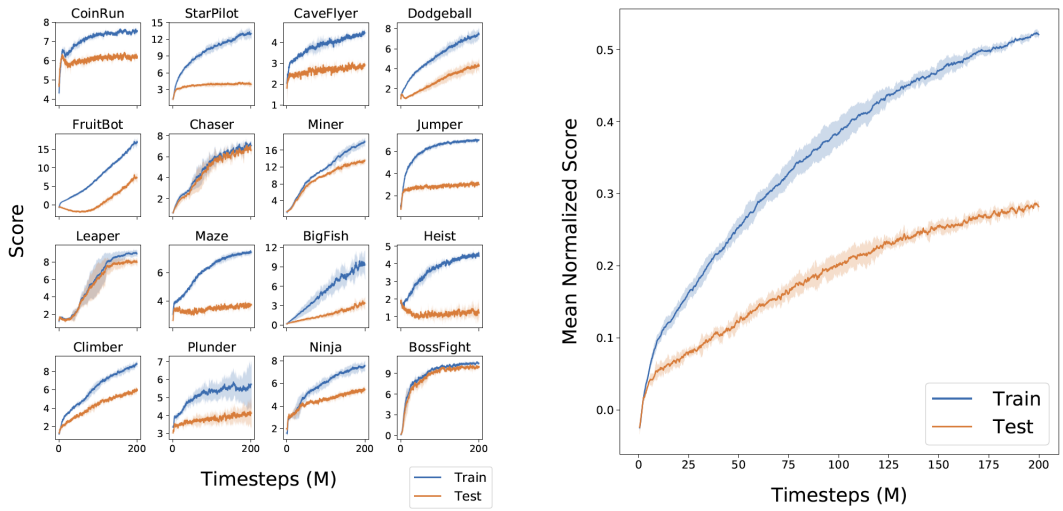


Abbildung 7: Experiment zur Generalisierung des Procgen-Papers für 200 Mio. Zeitschritte, in 500 Leveln [CHHS19].

liche Erklärung für die niedrigere Performance im Procgen-Paper könnte sein, dass über mehrere Seeds hinweg evaluiert wird und dass die Seeds der Autoren schwieriger sind, als der Seed, welcher in dieser Arbeit verwendet wird. Hierzu bietet die Arbeit [CHHS19] jedoch ebenso keinen Hinweis. Die Abbildung Chaser im Procgen-Paper (7) zeigt auf, dass nahezu keine Generalisierungslücke zwischen Training und Evaluation besteht. Das bedeutet, dass der Agent über den Verlauf dieses Trainings generalisieren kann. Dieser Fakt kann ebenfalls reproduziert werden. Abbildung 6 zeigt, dass wie im Paper fast keine Generalisierungslücke besteht. Der Leistungseinbruch bei den Checkpoints 50 Mio. und 60 Mio. ist im Rahmen der erwarteten Varianz von DRL.

Antwort: Set 1, Punkt 1 Somit kann in $\frac{2}{5}$ der Zeit und Level-Anzahl eine zufriedenstellende Reproduktion realisiert werden. Die erste Frage aus Set 1, kann also mit "Ja" beantwortet werden. Dadurch ist die Grundlage geschaffen, um weitere Experimente mit den eingesetzten Implementierungen durchzuführen.

5.2.1 Untersuchung zum Beitrag von prozeduraler Generierung bei DRL

Vergleich Level-Anzahl: 200, 1 Der folgende Paragraph vergleicht zwei Agenten, welche sich einzig in der Anzahl der im Training verwendeten Levels unterscheiden. Abbildung 8 zeigt Training und Evaluation des Agenten, welcher mit einer fixen Anzahl von 200 Leveln trainiert ist. Dies ist derselbe Agent, wie in Abbildung 5. Auf der rechten Seite ist ein weiterer Agent gezeigt. Der neue Agent ist unter denselben Umständen wie der Agent auf der linken Seite trainiert, mit dem Unterschied, dass der rechte Agent auf ein einziges Level im Training beschränkt ist.

Diskussion: Es ist eindeutig ersichtlich, dass der Agent, der nur auf einem Level trainiert wurde, im Training weitaus stabiler ist und allgemein mit einem höheren

Level-Anzahl: 200

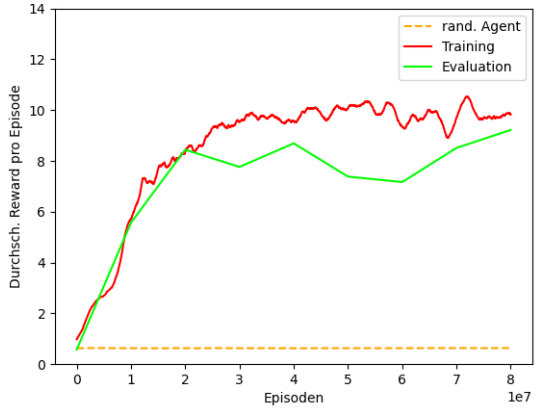


Abbildung 8: Evaluation: one-shot, Training mit fixer Level-Anzahl.

Level-Anzahl: 1

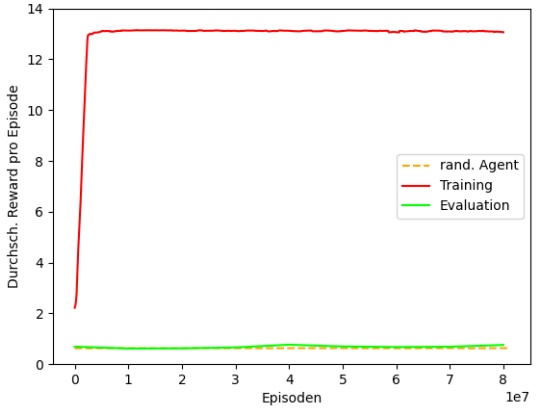


Abbildung 9: Evaluation: one-shot, Training mit unbeschr. Level-Anzahl.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Level-Anzahl 200	80 Mio	200	normal	r:0, g:255, b:0
Level-Anzahl 1	80 Mio	1	normal	r:0, g:255, b:0
Evaluation	-	50	normal	r:0, g:255, b:0

Tabelle 3: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 2.

durchschnittlichen Reward pro Episode (hier in rot) abschneidet. Das bestätigt die Annahme des Papers von Zhan, Vinyal, Munos und Bengio [ZVMB18], dass ein RL-Agent mit Leichtigkeit auf ein kleines Trainings-Set overfitted. Die Evaluation der beiden Agenten (hier jeweils in grün) zeigt jedoch ebenso klar, dass der Agent, welcher nur auf einem Level trainiert, keinerlei Generalisierung aufweisen kann. Der Agent mit 200 Trainings-Leveln hingegen weist bereits fast keine Generalisierungslücke mehr auf.

Um die zweite Frage des ersten Sets 5.1 vollständig beantworten zu können, ist ein weiteres Experiment notwendig, welches im folgenden Paragraphen erläutert wird.

Vergleich Level-Anzahl: 200, unbeschränkt In diesem Paragraphen wird der Agent, welcher mit 200 Leveln trainiert wurde, mit einem weiteren Agenten verglichen. Procgen bietet die Möglichkeit im Training verschiedene Bedingungen für die Level-Auswahl des Trainings festzulegen. Unter anderem bietet die Umgebung die Möglichkeit eine unbeschränkte Level-Anzahl zu setzen. Somit wird im Training nach positivem oder negativem Abschluss einer Episode immer ein neues, unbekanntes Level bereitgestellt. Durch diese Umsetzung soll die prozedurale Eigenschaft der Level-Erstellung maximal ausgeschöpft werden.

Der neue Agent ist ebenfalls nahezu unter denselben Umständen trainiert, wie der Agent von Abbildung 8. Dieses Mal besteht der oben beschriebene Umstand, dass die Level-Anzahl im Training unbeschränkt ist. In Abbildung 10 befindet sich wieder der

Agent mit beschränkter Level-Anzahl und Abbildung 11 zeigt den neue Agenten mit unbeschränkter Level-Anzahl.

Level-Anzahl: 200

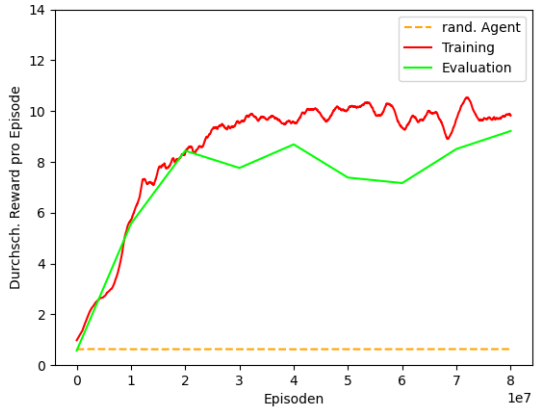


Abbildung 10: Evaluation: one-shot, auf 50 Leveln, 8 Checkpoints des Trainings

Unbeschränkt Level

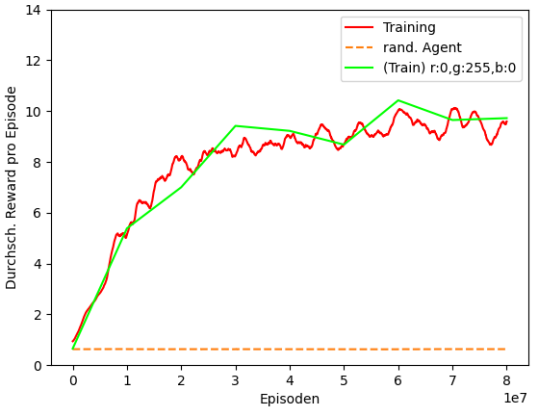


Abbildung 11: Evaluation: one-shot, auf 50 Leveln, 8 Checkpoints des Trainings

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
200 Level	80 Mio	200	normal	r:0, g:255, b:0
Unbeschr. Level	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Evaluation	-	1	normal	r:0, g:255, b:0

Tabelle 4: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 3.

Diskussion: Der direkte Vergleich der beiden Agenten zeigt eindeutig, dass die hohe intrinsische Diversität der Level-Distribution von Procgen [CHHS19][S.2] eine große Hilfe beim Erlernen von Generalisierung ist. Die Sample Efficiency ist von der prozeduralen Level-Generierung in diesem Fall weder positiv noch negativ beeinflusst. Das Procgen-Paper zeigt in seinem Experiment *500 Level Generalization* jedoch klar, dass die Auswirkungen der Eigenschaften von Procgen nicht in allen Atari-Environments so drastisch sind (vgl. siehe Abbildung 7).

Das Spiel Chaser ist durch die Anwesenheit der Orbs eine einfache Umgebung für einen RL-Agenten, welcher auf optischen Inputs beruht. Die Orbs bieten dem Agenten dadurch, dass sie eingesammelt werden können die visuelle Information darüber, wo er schon gewesen ist und wo er noch hin muss. Darüber hinaus geben die Orbs beim einsammeln einen kleinen Reward. Auch wenn der Reward sehr klein ist (0,04 im Vergleich mit 10, implizit für den letzten Orb) leidet dieses Environment nicht unter dem bekannten *Sparse-Reward-Problem*.

Bei diesem Problem bietet das Environment des Agenten über lange Zeit keinen Reward und er ist darauf angewiesen viele Aktionen auszuführen, bevor überhaupt

ein positiver Reward erwartet werden kann. Das Atari-Spiel "Montezuma's Revenge" stellt hierfür ein gutes Beispiel dar. Hier muss der Spieler häufig Schlüssel einsammeln, die zwar den Score erhöhen, jedoch gehören diese Schlüssel oft zu Türen, die mehrere Räume entfernt sein können.

Antwort: Set 1, Punkt 2 Die zweite Frage des ersten Sets (5.1) kann nun vollständig beantwortet werden. Die aus der prozeduralen Level-Generierung resultierende unbeschränkte Anzahl an möglichen Trainings-Leveln bzw. Größe der Trainings-Sets, hilft im Chaser-Environment eindeutig die Generalisierungslücke zu schließen. Jedoch muss das Ergebnis weiter mit größeren Experimenten untersucht werden, bspw. über mehrere Seeds hinweg. Das hier durchgeführte Experiment unterstützt lediglich die Hypothese, dass die intrinsische Diversität ideal für die Evaluation von Generalisierung und Sample Efficiency ist [CHHS19][S.9].

5.3 Visuelle Augmentation - Farbänderungen

Im folgenden Unterkapitel werden Experimente durchgeführt, welche visuelle Änderungen des Trainings- bzw. Evaluations-Environments untersuchen. Die Änderungen des Environments werden in den jeweiligen Experimenten dargelegt.

5.3.1 Maskierung der Orbs

Wie in Unterkapitel 4.2 erwähnt, ist es das Ziel des Agenten in Chaser alle Orbs einzusammeln. Erst dann gilt eine Episode als erfolgreich abgeschlossen. Dadurch eignen sich die Orbs für Maskierungen und Änderungen ihre Anwesenheit bzw. Erscheinung. Zu diesem Zweck wird im Environment das Hintergrundbild durch eines derselben Größe mit der RGB-Farbe (r:0, g:255, b:0) monoton ersetzt. Die gewählte Farbe entspricht der Farbe der Orbs und maskiert ihre visuelle Anwesenheit. Ein Bild des resultierenden Inputs für den Agenten kann dem Anhang 7 der Abbildung *Hintergrund grün* entnommen werden. Dem Agenten fehlt damit jede visuelle Information der Orbs. Das Reward-System bleibt weiter bestehen. Betritt der Agent ein Feld, welches er zuvor noch nicht besucht hat, bekommt er den Orb-Reward von 0,04. Das aufgeführte Experiment zeigt das Training und die Evaluation eines Agenten, welcher für 5 Mio. Zeitschritte auf einem Level trainiert. Dieses Experiment weicht von der one-shot Evaluation, wie in 5.1 definiert ab. Hier ist jeder Checkpoint 50 Mal auf dem einen bekannten Trainings-Level evaluiert.

Diskussion: Trotz des grünen Hintergrunds, welcher die visuellen Orb-Informationen maskiert, ist der Agent bereits nach 2,8 Mio. Zeitschritten in der Lage das Level im Training konsistent zu lösen. Auf die Evaluation in ungesehenen Leveln kann hier vorerst verzichtet werden, da in Unterkapitel 5.2.1 schon gezeigt wird, dass ein Agent mit

Training - 5 Mio. Zeitschritte

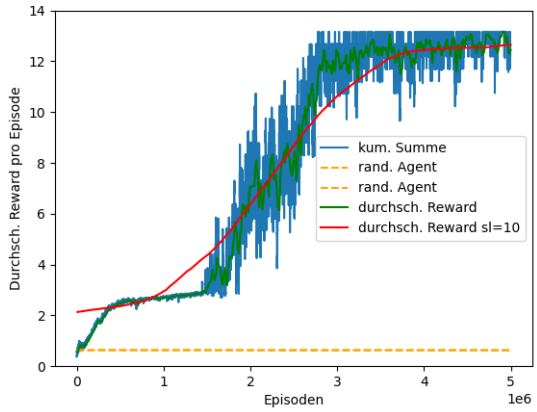


Abbildung 12: Training mit monoton grünem Hintergrund.

Evaluation

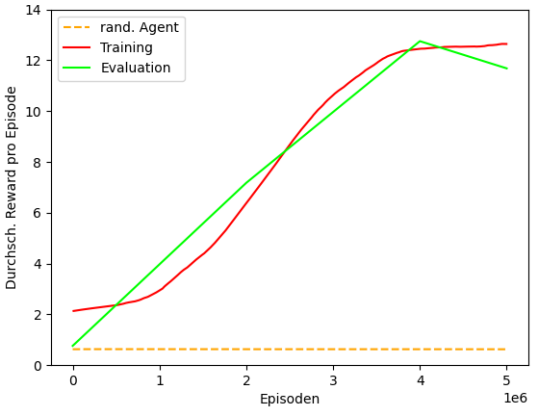


Abbildung 13: Eval: one-shot, auf bekanntem Level.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Level-Anzahl 1	5 Mio	1	r:0, g:255, g:0	r:0, g:255, b:0
Evaluation	-	1	r:0, g:255, g:0	r:0, g:255, b:0

Tabelle 5: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 4.

einem einzelnen Trainings-Level nicht generalisiert. Der Graph von Abbildung 13 zeigt starkes Overfitting auf dem Trainings-Level. Wie dieses Trainingsszenario mit maskierter Orb-Information in größerem Maßstab abläuft, wird im folgenden Paragraphen untersucht.

Vergleich Level-Anzahl: 200, unbeschr., mit grünem Hintergrund Für das folgende Experiment werden zwei Agenten, einer mit 200 Trainings-Leveln und einer mit unbeschränkter Level-Anzahl, für 80 Mio. Zeitschritte trainiert. Bei diesen Trainings ist der Hintergrund, wie im Paragraphen zuvor, monoton in derselben Farbe wie die Orbs und maskiert deren visuelle Erscheinung. Hierfür werden zwei neue Agenten trainiert.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
200 Level	80 Mio	200	r:0, g:255, g:0	r:0, g:255, b:0
Unbeschr. Level	80 Mio	unbeschr.	r:0, g:255, g:0	r:0, g:255, b:0
Evaluation	-	50	r:0, g:255, g:0	r:0, g:255, b:0

Tabelle 6: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 5.

Abbildung 14 zeigt den Agenten, welcher für 80 Mio. Zeitschritte auf 200 verschiedenen Leveln trainiert wurde. Abbildung 15 zeigt den Agenten, welcher ebenfalls für 80 Mio. Zeitschritte trainiert wurde, jedoch mit unbeschränkter Level-Anzahl.

200 Level

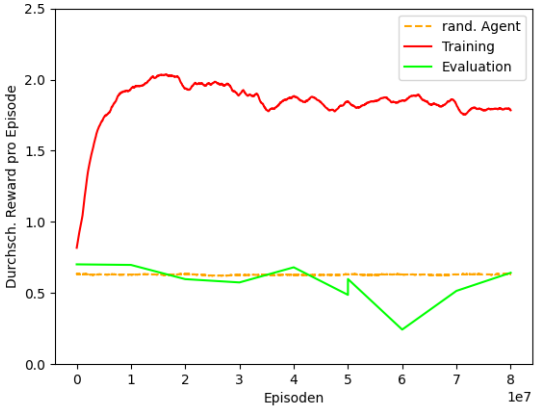


Abbildung 14: Training mit grünem Hintergrund in 200 Leveln

Unbeschränkt Level

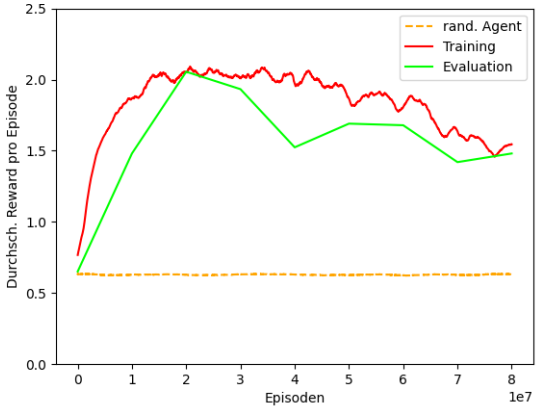


Abbildung 15: Training mit grünem Hintergrund in unbeschr. Leveln

Diskussion: Der Agent mit 200 Leveln im Training zeigt keinerlei Generalisierung. Bei 60 Mio. Zeitschritten lernt der Agent sogar etwas, was ihn schlechter als den zufällig laufenden Agenten macht. Im Kontrast dazu schließt der Agent, welcher mit einer unbeschränkten Level-Anzahl trainiert wurde, die Generalisierungslücke. Vergleicht man hingegen diese beiden Agenten mit den Agenten, welche die Orb-Information während des Trainings haben (5.2.1), kristallisiert sich die Hypothese heraus, dass die Orbs eine notwendige visuelle Information darstellen. Beide Agenten mit grünem Hintergrund, egal in wie vielen Leveln sie trainiert wurden, erreichen keine konstanten Siege im Environment. Genauer zeigt die Evaluation, dass keiner der Agenten auch nur ein Level erfolgreich abgeschlossen hat. Diese Ergebnisse werfen die Frage auf, wie ein Agent abschneidet, welcher mit der visuellen Orb-Information trainiert ist, wenn ihm in der Evaluation diese Information entzogen wird.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Eval. wie Training	-	50	normal	r:0, g:255, b:0
Eval. ohne Orbs	-	50	normal	vis. nicht vorhanden

Tabelle 7: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 6.

Um die Hypothese zu bestätigen, wird der Agent aus Experiment 5.2.1 mit unbeschränkten Trainings-Leveln wiederverwendet. Dieser Agent weist die beste Generalisierung der bisher trainierten Agenten auf und eignet sich daher am besten für dieses Experiment. Abbildung 16 zeigt die Evaluation mit und ohne visuelle Information der Orbs. Die Agenten sind beide auf denselben Leveln evaluiert.

Fehlende Orb-Information

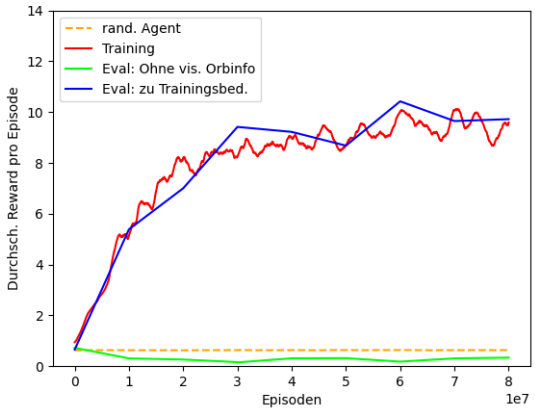


Abbildung 16: Evaluation mit und ohne visuelle Orb-Information.

Diskussion: Die beiden Graphen zeigen eindeutig, dass die visuelle Information der Orbs unentbehrlich für den Erfolg im gewählten Spiel sind. Beim Start des Trainings zeigt der Agent ohne die visuelle Information (hier grün) noch dieselbe Performance wie der Random-Agent (hier orange-gestrichelt). Bereits beim ersten Checkpoint nach 10 Mio. Zeitschritten im Training, ist seine Performance schlechter als die des Random-Agenten. Die Leistung der anderen Checkpoints oszilliert um einen Wert unterhalb dem des Random-Agenten.

Antwort: Set 2, Punkt 1 Nun kann die erste Frage des zweiten Sets (5.1) beantwortet werden. Die visuelle Information der Orbs ist notwendig, um unbekannte Level erfolgreich abzuschließen. Es ist egal, ob ein Agent mit dieser Restriktion schon im Training oder erst in der Evaluation konfrontiert wird. Die allgemeine Performance reicht kaum über die eines Random-Agentrn hinaus. Weiter zeigt Abbildung 15, dass die Performance der Agenten ab ca. 20 Mio. Zeitschritten sowohl im Trainings, als auch in der Evaluation einen Abwärtstrend aufweist.

5.3.2 Farbänderungen der Orbs

Im vergangenen Unterkapitel wird gezeigt, dass die visuelle Information der Orbs ein essenzieller Bestandteil zum erfolgreichen Abschluss unbekannter Level ist. Im folgenden Unterkapitel wird untersucht, welche Auswirkungen farbliche Änderungen der Orbs in der Evaluation auf die Performance eines Agenten haben, der keine dieser Änderungen im Training gesehen hat. Weiter wird hier untersucht, ob eine Konzeptualisierung des Orbs im Training stattfindet. Hierfür wird zunächst wieder der Agent aus Paragraph 5.3.1, mit unbeschränkter Level-Anzahl und normalem Hintergrund, weiteren Evaluationen unterzogen.

Zufälliger Wert für Grün Zuerst wird untersucht, wie die Performance in der Evaluation ausfällt, wenn dem Orb in jedem Frame ein neuer Wert für g in RGB, mitgegeben wird. Ein Frame dieser Einstellung kann dem Anhang 7 dem Bild *Orb-Farbe: gruen, zufaellig 0-255* entnommen werden. Resultierend aus der Änderung der Orb-Farbe in jedem Frame, flackert jeder Orb in verschiedenen Grüntönen.

Zufällige Grüntöne

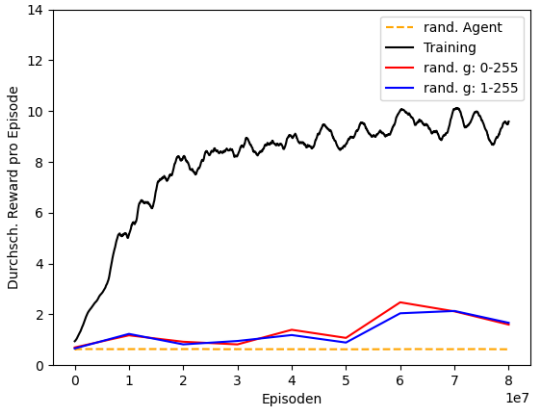


Abbildung 17: Evaluation mit zufälligen Grüntönen der Orbs

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Eval. zufälliges Grün	-	50	normal	r:0, g:0-255, b:0
Eval. zufälliges Grün	-	50	normal	r:0, g:1-255, b:0

Tabelle 8: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 7.

Der blaue Graph zeigt die Evaluation mit einem zufälligen Grünton, ohne die Option auf schwarze Orbs - der rote Graph zeigt die Evaluation mit der Option auf schwarze Orbs.

Diskussion: Die Auswertung dieser Daten legt vorerst nahe, dass keine Konzeptualisierung des Orbs stattfindet und dennoch ist der Agent bei den Checkpoints ab 60 Mio. Zeitschritte deutlich besser, als der Random-Agent. Hier muss erwähnt werden, dass die Zuweisung einer neuen, zufälligen Orb-Farbe in jedem Frame eine starke Änderung der gewohnten Pixel-Verteilung darstellt. So zeigt der Agent in beiden Evaluationen keine befriedigende Generalisierung für diese Änderungen auf. Motiviert durch dieses Experiment wird im folgenden Paragraphen untersucht, wie sich eine statische Änderung des Grün-Anteils auf die Performance auswirkt.

Grünabstufungen in 50er Schritten Zur Erinnerung, die Farbe der Orbs im Training hat die RGB-Werte (r:0, g:255, b:0). Hier wird in jeder Evaluation der grün-Wert (G-Wert) um 50 verringert, angefangen bei 250 bis 0. Die Graphen der Abbildung, außer dem Trainingsgraphen, haben alle die jeweilige Farbe, die in der Evaluation verwendet wurde.

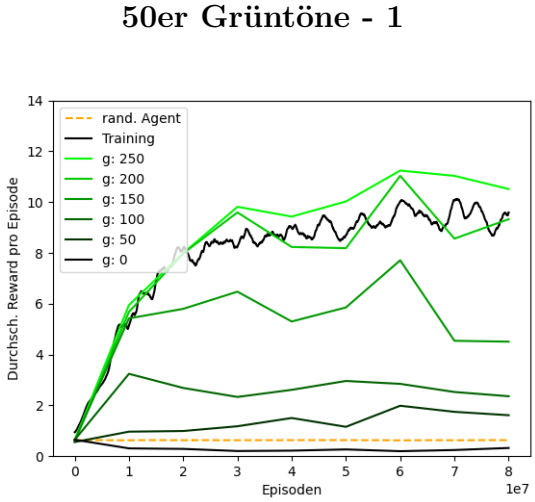


Abbildung 18: Evaluation von Grüntönen, schwärzer werdend.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Eval. Grün: 250	-	50	normal	r:0, g:250, b:0
Eval. Grün: 200	-	50	normal	r:0, g:200, b:0
Eval. Grün: 150	-	50	normal	r:0, g:150, b:0
Eval. Grün: 100	-	50	normal	r:0, g:100, b:0
Eval. Grün: 50	-	50	normal	r:0, g:50, b:0
Eval. Grün: 0	-	50	normal	r:0, g:0, b:0

Tabelle 9: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 8.

Diskussion: Die Auswertung der Abbildung 18 zeigt eine interessante These. Die Ergebnisse zeigen, dass die Performance stetig abnimmt, solange der G-Wert weiter abnimmt. Mit jeder weiteren Evaluation wird die Generalisierungslücke größer. So weisen die beiden Graphen mit einem G-Wert von 200 und 250 fast gleich gute Performance und nahezu keine Probleme bei der Generalisierung auf. Wohingegen der Graph mit einem G-Wert von 150 kaum besser als halb so gut ist, wie die zwei Graphen zuvor. Die restlichen drei Graphen mit den G-Werten 100, 50 und 0 zeigen einen stetigen Abstieg der Performance in der Evaluation. Bei einem G-Wert von 0 ist der Agent sogar schlechter als der Random-Agent.

Um die These weiter zu untersuchen, wird folgend ein ähnliches Experiment vorgestellt, in dem der G-Wert konstant bei 255 ist und sich die Werte für rot und blau von Evaluation zu Evaluation verändern. Zuvor wurden die Orbs dunkler und dunkler, bis hin zu schwarz - folgend werden die Orbs immer heller, bis hin zu weiß. Wie zuvor haben die Graphen der Abbildung 19 die Orb-Farbe der jeweiligen Evaluation. Die Farbe Weiß wird hier aus Gründen der Sichtbarkeit in Grau dargestellt.

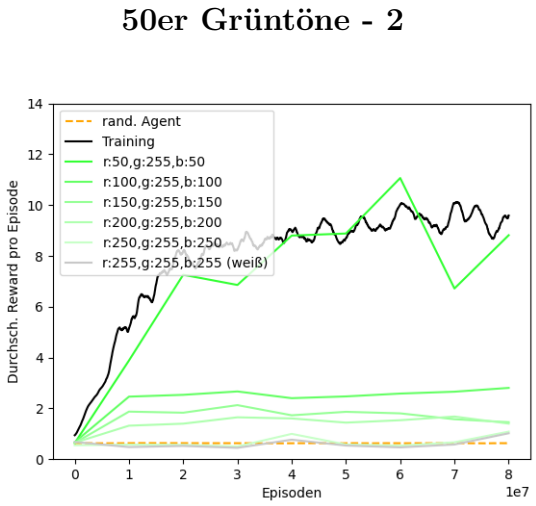


Abbildung 19: Evaluation von Grüntönen, weißer werdend.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Eval. r:50, b:50	-	50	normal	r:50, g:255, b:50
Eval. r:100, b:100	-	50	normal	r:100, g:255, b:100
Eval. r:150, b:150	-	50	normal	r:150, g:255, b:150
Eval. r:200, b:200	-	50	normal	r:200, g:255, b:200
Eval. r:250, b:250	-	50	normal	r:250, g:255, b:250
Eval. r:255, b:255	-	50	normal	r:255, g:255, b:255

Tabelle 10: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 9.

Diskussion: Die Auswertung der Abbildung 19 liefert Ergebnisse, welche die zuvor aufgestellte These vorerst widerlegen. Verglichen mit Abbildung 18 zeigt Abbildung 19 keinen stetigen Abstieg der Performance mit fortlaufender Evaluation. Stattdessen schneiden die Evaluationen mit R- und B-Werten 100 bis 255 deutlich schlechter ab, als die Evaluation mit schwärzer werdenden Orbs. Darüber hinaus zeigen die Graphen mit den R- und B-Werten (R-Werte, B-Werte) 100 bis 250 eine deutlich geringere Varianz, als die korrespondierenden Graphen der Abbildung 18. Eine mögliche Erklärung für die generell schlechtere Leistung der Graphen mit R- und B-Werten von 100 bis 250 kann

sein, dass die Abweichung der Orb-Farbe, verglichen zur Farbe im Training, an zwei Farbwerten der RGB-Farbe verändert wurde. Dementgegen wurde in der Evaluation von Abbildung 18 jeweils nur der G-Wert verändert.

Die Veränderung an zwei Stellschrauben der RGB-Farbe stellt im Kontext des gesamten Bildes eine größere Veränderung der Pixel-Verteilung dar. Aus beiden Experimenten dieses Paragraphen geht hervor, dass jene Durchläufe, bei denen die Abweichung der farblichen Distanz zu einem Input-Frame vom Training zu stark sind, generell schlecht abscheiden. So stellt eine Veränderung der Orbs um jeweils 50 in den R- und B-Werten (100 gesamt) keine Herausforderung für den Agenten aus Abbildung 19 dar. Dagegen hat eine Änderung um 105 am G-Wert, verglichen mit dem Trainingszustand des Orbs, deutlich gravierendere Auswirkungen auf die Performance (siehe Abbildung 18). Das bestätigt die intuitive These, dass Änderungen an zwei der drei Pixel-Werte und die daraus resultierende Änderung der Pixel-Verteilung größere Auswirkungen auf die Performance hat, als Änderungen an nur einem der Werte. Weiter lässt sich aus den Ergebnissen schließen, dass der Agent stark auf die Farbe des Orbs overfitted. Diese These wird im folgenden Paragraphen genauer untersucht.

Experimente mit verschiedenen Orb-Farben Für die folgenden Experimente werden zwei weitere Agenten trainiert. Einer mit roten und einer mit blauen Orbs (*Agent-Rot* und *Agent-Blau*), beide mit dem normalen Hintergrund. Bilder des jeweiligen Inputs der Agenten können dem Anhang entnommen werden (7, Titel: *Blaue Orbs*, *Rote Orbs*). Beide neuen Agenten sind für eine Dauer von 80 Mio. Zeitschritten und mit einer unbeschränkten Anzahl an Leveln trainiert. Die Agenten unterscheiden sich somit lediglich in der Farbe, welche die Orbs während des Trainings haben.

In dieser Reihe von Experimenten werden die oben beschrieben Agenten, mit dem in Unterkapitel 5.2 vorgestellten Agenten mit normalem Hintergrund und grünen Orbs (*Agent-Grün*) verglichen. Dieser Agent dient hier als Baseline für den Vergleich der Agenten. In der Evaluation wird jeder Agent zum Vergleich mit seiner Orb-Farbe wie im Training und den Trainingsfarben der beiden anderen Agenten konfrontiert. Alle Agenten haben den jeweiligen Farb-Channel auf 255 und die anderen beiden auf 0 gesetzt. So entspricht die Farbe Rot dem RGB-Wert $r : 255, g = 0, b = 0$, die anderen Farben ergeben sich analog. Somit weisen die drei Orb-Farben eine farbliche Distanz von 510 auf.

Diskussion: Im Training zeigen die unterschiedlichen Agenten sehr ähnliche Leistungen. Keiner der trainierten Agent weist eine relevante Generalisierungslücke auf. Die Abweichungen von der Baseline Agent-Grün ist im Rahmen der erwarteten Varianz, bei der Verwendung tiefer NNs. Wie zu erwarten, ist eine Änderung der Orb-Farbe während der Evaluation eine unüberwindbare Hürde für die Agenten. Dies gilt jedoch nicht für den Agent-Grün. Dieser Agent ist in der Lage mit der unbekannten Orb-Farbe Rot,

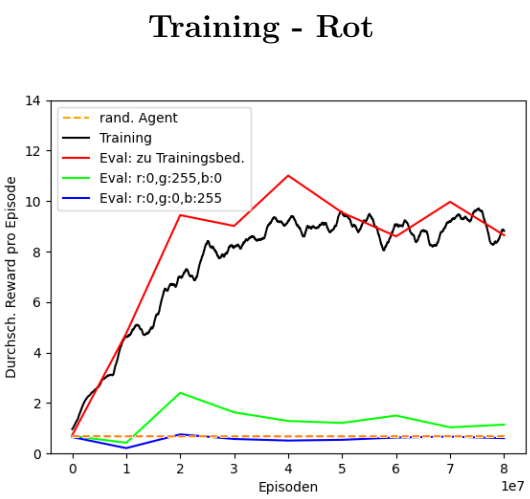


Abbildung 20: Training mit rotem Orb und normalem Hintergrund.

Abbildung 21: Training mit grünem Orb und normalem Hintergrund.

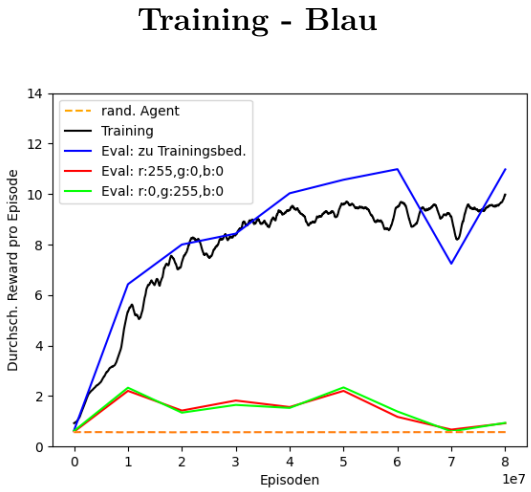


Abbildung 22: Training mit blauem Orb und normalem Hintergrund.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training rot	80 Mio	unbeschr.	normal	r:255, g:0, b:0
Training grün	80 Mio	unbeschr.	normal	r:0, g:255, b:0
Training blau	80 Mio	unbeschr.	normal	r:0, g:0, b:255
Eval. rot	-	50	normal	r:255, g:0, b:50
Eval. grün	-	50	normal	r:0, g:255, b:0
Eval. blau	-	50	normal	r:0, g:0, b:255

Tabelle 11: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 10.

eine nahezu identische Performance während der Evaluation zu erreichen. Die Evaluationen mit unbekannten Orb-Farben der Agenten Agent-Rot und Agent-Blau ist zwar weit unter der Performance der Evaluation zu Trainingsbedingungen, dennoch sind die beiden Agenten in drei von vier Fällen besser, als der Random-Agent. In Abbildung 20 (Agent-Rot) weist der Graph, welcher die Evaluation mit einem grünen Orb zeigt, eine deutliche Abweichung vom Random-Agenten auf. Die Abweichung wird in Abbildung 22 noch deutlicher. Hier ist der Agent-Blau in der Lage, mit beiden unbekannten Orb-Farben (hier rot und grün) eine noch deutlichere Abweichung zum Random-Agenten aufzuweisen. Genauer performt der Agent-Blau mit beiden unbekannten Farben nahezu gleich gut. Der Einsturz der Leistung beim Checkpoint 70 Mio. Zeitschritte spiegelt sich ebenso in der Evaluation mit der bekannten Orb-Farbe wieder und ist über ein schlechtes Update der Policy zu erklären.

In diesem Experiment werden zwar die Orb-Farben in die drei Extreme verändert (siehe Tabelle 12), jedoch ist der Hintergrund weiterhin ein farbiges Bild mit Textur (siehe Abbildung 3). Da die jeweiligen Farben der Orbs einen Farbkanal auf 255 und die anderen beiden auf 0 haben, stellen diese Farben die Grundfarbtöne von RBG dar. Um eine der drei Farben weiß zu machen, müssen die beiden Farbkanäle, welche auf 0 sind, ebenfalls auf 255 gesetzt werden. Somit weisen alle drei Orb-Farben eine farbliche Distanz zur Farbe Weiß von 510 RGB-Werten auf.

Um die Unterschiede in der farblichen Distanz der jeweiligen Orb-Farben zum Hintergrund auszuschließen, wird das hier aufgeführte Experiment mit einem weißen Hintergrund wiederholt.

Experimente mit weißem Hintergrund Im folgenden Paragraphen wird das Experiment der vorangegangenen Paragraphen mit einem weißen Hintergrund wiederholt. Dies soll Unterschiede der jeweiligen Orb-Farbe zum Hintergrund bezüglich Kontrast ausschließen und stellt sicher, dass die Farben alle die selbe farbliche Distanz zum Hintergrund haben. Hierfür werden drei neue Agenten, mit der eben erwähnten Veränderung zu den selben Bedingungen, wie im vorherigen Experiment trainiert.

Wie zuvor bezeichnet *Agent-Rot* den Agenten welcher mit einem roten Orb trainiert wird. *Agent-Grün* bezeichnet den Agenten mit grünen und *Agent-Blau* den mit blauen Orbs im Training. Ein Bild des aktuellen Agenten-Inputs kann dem Anhang den Bildern mit dem Titel *BG: weiß* entnommen werden (7).

Diskussion: Wie im oben aufgeführten Experiment, zeigen die Abbildungen 23, 24 und 25, dass die Evaluationen zu Trainingsbedingungen nahezu keine Generalisierungslücke aufweist. Weiter ist Agent-Grün nicht mehr in der Lage, mit den roten Orbs genauso hohe Rewards, wie mit den bekannten grünen Orbs zu erzielen, wie im Experiment zuvor. Hier ist sogar das Gegenteil der Fall. Der Graph der Evaluation mit rotem Orb (24) ist über jeden Checkpoint hinweg unter dem des Random-Agenten und auch

Training - Rot

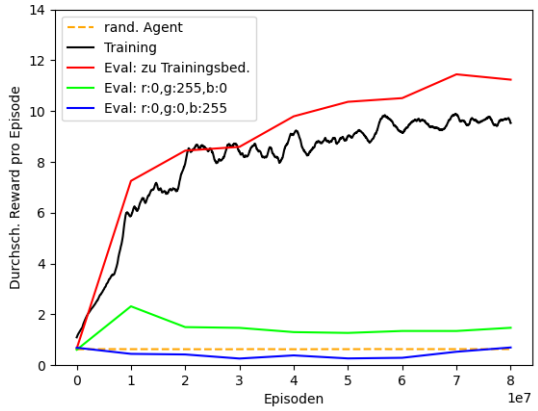


Abbildung 23: Training mit rotem Orb und weißem Hintergrund.

Training - Grün

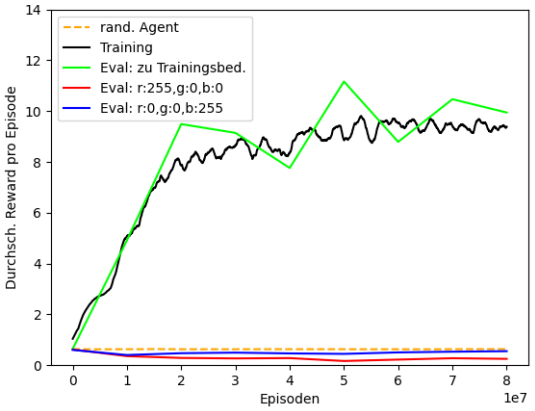


Abbildung 24: Training mit grünem Orb und weißem Hintergrund.

Training - Blau

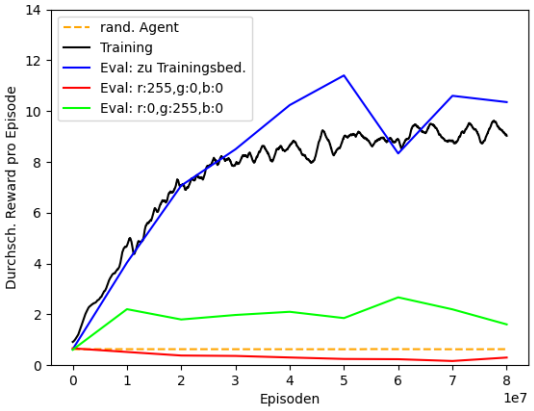


Abbildung 25: Training mit blauem Orb und weißem Hintergrund.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training rot	80 Mio	unbeschr.	weiß	r:255, g:0, b:0
Training grün	80 Mio	unbeschr.	weiß	r:0, g:255, b:0
Training blau	80 Mio	unbeschr.	weiß	r:0, g:0, b:255
Eval. rot	-	50	weiß	r:255, g:0, b:50
Eval. grün	-	50	weiß	r:0, g:255, b:0
Eval. blau	-	50	weiß	r:0, g:0, b:255

Tabelle 12: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 11.

unter dem Graphen der Evaluation mit blauem Orb. Darüber hinaus sind nicht mehr drei von vier Graphen (unterschiedlicher Farben) der Agenten aus den Abbildungen 23 und 25 besser als der Random-Agent. Lediglich die Graphen der jeweiligen Evaluation mit grünen Orbs sind besser als der Random-Agent.

Antwort: Set 2, Punkt 2 Die zwei Fragen des zweiten Punkts vom Set zur farblichen Änderung (5.1) sind mit den durchgeführten Experimenten nicht mit ausreichender Sicherheit zu beantworten. Die erste Frage kann klar beantwortet werden. Über das Training mit den geltenden Bedingungen findet keine Konzeptualisierung eines Orbs bezogen auf farbliche Änderungen statt. Die vier Experimente in Unterkapitel 5.3.2 belegen empirisch, dass jede Veränderung der visuellen Darstellung des Orbs drastische Auswirkungen auf den erhaltenen Reward des Agenten haben.

Die zweite Frage kann zwar beantwortet werden, jedoch widersprechen die Ergebnisse der initialen Intuition und werfen wiederum selbst neue Fragen auf. Die beschriebenen Experimente bestätigen, dass sich farbliche Änderungen, nahezu jeder farblichen Distanz, negativ auswirken. Allerdings kann der Agent sein Wissen in diesem einen Fall abstrahieren. So ist der Agent aus Abbildung 21 in der Lage, mit der unbekannten Orb-Farbe Rot, nahezu identische Rewards zu erzielen, wie in der Evaluation zu Trainingsbedingungen mit grünem Orb. Ein blauer Orb hingegen, welcher dieselbe farbliche Distanz zu einem grünen Orb hat, bricht die Generalisierung des Agenten in ungesehenen Leveln.

5.3.3 Untersuchung des Speichervermögens

Experiment zum Speichervermögen Folgend wird untersucht, wie viele Level ein Agent innerhalb einer begrenzten Anzahl an Zeitschritten auswendig lernen kann. Das Experiment im Anhang 7 zeigt, dass ein Agent im Training bereits nach ca. 1,2 Mio. Zeitschritten in einem Level einen Reward von ≥ 10 erreicht. Somit overfitted der Agent bereits nach 1,2 Mio. Zeitschritten stark auf das Trainings-Level. Das Experiment mit grünem Hintergrund zu denselben Trainingsbedingungen zeigt, dass das Level nach circa 2,6 Mio. Zeitschritten ebenso problemlos bestanden werden kann, auch ohne die visuelle Information der Orbs. Diese Ergebnisse motivieren die folgenden Experimente.

Dieses Experiment untersucht mit 30 Agenten, wie viele Level ein Agent im Training innerhalb von 5 Mio. Zeitschritten auswendig lernen kann. Es wird nicht dieselbe one-shot Evaluation wie zuvor angewandt. Dieses Mal wird nach jeder Episode, eines der Trainings-Level, zufällig ausgewählt und das über 50 Iterationen hinweg. Die 30 Agenten sind in zwei Sets mit je 15 Agenten aufgeteilt. Das eine Set trainiert mit dem normalen und das andere mit dem grünem Hintergrund und somit mit maskierter, visueller Orb-Information. Der erste Agent eines Sets trainiert in einem Level, der zweite

in zwei usw. Der letzte trainiert in 15 Leveln. Egal in wie vielen Leveln trainiert wird, dem Agenten stehen nur die 5 Mio. Zeitschritte zur Verfügung. Die ersten Balken von links der Abbildungen 26 und 27 repräsentiert den ersten Agenten eines Sets, der Balken ganz rechts den letzten. Jeder Balken ist in vier Abschnitte unterteilt. Der unterste Teil zeigt den ersten Checkpoint (hier rot) nach dem ersten Zeitschritt und zeigt somit ebenso einen Random-Agenten. Checkpoint 2 (hier grün) den Agenten nach 2 Mio. Zeitschritten und Checkpoint 3 (hier blau) nach 4 Mio. Zeitschritten. Der letzte (hier cyan) zeigt die abschließende Evaluation nach 5 Mio. Zeitschritten. Der Anhang bietet eine tabellarische Übersicht über die Erfolgsraten der jeweiligen Agenten (7).

Normaler Hintergrund
0 bis 15 Level

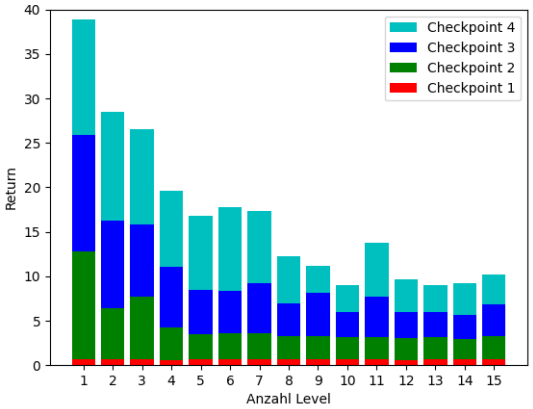


Abbildung 26: Training in 1 - 15 Leveln mit normalem Hintergrund.

Grüner Hintergrund
0 bis 15 Level

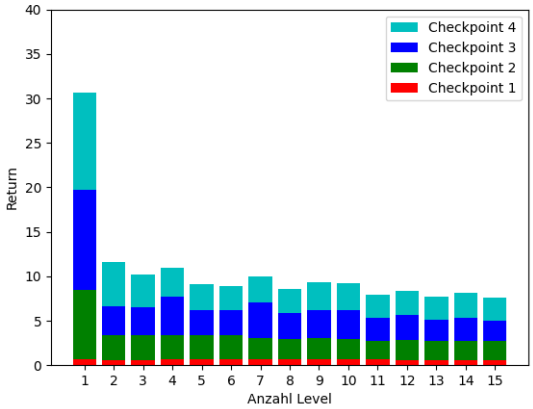


Abbildung 27: Training in 1 - 15 Leveln mit grünem Hintergrund.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Train. normaler BG	5 Mio	1 - 15	normal	r:0, g:255, b:0
Train. grüner BG	5 Mio	1 - 15	r:0, g:255, g:0	r:0, g:255, b:0
Eval. normaler BG	-	1 - 15	normal	r:0, g:255, b:0
Eval. grünem BG	-	1 - 15	r:0, g:255, g:0	r:0, g:255, b:0

Tabelle 13: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 12.

Diskussion: Der erste Agent von Abbildung 26 im direkten Vergleich mit dem ersten Agenten der Abbildung 27 zeigt einerseits, dass die Performance der jeweiligen Checkpoints mit der visuellen Orb-Information höher ist, was auf eine bessere Sample Efficiency schließen lässt. Andererseits ist der gesamte erhaltene Reward jedes Checkpoints deutlich höher, als der des Agenten mit grünem Hintergrund. Die visuelle Anwesenheit der Orbs verhilft einem Agent also nicht nur zu verbesserter Sample Efficiency, sondern verhilft ihm auch zu einer robusteren Policy. Das lässt sich dadurch erklären, dass die

Orbs, sofern sichtbar, eingesammelt werden können und nicht mehr vorhanden sind, sobald sie einmal eingesammelt sind. Das kann dem Agenten helfen eine zielführendere Strategie zu wählen. Das funktioniert jedoch nur unter der Prämisse, dass der Agent eine Vorstellung des finalen Ziels des Environments hat. Die Agenten zwei bis neun der linken Abbildung unterscheiden sich stark von denen der rechten. Die Agenten mit visueller Orb-Information sind in der Lage, bis zu drei Level innerhalb der 5 Mio. Zeitschritte auswendig zu lernen. Der Reward des vierten Checkpoints des dritten Agenten von Abbildung 26 liegt bei $10,71$. Dieser Agent schafft 76% der 50 Wiederholungen seiner Trainings-Level. Im Vergleich hierzu schafft der vierte Checkpoint des ersten Agent aus Abbildung 27 78% dieser Wiederholungen. Der dritte Agent derselben Abbildung schafft lediglich 8% der Level. Der direkte Vergleich der dritten Agenten der beiden Abbildungen legt nahe, dass die visuelle Information der Orbs ein wichtiger Faktor für den erfolgreichen Abschluss unbekannter Level ist. Hier ergibt sich eine Differenz der Erfolgsrate von 68%. Diese Ergebnisse unterstützen die des Experiments zur ersten Frage des zweiten Sets. Die Evaluation der letzten vier Agenten der beiden Sets legt nahe, dass die Performance des Agenten mit der visuellen Orb-Information weiter abnehmen wird, bis auf das Niveau der letzten Agenten aus Abbildung 27.

Antwort: Set 2, Punkt 3 Nun können auch die zwei letzten Fragen des zweiten Sets beantwortet werden. Zuerst wird die erste Frage beantwortet. Innerhalb der beschränkten Anzahl an Zeitschritten im Training kann die verwendete Architektur 3 Level auswendig lernen. Der dritte Agent aus Abbildung 26 erreicht nach 5 Mio. Zeitschritten einen durchschnittlichen Reward von $10,71$. Nach der Definition einer bestandenen Evaluation in 4.2 kann man sagen, dass der Agent die Evaluation mit drei Leveln besteht. Die Tabelle im Anhang (7) zeigt, dass dieser Agent die Level zu 76% erfolgreich besteht.

Die zweite Frage ist eindeutig zu beantworten. Die Orbs bzw. ihre visuelle Existenz ist sehr relevant für einen erfolgreichen Abschluss der Evaluation. Dieses Experiment unterstützt damit die Hypothese des zweiten Experiments in 5.3.1, dass die kleinen Orbs eine notwendige visuelle Information darstellen. Wie am Anfang dieses Experiments erwähnt, zeigt Abbildung 7 im Anhang, dass es circa 2,6 Mio. Zeitschritte benötigt, um ein Level ohne visuelle Orb-Information erfolgreich abzuschließen. Somit ist es nur logisch, dass innerhalb derselben Zeit keine zwei Level gelernt werden können. Dennoch ist die Beantwortung dieser Frage aufschlussreich. Der letzte Agent im Set mit normalem Hintergrund erreicht einen durchschnittlichen Reward, von $2,55$ über alle vier Checkpoints. Der letzte Agent des Sets mit grünem Hintergrund hat einen durchschnittlichen Reward der Checkpoints von $1,89$. Bei 15 Leveln ist der Unterschied der Performance beiden letzten Agenten der Sets nur noch bei $0,66$ ($2,55 - 1,89 = 0,66$). Die Agenten der Abbildung 26 zeigen alle den Trend, sich an den erhaltenen Reward der

Agenten der Abbildung 27 anzunähern. Jedoch zeigt sich auch, dass die Agenten mit visueller Orb-Information im Training über die 15 Evaluationen hinweg konstant besser sind als die Agenten, welche ohne diese Information trainiert sind. Ein Experiment mit einer größeren Anzahl an Zeitschritten und Leveln im Training könnte weiteren Aufschluss über visuelle Relevanz der Orbs geben.

5.4 Visuelle Augmentation - Semantische Invertierung

Im folgenden Unterkapitel sind Experimente aufgeführt, welche sich mit visueller semantischer Invertierung beschäftigen. Das bedeutet, dass die Mechanik von optisch vertauschten Spielelementen weiterhin bestehen bleibt. Vertauscht man auf diese Weise die Semantik von bspw. der Mauer und den kleinen Orbs, kann der Agent nun, zumindest optisch, Mauern einsammeln und kleine Orbs bilden das Labyrinth. Somit sind alle Änderungen an Chaser in diesem Kapitel ausschließlich visueller Natur. Ein Bild des veränderten Environments kann dem Anhang entnommen werden (7, Titel: *Default Spiel mit Orb-Sprite*). Im folgenden Kapitel wird wieder one-shot evaluiert, wie in 5.1 definiert. Das bedeutet, dass die beide Evaluationen auf denselben 50 Leveln stattfinden. Da die Orbs im Spiel Chaser als farbliche Quadrate ohne Textur implementiert sind, muss für dieses Experiment ein neuer Agent verwendet werden. Hierbei ist die Darstellung des Orbs durch ein Sprite ersetzt. Dies ermöglicht das Tauschen der beiden Sprites.

Invertierung - Mauer und kleine Orbs Der folgenden Paragraph beschreibt Experimente, die sich mit der visuellen semantischen Invertierung der Mauer und der kleinen Orbs beschäftigen. Hierfür wird das optische Erscheinungsbild dieser beiden Spielelemente vertauscht. Der rote Graph der Abbildung 28 zeigt die Evaluation zu Trainingsbedingungen, der schwarze zeigt das Training. In dunklem Blau ist die Evaluation mit vertauschter Semantik. Wie zuvor zeigt die orangene gestrichelte Linie den Random-Agenten. Ein Bild des Agenten-Inputs kann dem Anhang entnommen werden (7, Titel: *Sprite-Tausch: kleiner Orb u. Wand*).

	Zeitschritte	Anzahl Level	Invertierung
Training	80 Mio	unbeschr.	keine
Eval. zu Trainingsbed.	-	50	keine
Eval. Invertierung	-	50	Maucher u. kl. Orbs

Tabelle 14: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 13.

Diskussion: Die Evaluation zu Trainingsbedingungen (Abbildung 28) unterstützt die bisher gezeigten Ergebnisse. Ein Agent schafft es innerhalb von 80 Mio. Zeitschritten und einer unbeschränkten Level-Anzahl im Training die Generalisierungslücke zu

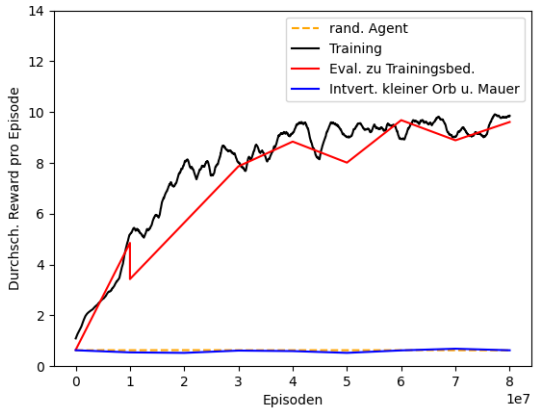


Abbildung 28: Evaluation zu Trainingsbed. und mit Invertierung der Mauer und kl. Orbs

schließen. Diese Evaluation bestätigt die Ergebnisse des Experiments in 5.2.1. Dagegen schafft der Graph der Evaluation mit semantischer Invertierung schafft es zu keinem Zeitpunkt über den des Random-Agenten hinaus. Eine mögliche Erklärung hierfür wäre, dass diese Änderung der Pixel-Verteilung zu stark ist. Im Verhältnis gibt es am Anfang eines Levels mehr kleine Orbs als Mauern. Vertauscht man nun diese beiden Sprites ist das Verhältnis umgedreht, wodurch sich die Pixel-Verteilung eines Frames stark verschiebt. Aufgrund dieser drastischen Änderung kann die Feature Extraction Pipeline die korrekten Features nicht mehr bereitstellen, um ein unbekanntes Level erfolgreich abzuschließen. Der folgende Paragraph beschreibt ein Experiment welches darauf aus ist, trotz semantischer Invertierung die Verschiebung der Pixel-Verteilung möglichst klein zu belassen.

Antwort: Set 3, Punkt 1 Die Antwort auf diese Frage ist klar. Der Agent ist zu keinem Zeitpunkt besser als der Random-Agent. Somit ist die visuelle semantische Invertierung der Sprites des kleinen Orbs und der Mauern eine unüberwindbare Hürde für den Agenten. Der Agent ist mit seinem Grad an Generalisierung nicht in der Lage, in der Evaluation zu assimilieren.

Invertierung - Geist und gr. Orbs Dieser Paragraph beschreibt Experimente mit der visuellen semantischen Invertierung der großen Orbs und der Geister. Hier wird derselbe Agent wie im Paragraphen zuvor verwendet. Dieses Experiment ist darauf aus die Veränderung der Pixel-Verteilung möglichst klein zu belassen. Da es sowohl drei Geister als auch drei große Orbs gibt, bieten sich diese beiden Spielelemente an, um auf diese Weise vertauscht zu werden. Wie in 4.2 definiert haben die Geister in Chaser vier verschiedene States. Zwei dieser States (killing- und tödlich-State) haben Animation aus drei Sprites. Um die Pixel-Verteilung dieser Invertierung möglichst gering zu halten,

sind die Geister auf ein einziges Sprite, während dieser beiden States, restriktiert. Die Sprites der anderen beiden States (idle-, verletzlich-State) bleiben unverändert. Somit stellt diese Invertierung keine Veränderung der Pixel-Verteilung des Agenten-Inputs dar. Wie zuvor stellt der schwarze Graph das Training dar und der rote die Evaluation zu Trainings-Bedingungen. Training und Evaluation zu Trainings-Bedingungen sind dieselben Graphen wie in Abbildung 28.

Ein Bild des Agenten-Inputs kann dem Anhang entnommen werden (7, Titel *Sprite-Tausch: großer Orb u. Geist*).

Invertierung Mauer u. kleine Orbs

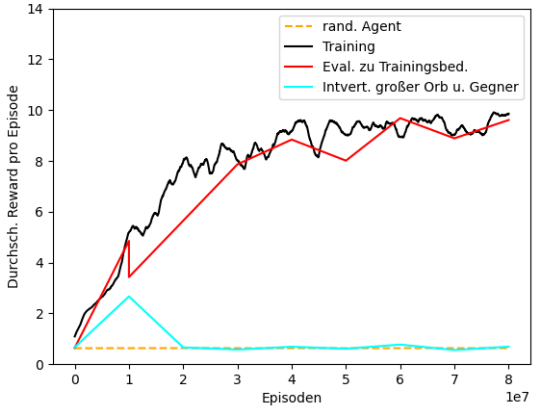


Abbildung 29: Evaluation zu Trainingsbed. und mit Invertierung der Geister und gr. Orbs

	Zeitschritte	Anzahl Level	Invertierung
Training	80 Mio	unbeschr.	keine
Eval. zu Trainingsbed.	-	50	keine
Eval. Invertierung	-	50	Geister u. gr. Orbs

Tabelle 15: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 14.

Diskussion: Die Evaluation mit der semantischen Invertierung der Geister und den großen Orbs (hier in cyan) entspricht der Intuition, die das vorangegangene Experiment suggeriert. Wie zuvor oszilliert die Performance des Agenten der Checkpoints 20 bis 80 Mio. eng um den Random-Agenten. Lediglich der Checkpoint bei 10 Mio. Zeitschritten weist einen Reward, höher als der des Random-Agenten, in Höhe von $2,67$ auf. Dieser Checkpoint schafft 10% der Evaluations-Level. Diese semantische Invertierung ist in diesem Environment die mit den geringsten Veränderungen der Pixel-Verteilung und dennoch erreicht der Agent keinen durchschnittlichen Reward ≥ 10.

Antwort: Set 3, Punkt 2 Die Antwort dieses Experiments ist ebenso klar zu beantworten, wie die des vergangenen Experiments. Auch wenn der Reward des Agent beim Checkpoint bei 10 Mio. Zeitschritten überhalb dem des Random-Agenten ist, so liegen alle anderen Checkpoints nahe um den Reward des Random-Agenten. Der Agent scheitert somit eindeutig in der Evaluation. Eine mögliche Erklärung für den Ausreißer bei 10 Mio. ist, dass der Object Recognition Layer zu diesem Zeitpunkt im Training noch nicht so stark overfitted, wie bei den folgenden Checkpoints. Aufgrund dessen ist der Agent in der Lage in dieser veränderten Umgebung zu assimilieren und zielführende Entscheidungen zu treffen.

6 Ausblick

Über die Ausarbeitung der Experimente sind an einigen Stellen Fragen aufgekommen, welche hier aufgegriffen werden. Die Experimente in Unterkapitel 5.2.1 zeigen, dass eine Generalisierungslücke entsteht, wenn man auf einer beschränkten Anzahl an Leveln trainiert. Trainiert man einen Agenten für dieselbe Dauer mit einer unbeschränkten Anzahl an Trainings-Leveln, wird diese Lücke geschlossen. Da alle Experimente dieser Arbeit lediglich mit einem Seed für die prozedurale Generierung ausgeführt sind, kann die Aussage, dass die prozedurale Generierung hilft die Lücke zu schließen nur für diesen einen Seed beantwortet werden. Ein ähnliches Experiment über mehrere Seeds hinweg würde diese Aussage bekräftigen und die Hypothese der Autoren von Procgen unterstützen, dass die Procgen-Environments gut geeignet sind, um die Generalisierung und Sample Efficiency eines Algorithmus oder einer Architektur zu evaluieren.

In den Experimenten zur Beantwortung der Fragen aus Set 2, Punkt 2 (5.1), ist der Agent von Abbildung 21 in der Lage mit einer ihm unbekannten Orb-Farbe nahezu identische Rewards zu erzielen, wie mit der ihm bekannten Farbe. Diese Ergebnisse werden im darauffolgenden Experiment unter Bedingungen untersucht, die diesen Fakt genauer darstellen sollen. Die Ergebnisse können jedoch nicht reproduziert werden. Das bringt die Frage auf, ob nahezu identischen Rewards mit zwei Orb-Farben lediglich Zufall sind oder ob hier eine unbekannte Korrelation zwischen den beiden Farben besteht, die dem Agenten geholfen hat. Es wäre auch denkbar, dass eine farbunabhängige Korrelation zwischen den Durchläufen besteht, die dem Agenten zu dieser Performance verhilft.

Das Experiment in 5.3.3 untersucht, wie viele Level mit der verwendeten Architektur auswendig gelernt werden können und welche Auswirkungen die visuelle Information der Orbs dabei hat. Hierbei zeigt sich der Trend, dass die Agenten, welche mit visuellen Orb-Information trainiert sind zwar konstant mehr Rewards bekommen, sich jedoch den Rewards der Agenten ohne visuelle Orb-Information mehr und mehr anpassen. Dieses Experiment in größerem Maßstab könnte zeigen, ob sich der Reward des Agenten mit visueller Information der Orbs, asymptotisch an die Rewards der anderen Agenten annähert oder ob die visuelle Orb-Information eine höhere Untergrenze setzt. Das würde die Hypothese aus 5.3.1 unterstützen, dass die Orbs eine notwendige bzw. sehr hilfreiche visuelle Information im Chaser-Environment darstellen.

Die Experimente zur visuellen semantischen Invertierung (5.3) zeigen eindeutig, dass diese Veränderung die Performance der Agenten auf die des Random-Agenten abfallen lässt. Videoanalysen der Evaluation beider Invertierungen zeigen, dass die Agenten zu keinem Zeitpunkt zielführende Entscheidungen treffen können. Daraus kann man schließen, dass die Objekt-Detektion und Objekt-Erkennung der Agenten stark auf die jeweiligen Texturen und Farben der vertauschten Sprites overfitted und die Mechanik der Spielelemente wenig bis keine Auswirkung auf Erkennung und Detektion

der Spielelemente hat. Hier ist interessant, ob die Änderung der Pixel-Verteilung von Training zu Evaluation durch mindern der visuellen Komplexität des Agenten-Inputs, semantische Invertierungen in der Evaluation erlauben würde. Eine Möglichkeit die visuelle Komplexität zu mindern wäre alle Texturen in Chaser durch einfarbige, solide Farben zu ersetzten. Die Feature-Extraction-Pipeline hätte somit lediglich farbliche Unterscheidung zwischen den Objekten zur Verfügung. Das könnte der Pipeline erlauben Features aus anderen Information, wie bspw. Bewegungsmuster oder Mechaniken von Spielelementen zu extrahieren.

Des Weiteren kann man hinterfragen, ob der Agent des Experiments zur semantischen Invertierung nur scheitert, da die Änderung in der Evaluation unbekannt ist. Darauf aufbauend kann man untersuchen, ob ein Agent mit ausreichend Zeitschritten im Training jede Änderung des Environments lernen kann. Würde man in Chaser nach jeder Episode dem Orb eine zufällige andere Farbe zuweisen, kann man hiermit erneut untersuchen, ob der Agent die Evaluation mit mehreren Orb-Farben besser besteht als in dieser Arbeit.

7 Konklusion

Ziel dieser Arbeit ist die Untersuchung der Generalisierung des eingesetzten IMPALA-Netzwerks 3.7 im Zusammenspiel mit dem verwendeten PPO Algorithmus 3.6 in der Procgen-Umgebung [CHHS19]. Dies geschieht anhand unterschiedlicher visueller Augmentationen des Trainings und/oder der Evaluation. Für die Untersuchung werden drei Sets an Fragen verwendet, welche sich jeweils mit unterschiedlichen Thematiken befassen.

Das erste Set untersucht die Auswirkung der prozeduralen Level-Generierung auf die Generalisierung des Agenten. Hier wird vorab die verwendete Implementierung auf ihre Richtigkeit überprüft und mit den Ergebnissen des Procgen-Papers ([CHHS19]) verglichen. Der Vergleich zeigte, dass die Implementierung im Stande ist die Ergebnisse des Papers für das Chaser-Environment zu reproduzieren. Das beantwortet die erste Frage des ersten Sets aus 5.1 mit "Ja". Die Untersuchung zur Auswirkung der prozeduralen Level-Generierung vergleicht zwei Agenten, wobei einer mit einer beschränkten Anzahl (von 200 Leveln) und der andere mit einer unbeschränkten Anzahl an Leveln, beide für 80 Mio. Zeitschritte, trainiert werden. Der Agent mit einer fixen Anzahl an Trainings-Leveln generalisiert zwar ausreichend, um bei der Evaluation in unbekannten Leveln zu bestehen, jedoch weist die Evaluation eine Generalisierungslücke zwischen Training und Evaluation auf. Der Agent mit unbeschränkt vielen Trainings-Leveln ist in der Lage diese Lücke zu schließen. Die Auswirkungen der prozeduralen Generierung und der daraus resultierende, theoretisch endlose Level-Stream für den Agenten, sind somit eindeutig hilfreich, um in unbekannten Leveln besser abzuschneiden.

Das zweite Set befasst sich mit Farblichen Änderungen der kleinen Orbs und dem Hintergrund des Spiels. Hierbei wird untersucht, welche Auswirkungen die visuelle Information der kleinen Orbs auf die Performance hat und wie robust die Generalisierung eines Agenten gegenüber Farbänderungen der kleinen Orbs ist. Weiter wird in diesem Set untersucht, wie viele Level die verwendete IMPALA-Architektur (3.7) auswendig lernen kann und welche Rolle die visuelle Orb-Information dabei spielt. Die Beantwortung der ersten beiden Punkte im zweiten Set zeigen auf, dass die Einstellung, welche die Agenten hervorbrachte und welche am besten generalisiert haben, keine Robustheit gegenüber farblichen Änderungen der kleinen Orbs aufweisen. Des Weiteren wird hier gezeigt, dass die visuelle Information des Orbs eine notwendige Hilfe bietet um in der Evaluation erfolgreich zu bestehen. Die Experimente zum Auswendiglernen der Level zeigen, dass die verwendete Architektur in der Lage ist, in 5 Mio. Zeitschritten 3 Level erfolgreich auswendig zu lernen. Maskiert man jedoch im Training und in der Evaluation die Orbs visuell, so kann lediglich ein Leveln gelernt werden. Das und der konstant niedrigere Reward der Agenten ohne visuelle Orb-Information aus Abbildung 27, unterstützt die Hypothese aus 5.3.1, dass die Orbs eine notwendige visuelle Information darstellen, um das Chaser-Environment erfolgreich abzuschließen.

Allgemein zeigen die Ergebnisse dieses Sets, dass die eingesetzte Implementierung stark auf Farben overfitted und wenig bis garnicht robust gegenüber farblichen Änderungen ist. Die Ergebnisse der Experimente aus Unterkapitel 5.3.2 untermauern diese Aussage.

Das dritte Set untersucht die Auswirkungen visueller semantischer Invertierungen, verschiedener Spielelemente während der Evaluation. Hierfür werden in einem Experiment die Sprites von Mauer und kleinem Orb vertauscht und in einem weiteren die Sprites des großen Orbs und der Geister. Die Invertierung von Mauer und kleinem Orb stellt dabei eine größere Veränderung der Pixel-Verteilung dar, als die Invertierung von großen Orbs und Geistern. Die prozedurale Level-Generierung bringt immer mehr kleine Orbs als Mauern auf das Spielfeld. Eine Invertierung dieser beiden Sprites vertauscht somit auch das Verhältnis, wie oft jedes Spielelemente auf dem Spielfeld vorkommt. Die Invertierung von großen Orbs und Geistern stellt im Gegensatz zu der anderen Invertierung keine Veränderung der Pixel-Verteilung dar, da Geister und große Orbs im Verhältnis 1:1 vorkommen. Von beiden gibt es jeweils drei. Jedoch zeigen die Ergebnisse der Experimente, dass die Einstellung, die die Agenten hervorbringt, welche die beste Generalisierung aufweisen, bei dieser Art der Invertierung versagen. Eine Videoanalyse der Evaluation beider Invertierung zeigt ein interessantes Verhalten. In beiden Experimenten hingen die Agenten die meiste Zeit einer Episode in einem Eck des Labyrinths und sind kaum von der Stelle weggekommen, an der sie in der Episode erscheinen sind. Das erklärt die Rewards ähnlich dem Random-Agent. Bei der Analyse der Evaluation wird jedoch ersichtlich, dass sich der Agent aus dem Experiment zur Invertierung von großen Orbs und Geistern nicht davor scheut einen großen Orb alias einen Geist, welcher sich mit dem Bewegungsmuster der Geister auf ihn zubewegt, entgegen zu gehen. Wohingegen ein Geist alias ein großer Orb, welcher stationär an einer Stelle ist, potentiell gemieden wird.
Zusammen mit den Ergebnissen des vorangegangenen Sets kann man davon ausgehen, dass die Feature-Extraction-Pipeline stark auf die Texturen der jeweiligen Spielelemente overfitted und eine Invertierung zweier Sprites, selbst wenn sie im Verhältnis 1:1 vorkommen, eine unüberwindbare Hürde darstellen.

Wie in Kapitel 2 bereits erwähnt, besteht in RL generell das Problem des Overfittings und die daraus resultierende schlechte Generalisierung. Die prozedurale Generierung von Procgen schafft es im Chaser-Environment die Generalisierungslücke zu schließen 5.2.1. Der Agent mit unbeschränkter Level-Anzahl im Training zeigt zu seinen bekannten Trainingsbedingungen, dass nahezu jedes unbekannte Level beim ersten Versuch geschafft wird. Verändert man jedoch die Farbe eine Spielelements bspw. der Orbs, fällt die Performance auf die des Random-Agenten herab. Diese Generalisierung findet somit nahezu ausschließlich innerhalb der ihm bekannten Pixel-Verteilung des Trainings statt. Das stellt eine weitere Problematik in Simulatoren dar. Arbeiten wie [RGY+20] bieten die Möglichkeit die Pixel-Verteilung im Training, durch verschiedene visuelle Augmen-

tationen mit einer höheren Diversität zu versehen. Somit kann die Problematik der Generalisierung in Procgen von zwei unterschiedlichen Aspekten angegangen werden. Einerseits mit der prozeduralen Generierung, welche theoretisch unendlich viele Versionen des gleichen Levels bereitstellen kann. Andererseits würde mit einem Ansatz wie in [RGY+20] oder [ZWP18] die visuelle Diversität des Environments erhöht werden. Diese beiden Ansätze schaffen Grundlagen für systematische Generalisierung.

Literaturverzeichnis

[Ast65] Astrom, Karl J.: Optimal control of Markov processes with incomplete state information. In: *Journal of mathematical analysis and applications* 10 (1965), Nr. 1, S. 174–205

[B+09] Bengio, Yoshua u. a.: Learning deep architectures for AI. In: *Foundations and trends® in Machine Learning* 2 (2009), Nr. 1, S. 1–127

[BCP+16] Brockman, Greg ; Cheung, Vicki ; Pettersson, Ludwig ; Schneider, Jonas ; Schulman, John ; Tang, Jie ; Zaremba, Wojciech: *OpenAI Gym.* 2016

[Bel57] Bellman, Richard: A Markovian Decision Process. In: *Indiana Univ. Math. J.* 6 (1957), S. 679–684. – ISSN 0022–2518

[BNVB13] Bellemare, Marc G. ; Naddaf, Yavar ; Veness, Joel ; Bowling, Michael: The arcade learning environment: An evaluation platform for general agents. In: *Journal of Artificial Intelligence Research* 47 (2013), S. 253–279

[CHHS19] Cobbe, Karl ; Hesse, Christopher ; Hilton, Jacob ; Schulman, John: Leveraging Procedural Generation to Benchmark Reinforcement Learning. In: *arXiv preprint arXiv:1912.01588* (2019)

[DAP+18] Dubey, Rachit ; Agrawal, Pulkit ; Pathak, Deepak ; Griffiths, Thomas L. ; Efros, Alexei A.: Investigating human priors for playing video games. In: *arXiv preprint arXiv:1802.10217* (2018)

[DDS+09] Deng, J. ; Dong, W. ; Socher, R. ; Li, L.-J. ; Li, K. ; Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*, 2009

[ESM+18] Espeholt, Lasse ; Soyer, Hubert ; Munos, Remi ; Simonyan, Karen ; Mnih, Volodymir ; Ward, Tom ; Doron, Yotam ; Firoiu, Vlad ; Harley, Tim ; Dunning, Iain u. a.: Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: *arXiv preprint arXiv:1802.01561* (2018)

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning.* MIT Press, 2016. – `http://www.deeplearningbook.org`

[GRM+18] Geirhos, Robert ; Rubisch, Patricia ; Michaelis, Claudio ; Bethge, Matthias ; Wichmann, Felix A. ; Brendel, Wieland: ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In: *arXiv preprint arXiv:1811.12231* (2018)

[JCD+19] Jaderberg, Max ; Czarnecki, Wojciech M. ; Dunning, Iain ; Marris, Luke ; Lever, Guy ; Castaneda, Antonio G. ; Beattie, Charles ; Rabinowitz, Neil C. ; Morcos, Ari S. ; Ruderman, Avraham u. a.: Human-level performance in 3D multiplayer games with population-based reinforcement learning. In: *Science* 364 (2019), Nr. 6443, S. 859–865

[KL02] Kakade, Sham ; Langford, John: Approximately optimal approximate reinforcement learning. In: *ICML* Bd. 2, 2002, S. 267–274

[KNH] Krizhevsky, Alex ; Nair, Vinod ; Hinton, Geoffrey: CIFAR-10 (Canadian Institute for Advanced Research). http://www.cs.toronto.edu/~kriz/cifar.html

[Kru56] Kruskal, Joseph B.: On the shortest spanning subtree of a graph and the traveling salesman problem. In: *Proceedings of the American Mathematical society* 7 (1956), Nr. 1, S. 48–50

[Mau] Maucher, Johannes: *Einführung in die Künstliche Intelligenz - Kapitel 6, Teil 4, S. 51*

[Mau20] Maucher, Johannes: *Notebook Convolutional Neural Network*. https://gitlab.mi.hdm-stuttgart.de/maucher/KI/-/blob/master/nb/N03ConvolutionNeuralNetworks.html. Version: Januar 2020, Abruf: 2020/07/21

[MBM+16] Mnih, Volodymyr ; Badia, Adria P. ; Mirza, Mehdi ; Graves, Alex ; Lillicrap, Timothy ; Harley, Tim ; Silver, David ; Kavukcuoglu, Koray: Asynchronous methods for deep reinforcement learning. (2016)

[ope19] https://github.com/openai/baselines

[ope20] https://openai.com/blog/procgen-benchmark/

[RGY+20] Raileanu, Roberta ; Goldstein, Max ; Yarats, Denis ; Kostrikov, Ilya ; Fergus, Rob: Automatic Data Augmentation for Generalization in Deep Reinforcement Learning. In: *arXiv preprint arXiv:2006.12862* (2020)

[RT19] Risi, Sebastian ; Togelius, Julian: Procedural Content Generation: From Automatically Generating Game Levels to Increasing Generality in Machine Learning. In: *arXiv preprint arXiv:1911.13071* (2019)

[RT20] Risi, Sebastian ; Togelius, Julian: *Increasing Generality in Machine Learning through Procedural Content Generation*. 2020

[RW19] Reith, Fabian ; Wandell, Brian: A convolutional neural network reaches optimal sensitivity for detecting some, but not all, patterns. In: *arXiv* (2019), S. arXiv–1911

[SB11] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011

[SLA+15] Schulman, John ; Levine, Sergey ; Abbeel, Pieter ; Jordan, Michael ; Moritz, Philipp: Trust region policy optimization. In: *International conference on machine learning*, 2015, S. 1889–1897

[SWD+17] Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal policy optimization algorithms. In: *arXiv preprint arXiv:1707.06347* (2017)

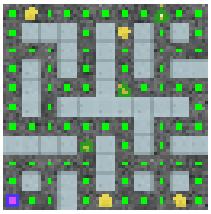
[ZVMB18] Zhang, Chiyuan ; Vinyals, Oriol ; Munos, Remi ; Bengio, Samy: A study on overfitting in deep reinforcement learning. In: *arXiv preprint arXiv:1804.06893* (2018)

[ZWP18] Zhang, Amy ; Wu, Yuxin ; Pineau, Joelle: Natural environment benchmarks for reinforcement learning. In: *arXiv preprint arXiv:1811.06032* (2018)

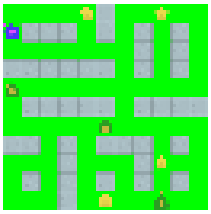
Anhang

Environment - Agenteninput

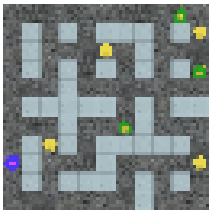
Folgend werden die Environment-Konfigurationen gezeigt, die über den Verlauf der Experimente verwendet wurden.



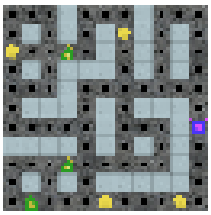
Default Spiel



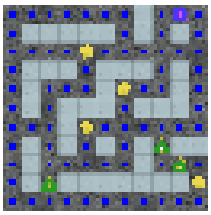
Hintergrund grün



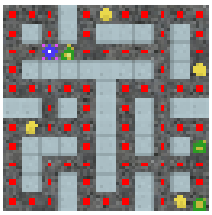
Keine vis. Orbinformation



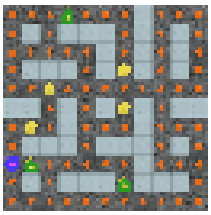
Schwarze Orbs



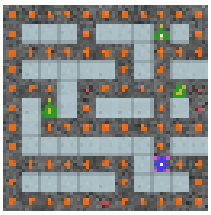
Blaue Orbs



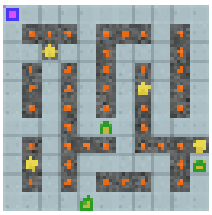
Rote Orbs



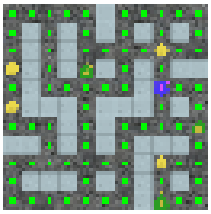
Default Spiel mit Orb-Sprite (Flammen)



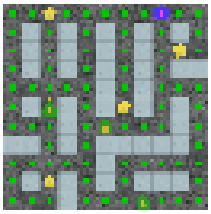
Sprite-Tausch: großer Orb u. Geist



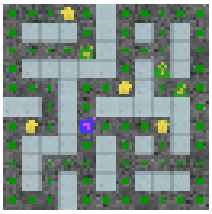
Sprite-Tausch: kleiner Orb u. Wand



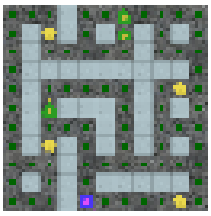
Orb-Farbe: grün:250



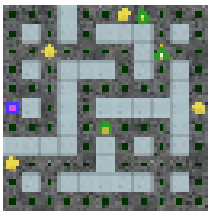
Orb-Farbe: grün:200



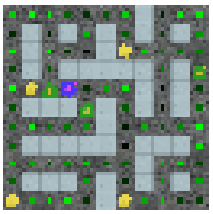
Orb-Farbe: grün:150



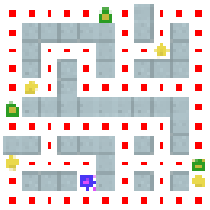
Orb-Farbe: grün:100



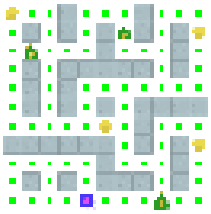
Orb-Farbe: grün:50



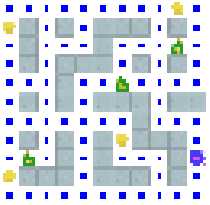
Orb-Farbe: gruen, zufaellig 0-255



BG: weiß;
Orb: r:255, g:0, b:0

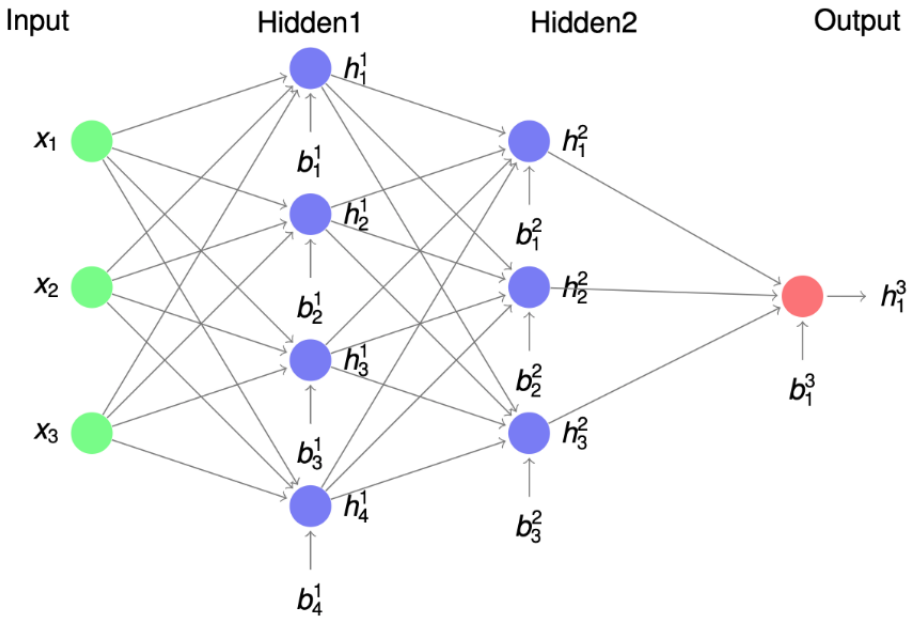


BG: weiß;
Orb: r:0, g:255, b:0



BG: weiß;
Orb: r:0, g:0, b:255

Beispiel Topologie eines dreischichtigen MLP Folgend ist ein Beispiel für den topologischen Aufbau eines Feed Forward MLP gegeben.



Darstellung eines MLP mit drei Schichten [Mau]

Hyperparameter Folgend sind die verwendeten Hyperparameter in tabellarischer Form aufgearbeitet.

Hyperparameter	Setting
Learning Rate	0,00025
Entropie Koeffizient	0,01
Value Funciton Koeff.	0,5
gamma (γ)	0,999
lambda (λ)	0,95
Anzahl Zeitschritte T	256
Anzahl Minibatches	8
Anzahl Epochen	3
Clipping Range	0,2

Tabelle 16: Übersicht über die verwendeten Hyperparameter für PPO.

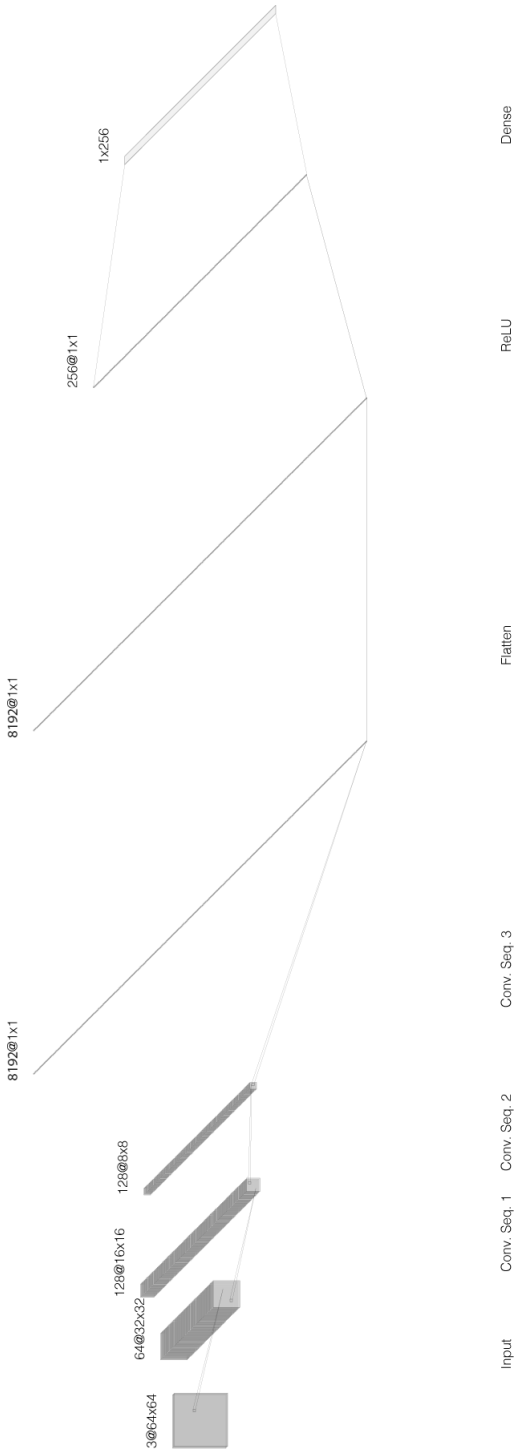
Hardware Folgend ist die verwendete Hardware in tabellarischer Form aufgelistet. Die Experimente werden auf dem Servern der Hochschule der Medien ausgeführt. Genauer wurde der GLaDOS-Server des Institute for Applied Artificial Intelligence verwendet.

	GLaDOS
Host	glados.mi.hdm-stuttgart.de
CPU	i7-6950X (10-core) @ 3.0GHz
GPU	4x NVIDIA TITAN Xp 12GB
Memory	128GB (8x 16GB) DDR4 PC2400

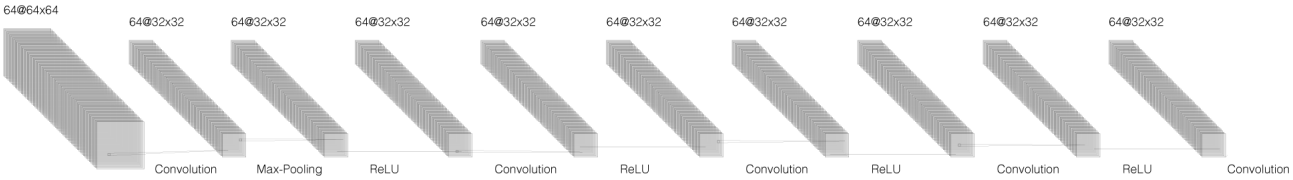
Tabelle 17: Übersicht über die verwendete Hardware.

IMPALA-Architektur

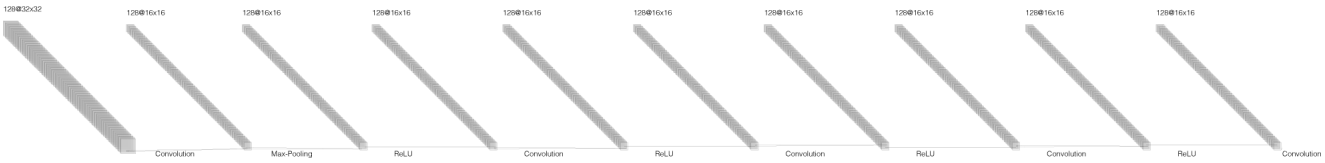
Folgend ist die IMPALA-Architektur aus dem Paper [ESM+18] dargestellt. Der Output dieser Architektur liefert den Input für sowohl die Policy-Approximation, als auch die Value-Function-Approximation.



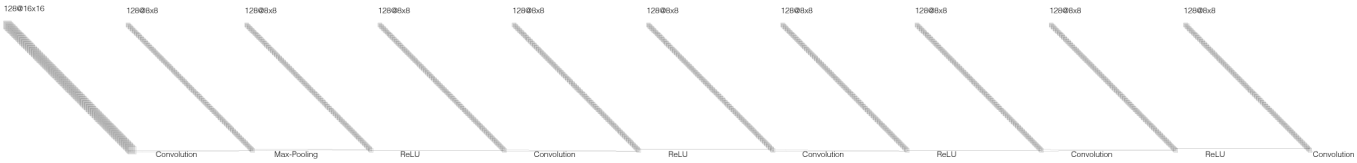
Vereinfacht dargestellte IMPALA-Architektur



Erste Convolutional Sequenz



Zweite Convolutional Sequenz



Dritte Convolutional Sequenz

Experiment - 5 Mio. Zeitschritte, Grüner und Normaler Hintergrund

Folgend sind zwei Experimente mit zwei unterschiedlichen Agenten beschrieben. Hier werden beide Agenten für 5 Mio. Zeitschritte in einem einzigen Level trainiert. Das Training der beiden Agenten unterscheidet sich lediglich im verwendeten Hintergrund. Die genauen Trainingsbedingungen können Tabelle 18 entnommen werden.

Abbildung 30 zeigt das Training mit normalem Hintergrund. Abbildung 31 zeigt das Training mit dem grünen Hintergrund.

Normaler Hintergrund
0 bis 15 Level

Abbildung 30: Training mit normalem Hintergrund für 5 Mio. Zeitschritte.

Grüner Hintergrund
0 bis 15 Level

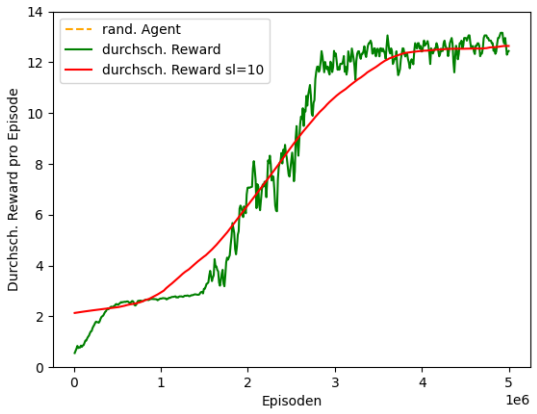


Abbildung 31: Training mit grünem Hintergrund für 5 Mio. Zeitschritte.

	Zeitschritte	Anzahl Level	Hintergrund	Farbe Orb
Training normaler Hintergrund	5 Mio	1	normal	r:0, g:255, b:0
Training grüner Hintergrund	5 Mio	1	r:0, g:255, g:0	r:0, g:255, b:0

Tabelle 18: Übersicht über geltende Rahmenbedingungen in Training und Evaluation - 15.

Erfolgsrate Experiment zum Speichervermögen Die folgende Tabelle zeigt die Erfolgsrate des jeweils letzten Checkpoints aller Agenten des Experiments 5.3.3.

	Erfolgsrate %, norm. BG	Erfolgsrate %, grüner BG
Checkpoint 1	98	78
Checkpoint 2	90	20
Checkpoint 3	76	8
Checkpoint 4	56	4
Checkpoint 5	54	0
Checkpoint 6	64	0
Checkpoint 7	52	2
Checkpoint 8	24	0
Checkpoint 9	4	4
Checkpoint 10	4	4
Checkpoint 11	34	0
Checkpoint 12	8	0
Checkpoint 13	4	0
Checkpoint 14	10	2
Checkpoint 15	8	0

Tabelle 19: Übersicht über die Erfolgsrate der Agenten mit normalem und grünem Hintergrund.

Konzept des Convolutional Filtering Die folgende Abbildung zeigt einen Schritt des Convolutional Filtering anhand eines zweidimensionalen Inputs. Diese Abbildung dient einer bildlichen Veranschaulichung des in Abschnitt 3.3 vorgestellten Konzepts von Convolutional-Layers.

Convolutional-Layer für 2D-Input Die folgende Abbildung zeigt eine Berechnung eines Features aus einem zweidimensionalen Convolutional-Layer. Weiter zeigt das Bild die jeweiligen geteilten Gewichte (hier in rot und blau). Die Kernel Size ist hier $[3 \times 3]$. Somit teilen sich die Neuronen $x_{11}, x_{12}, ..., x_{33}$ eine Gewichtsmatrix. Dasselbe gilt für jeden Block des Inputs der Größe $[3 \times 3]$, wie in Abbildung 33 ersichtlich.

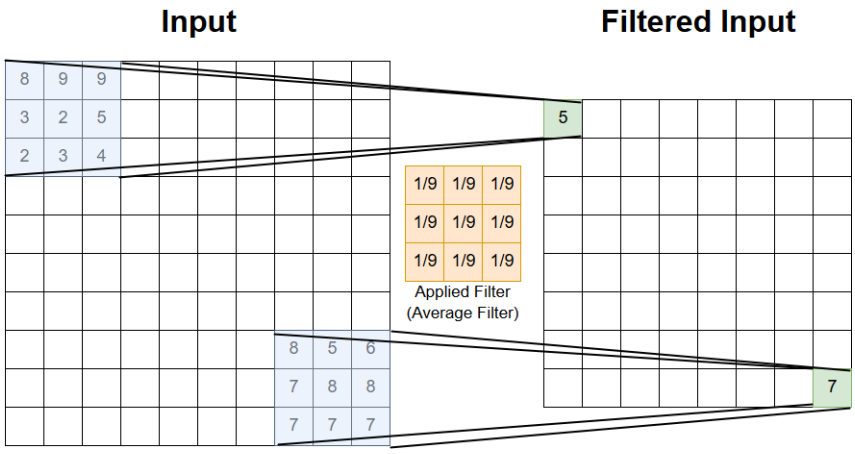


Abbildung 32: Darstellung des Konzepts des Convolutional Filtering [Mau20].

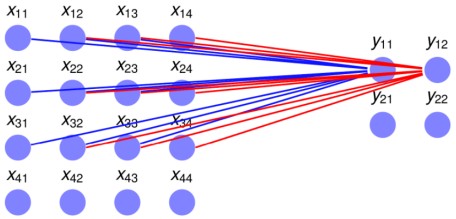


Abbildung 33: Darstellung einer Single Feature Map, extrahiert aus einem einkanäligen, zweidimensionalen Convolutional-Layer [Mau20].

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Arbeit
Deep Reinforcement Learning

Wie wirkt sich die visuelle Augmentation des Environments auf die Generalisierung eines RL-Agenten in Procgen aus?

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keine anderen Prüfungsbehörde vorgelegt bekommen.

$Unterschrift$: $Ort, Datum$: