Projektdokumentation

Praktikum Applied Reinforcement Learning

22.09.2019

Gruppe 1
Mauritius Klein
Ofek Lewinsohn
Julian Titze (hat wegen Operation nicht an Projekt teilgenommen)

Über Sonic

Sonic the Hedgehog ist ein klassisches Jump and Run Game. Erstmals wurde 1991 ein Sonic Spiel auf den Markt gebracht. Der Entwickler Sega wollte damals die Geschwindigkeit und Leistung seiner neuen 16-Bit Konsole demonstrieren.

Ziel des Spiels ist es die von Dr. Eggman gefangen gehaltenen Tiere zu befreien und so zu verhindern, dass dieser die Tiere als Roboter missbraucht um an die Weltherrschaft zu gelangen. Nur Sonic ist dem bösen Dr. Eggman gewachachen und kann ihn ein für alle Mal vernichten.

Mit Sonic Mania wurde 2017 das aktuell letzte Sonic Spiel herausgebracht. War Sonic zunächst nur ein Konsolenspiel, so wurden Dank seines großen Erfolges Sonic Versionen für nahezu alle Plattformen entwickelt und der Charakter Sonic auch in vielen anderen Spielen, wie zum Beispiel Super Smash Bros. Brawl oder Little Big Planet integriert.

Aufgabenstellung-Abstract

Ziel des Projekts ist es einen Reinforcement Learning Algorithmus zu implementieren, welcher dazu in der Lage ist das Spiel Sonic the Hedgehog zu durchspielen.

Hierbei soll eine allgemeine Methodik über alle Level des Spiels erlernt werden und nicht einfach nur einzelne Level des Spiels auswendig gelernt werden.

Im Folgenden wird zunächst auf verwandte Arbeiten und Vorarbeiten wie dem OpenAI Contest oder der Architektur des Dueling Networks eingegangen, im Anschluss wird zunächst die Methodik unseres Learning Algorithmus genauer erläutert, die auf vielen vorher vorgestellten Konzepten basiert, sich aber doch grundlegend unterscheidet. Im Anschluss wird eine Evaluierung über den Lernerfolg vorgestellt, insbesondere der Rewardfunktion des Algorithmus und der Veränderung relevanter Hyperparameter wie Epsilon und die Anzahl an Trainings und Experimenten pro Trainings. Abschließend wird zunächst auf die Problematiken/Challenges die während des Projektes auftraten eingegangen und zum Schluß ein Fazit bzw. Ausblick gegeben und die Aufgabenverteilung innerhalb des Teams skizziert.

Related Work

Im folgenden Abschnitt wird auf wichtige und relevante Paper und Vorarbeiten eingegangen, die im Bezug zum Sonic Reinforcement Learning stehen.

So wird zunächst auf einen von OpenAI veranstalteten Contest zu Reinforcement Learning eingegangen, im Anschluss werden für unser Konzept wichtige Paper zum Double Q-Learning und zu Dueling Network Architecture vorgestellt.

I. OpenAI Retro Contest

Im April 2018 veranstaltete OpenAI einen Contest zur Implementierung von Retro Spielen (https://openai.com/blog/retro-contest/). Zu diesem Anlass wurde die Gym Retro Plattform (https://github.com/openai/retro) zur Verfügung gestellt, die es ermöglicht Retro Games in einer Gym Environment (https://gym.openai.com) zu verwenden. Gym ist ein Toolkit, das speziell darauf ausgerichtet ist Reinforcement Learning Algorithmen zu entwickeln und miteinander zu vergleichen. Es wurden einige Baseline Algorithmen, z.B. Rainbow DQN und PPO für den Contest zur Verfügung gestellt. In “Gotta learn fast: A new Generalization in RL” (https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/retro-contest/gotta_learn_fast_report.pdf) werden diese vorgestellt und evaluiert. Die Autoren machen außerdem den Vorschlag “training” und “test” Environments strikt zu trennen um eine bessere Generalisierung zu erhalten.

Im Rahmen des OpenAI Retro Contest gab es mehrere Gruppen die mit dem Spiel Sonic the Hedgehog an den Start gingen und deren Ansätze eine große Hilfe für das Konzept dieses Projekts waren. Die Projekte und Ideen der einzelnen Gruppen werden im Folgenden kurz vorgestellt.

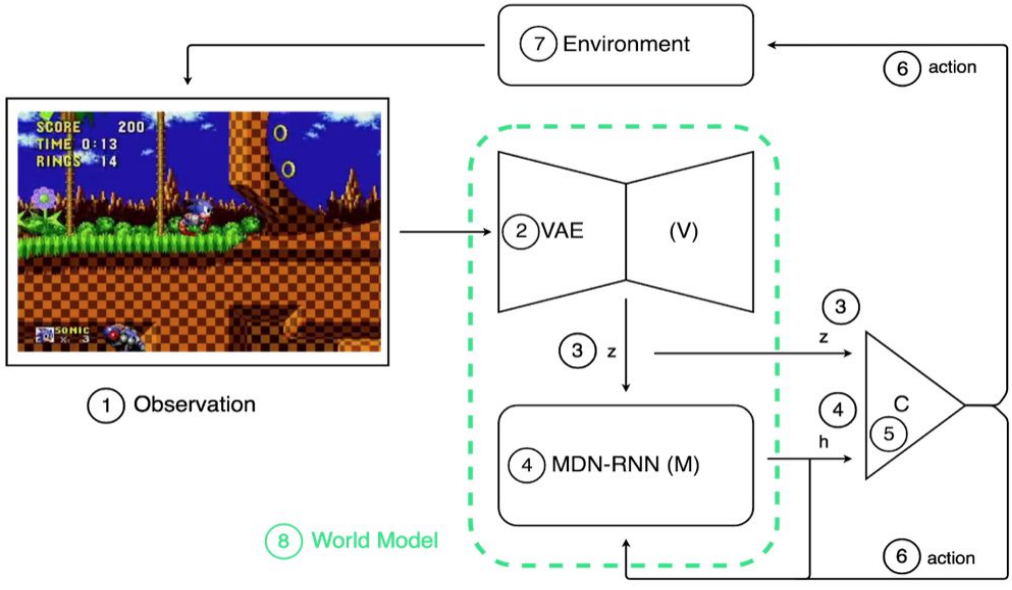
A. Aurelian Tactics:

Dieses Team beschrieb in seinem Blogeintrag (https://medium.com/aureliantactics/attempting-to-beat-sonic-the-hedgehog-with-reinforcement-learning-6ca32d4fd86e) ausführlich über die Testergebnisse der von OpenAI mitgelieferten Baseline Algorithmen. So waren diese nur in der Lage 6 der 17 Level durchzuspielen. Der Ansatz des Aurelian Teams war es PPO zu verwenden und ausschließlich dessen Rewardfunktion anzupassen. Die mitgelieferte Rewardfunktion basierte allein darauf, dass Sonic horizontalen Fortschritt macht. Idee des Aurelian Teams war es nun Sonic entlang von “human trajectories” laufen zu lassen, wobei er für Fortschritt auf dieser Trajektorie belohnt wurde. Mit diesem Ansatz schaffte das Team 8 der 11 verbleibenden Level, die noch nicht von einem der Baseline Algorithmen geschafft wurden, zu durchspielen.

B. Daniel Bourke:

Daniel Bourke veröffentlichte einige interessante und hilfreiche Artikel zum Thema Reinforcement Learning mit Sonic (z.B.: https://medium.com/@mrdbourke/the-world-model-of-a-hedgehog-6ff056a6dc7f). Zunächst versuchte sein Team den Rainbow DQN (einer der von OpenAI mitgelieferten Algorithmen) umzuschreiben. Später aber entschloß sich das Team dazu ein anderes Konzept zu verwenden. Basierend auf dem Artikel “Hallucinogenic Deep Reinforcement Learning using Kera and Python” (https://medium.com/applied-data-science/how-to-build-your-own-world-model-using-python-and-keras-64fb388ba459), entwickelten sie ihren Learning Algorithmus, indem sie ihr “World Model” wie folgt definierten.

Kernidee war, dass sie Pixeldaten einer Observation über ein Vision Model (Convolutional Variational Autoencoder) in encodeter Form an ein Memory Model (Recurrent Neural Network mit Mixture Density Network als Output), sowie an einen Controller übergeben werden, wobei der Controller zusätzlich den Output des Memory Models erhält, um basierend darauf Aktionen auszuwählen. Die gewählte Aktion wird wiederum zurück an das Memory Model gegeben, sowie an das Environment.



An overview of the World Models architecture described in the original World Models paper.

II. Deep Reinforcement Learning with Double Q-learning

Das Paper zu Double Q-Learning (https://arxiv.org/pdf/1509.06461.pdf) beschreibt eine Methode die das häufig aufretenden Problem der Überschätzung beim Deep Q-Network (einem "einfachen" Q-Learning mit Neuronalem Netz (https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4)) verringern soll. Der beschriebene Double Q-Learning Algorithmus soll neben der niedrigeren Überschätzung auch eine bessere Performance auf einigen Spielen geleistet haben, sowie stabiler und besser lernen. Als Weiterentwicklung wird das so genannte Double DQN eingeführt.

Im herkömmlichen DQN wird derselbe Wert verwendet, um eine Aktion auszuwählen und zu bewerten. Dies führt mit größerer Wahrscheinlichkeit dazu überschätzte Werte auszuwählen, was dazu führt sehr optimistische Werte zu schätzen. Die Idee des Double Q-Learnings ist es nun 2 verschiedene Wertefunktionen zu haben und jeweils eine der beiden Gewichtmengen nach dem Zufallsprinzip zu updaten. So ist für jedes Update eine Gewichtemenge für den die Greedy Policy und die andere für die Auswahl des Wertes zuständig.

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

Außerdem wird als Weiterentwicklung des Double Q-Learnings der so genannte Double DQN Algorithmus vorgestellt. Hier werden zur Auswahl der Greedy Policy nicht wie vorher die Gewichte des zweiten Netzes verwendet, sondern diese durch die Gewichte des Zielnetzes ersetzt. Auf diese Art und Weise können sowohl die Vorteile des Double Q-Learnings eingebracht werden, aber auch der erprobte DQN Algorithmus beibehalten werden.

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-)$$

Die Gewichte des Zielnetzes θ^- werden in relativ großen Abständen (jede x>1 Trainingsschritte) direkt als eine Kopie der Gewichte des Onlinenetztes θ erstellt. Dies trägt zu stabilerem Lernen bei.

III. Dueling Network Architectures for Deep Reinforcement Learning

Das Paper zu Dueling Network (https://arxiv.org/pdf/1511.06581.pdf) stellt eine Architektur für neuronale Netze vor, die spezifisch für Reinforcement Learning (RL) konstruiert ist und einige Vorteile gegenüber herkömmlichen neuronalen Netze bietet.

In der Regel wird in einer DQN Applikation ein Convolutional neuronales Netz verwendet, das von den Bildern eines Spiels, die Q-Werte der Art, Q (s=Bild, a=Aktion), für alle möglichen Aktionen schätzt. In Computerspiele ist es aber oft der Fall, dass der V-Wert, V (s=Bild), also der Wert des Zustands selbst wichtiger ist, unabhängig von der Aktion. Dies sieht man zum Beispiel daran wenn man mit Sonic nach links oder rechts geht. Sofern keine unmittelbare Gefahr durch einen Gegner besteht, ist es relativ unwichtig ob sich Sonic nach rechts oder links bewegt. Der Zustand an sich zählt (Sonic ist dort wo er ist) und nicht die Aktion. Um dies zu berücksichtigen wurde von den Autoren das Konzept des Dueling Networks entwickelt, das im folgenden kurz vorgestellt wird.

Der Q-Wert besteht aus dem V-Wert für den Zustand und der Advantage A-Wert, der die Überführung von einer Aktion in einen bestimmten Zustand beschreibt:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \qquad (1)$$

Leider lassen sich mit dieser Formel allerdings für einen bestimmten Q-Wert nicht mehrere Kombinationen aus V-Wert und A-Wert ermitteln.

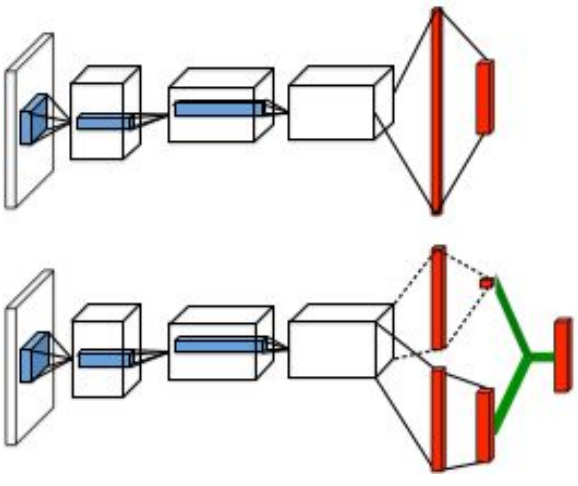
Daher wird eine verbesserte Variante für die Berechnung des Q-Werts vorgeschlagen:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right) \qquad (2)$$

Die Parameter α und β stehen für die unterschiedlichen Gewichte im neuronalen Netz.

Vorteil dieser Variante ist, dass diese stabiler trainiert, da der Advantage sich nur so schnell ändern muss, wie der Durchschnitt.

Die resultierende Architektur ist in zwei gleiche Convolutional Networks aufgeteilt, die jeweils einen V-Wert und einen Vektor mit A-Werten schätzen. Beide werden unter Verwendung von Formel 2 konkateniert, um die Q-Werte für alle möglichen Aktionen zu schätzen.



Diese Architektur erwies sich im Vergleich zu einfachen Convolutional Networks für viele RL-Applikationen als besser, insbesondere dann, wenn die Anzahl an möglichen Aktionen sehr groß war.

Learning Process-Algorithmus im Detail

Im folgenden werden zum einen die Eingabe und die nötige Vorverarbeitung für den eigentlichen Learning Algorithmus erläutert, anschließend die allgemeine Architektur im Bezug auf die vorgestellten Konzepte aus Related Work. Abschließend wird der Algorithmus selbst im Detail vorgestellt und auf dessen wichtigen Teilfunktionen eingegangen.

I. Eingabe und Vorverarbeitung

Die Eingabe des neuronalen Netzes, welches die Q-Werte schätzt, besteht im wesentlichen aus 2 Komponenten. Zum einen werden die Pixel eines Spielmomentes enbezogen, zum anderen aber auch ein so genannter Infovektor, welcher Attributwerte zum Zeitpunkt der Aufnahme speichert.

A. Bilderpixel:

Als erster Input für die Auswahl einer Aktion für Sonic, dient ein Bild. Dieses wird zunächst auf 128x128 Pixel verkleinert, danach normalisiert und im Anschluss in Graustufen gewandelt . Auf diese Art und Weise werden weniger unnötige Pixel und Informationen verwendet und trotzdem die relevanten Bestandteile des Bildes aufgenommen. Außerdem werden 4 aufeinanderfolgende Bilder verwendet, um die Bewegung im Spiel ideal zu erfassen.

B. Infovektor:

Der Infovektor dient als zweiter Input und besteht aus Attributwerten des Spiels. So enthält er Informationen wie zum Beispiel den aktuellen Score, die Anzahl an Leben von Sonic oder wie viele Ringe er bisher gesammelt hat. Insgesamt gibt es 11 dieser Attribute, die direkt vom Hauptspeicher eingelesen werden.

II. Architektur

Die verwendete Architektur ist eine Variante des schon erläuterten Dueling Netzes.

Zunächst werden 4 aufeinanderfolgende Bilder des Spiels als Eingabe einem Convolutional Neural Networks übergeben.

Der Anteil des Convolutional Networks besteht aus nur 4 Layern und ist somit relativ klein, aber ausreichend groß, denn die Eingabe-Bilder sind ebenso recht klein. Da die genaue Position und Bewegung von Sonic und von den Gegnern aus den Bildern erfasst werden müsste, werden Layer wie Max-Pooling und andere Merkmal extrahierende positionsunabhängige Layer nicht verwendet.

Die Ausgabe des Netzes wird mit dem beschriebenen Infovektor konkateniert, eine Idee von uns, die im Internet bei anderen Sonic RL-Ansätzen nach unseren Recherchen noch nicht probiert wurde.

Die resultierende Konkatenation wird einem von uns implementierten Dueling Netzes als Eingabe übergeben. Dessen Ausgabe wiederum ist die Ausgabe des Gesamtnetzes, also ein Vektor mit den geschätzten Rewards für alle 8 möglichen Aktionen die Sonic ausführen kann.

Das neuronale Netz wird an dieser Stelle deshalb verwendet, weil die Anzahl aller möglichen Kombinationen aus 4 Bildern und dem Infovektor sehr groß wäre und dessen Implementierung in Form einer riesigen Tabelle unrealisierbar wäre.

Ein weiterer positiver Effekt der Darstellung über ein neuronales Netz ist die Tatsache, dass ähnliche Kombinationen sich gegenseitig beeinflussen lassen und nicht nur unabhängig voneinander in Zeilen Tabelle existieren.

III. Algorithmus

A. Allgemein:

Für den Algorithmus selbst wurde eine Variante des beschriebenen Double-DQN Algorithmus gewählt.

So werden zufällige Level des Spiels abhängig von der Anzahl der vorgegebenen Anzahl an Experimente eines Trainings und der vorgegebenen zeitlichen Länge/ Dauer eines Experimentes gespielt. Hierbei wird ein Decaying Epsilon-Greedy Verfahren verwendet, wobei die gesammelte Erfahrung in einer Queue gespeichert wird.

Nach dem Decaying Epsilon-Greedy Verfahren hat Sonic eine kleine Epsilon-Wahrscheinlichkeit eine willkürliche Aktion zu wählen. Ansonsten wählt er die beste Aktion nach dem aktuellen Modell. Der Epsilonwert wird nach jedem Level verkleinert, um die Anzahl der willkürlichen Aktionen zu reduzieren und so die Exploration schrittweise zu reduzieren und stattdessen eine höhere Exploitation zu erreichen. Eine höhere Explotation bedeutet, dass die wahrscheinlicheren Aktionen besser geschätzt werden.

Alle 100 Schritte wird ein Batch Training der Größe 32 von diesem Queue zufällig gesamplet und darauf trainiert. Das Target des neuronalen Netzes wird dann mittels der oben beschriebenen Double-DQN Update Formel berechnet. Alle 8192 Schritte werden die Gewichte des Onlinenetzes zum Targetnetz kopiert.

Sonic kann sich für eine von 8 diskreten Aktionen entscheiden. Jede gewählte Aktion wird von Sonic 4 mal hintereinander ausgeführt, um die Anzahl an Entscheidungen pro Sekunde zu reduzieren.

Diese Maßnahme ist nötig und sinnvoll, denn das Spiel ändert sich nicht schnell genug, als das es Sinn machen würde so viele Aktionsentscheidungen in so kurzer Zeit zu treffen und somit auch unnötigerweise die Trainingsdauer zu erhöhen.

B. Epsilon-Decay Funktion:

In jedem Experiment eines Trainings wurde der Epsilon Wert durch eine Epsilon-Decay Funktion verringert, indem der jetzige Epsilon mit dem gewählten Decay multipliziert wurde.
Der Decay selbst wird am Anfang vom Training an Hand der Anzahl von Experimenten gerechnet, damit er die passende Verringerungsrate findet.

In “Reward based Epsilon Decay” (https://aakash94.github.io/Reward-Based-Epsilon-Decay/) wird vorgeschlagen, dass Epsilon an Hand des Rewards angepasst werden sollte. Da sich allerdings der Reward von Sonic-Level zu Level stark unterschied, war es nicht leicht einen Reward basierten Epsilon Decay zu implementieren und wir entschlossen uns für den klassischen Ansatz.

C. Rewardfunktion:

Kernbestandteil des Learning Verfahrens ist die Rewardfunktion. Auf die Schwierigkeiten beim Finden der idealen Rewardfunktion wird im Kapitel Challenges genauer eingegangen. Im folgenden wird die in der Projektabgabe finale Rewardfunktion kurz zusammengfasst.

Ein sehr großer Reward wurde zunächst für das Gewinnen eines Levels gegeben, da dies als Hauptziel des Algorithmus gesehen wurde.

Neben dem Durchlaufen eines Levels wurden außerdem folgende Werte in die Rewardfunktion aufgenommen und belohnt:

- Vorankommen in x-Richtung
    - von Schritt zu Schritt normalisiert über die Länge des Levels
    - global -> Übertreffen des x-Richtungs Rekordes
    - Rennen, mehr als ein x weiter in einem schritt.
- Anzahl gesammelter Ringe
- Score

Die Rewardfunktion wurde kontinuierlich angepasst und verbessert. So flossen in früheren Versionen auch besipielsweise folgende Werte mit ein die durch Belohnung oder Bestrafung Einfluss auf den Reward nahmen:

- Anzahl an Leben
- Auf der Stelle hüpfen
- Verlust von Ringen
- Nach links gehen

Evaluierung

Wie in der Aufgabenstellung beschrieben, war das Ziel eine allgemeine Methode zu finden, sodass Sonic auf verschiedenen Leveln möglichst weit läuft und nicht nur einzelne Level auswendig lernt. Kernpunkt der Evaluierung war dementsprechend, wie weit Sonic in einem Level gekommen ist bzw. welche Level unter welchen Paramtervarianten geschafft wurden.

Ein Level gilt dann als geschafft, wenn Sonic bis vor den Endgegner gelaufen ist. Besiegen musste er diesen nicht, da die Endgegner von Level zu Level verschieden sind und eine allgemeine Strategie daher nicht möglich war.

Im folgenden werden zunächst das Setup der Evaluierung erläutert und anschließend auf einige interessante Ergebnisse der verschieden Paramter eingeganegen.

I. Hyperparameter

Für die Evaluierung wurden 4 verschiedene Hyperparameter verändert, um festzustellen, welche der Parameter die ideale Kombination für schnelles und erfolgreiches Lernen sind.

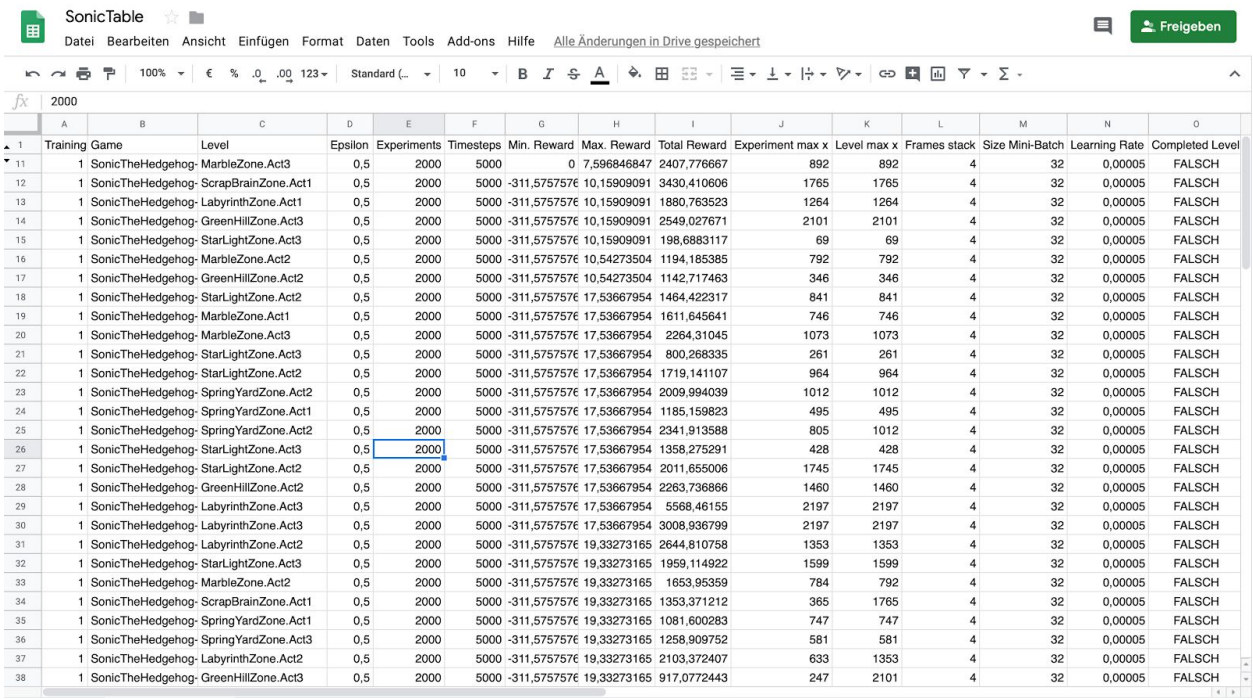
So wurden die Epsilonwerte, also die Wahrscheinlichkeit mit der eine zufällige Aktion ausgeführt wird, variiert. Es wurden hierfür die Werte 0.3, 0.5, und 0.7 betrachtet.

Außerdem wurde die Länge eines Experimentes (seine timesteps) schrittweise von 5000 über 10000 bis 15000 erhöht. Außerdem wurde die Menge an Trainings abhängig von den timesteps angepasst, um eine konstante Laufzeit zu erhalten.

Während eines Trainings wurde pro Experiment ein zufälliges Spiel aus Sonic The Hedgehog Genesis gewählt. Das erste Level des ersten Spiels wurde bewusst nicht trainiert, da dieses verwendet wurde, um das trainierte Model zu testen.

II. Teilen von Evaluierungsdaten über Google Spreadsheets

Um zu gewährleisten, dass sämtliche Evaluierungsparamter und Trainingserfolge nicht nur lokal auf den insgesamt 5 verschiedenen Computern liegen, wurde eine Funktion implementiert, die Einträge in einem Google Spreadsheet generiert. So wurden nach Ablauf eines Trainings für jedes einzelne Experiment die für die Evaluierung relevanten Daten in einer neuen Zeile im Spreadsheet gespeichert. Dies ermöglichte einen schnellen und einfachen Zugriff auf die Gesamtheit der Daten, ohne diese manuell zusammenführen zu müssen.


SonicTable

Datei Bearbeiten Ansicht Einfügen Format Daten Tools Add-ons Hilfe Alle Änderungen in Drive gespeichert

Freigeben

2000

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Training	Game	Level	Epsilon	Experiments	Timesteps	Min. Reward	Max. Reward	Total Reward	Experiment max x	Level max x	Frames stack	Size Mini-Batch	Learning Rate	Completed Level
11	1	SonicTheHedgehog-	MarbleZone.Act3	0,5	2000	5000	0	7,596846847	2407,776667	892	892	4	32	0,00005	FALSCH
12	1	SonicTheHedgehog-	ScrapBrainZone.Act1	0,5	2000	5000	-311,5757576	10,15909091	3430,410606	1765	1765	4	32	0,00005	FALSCH
13	1	SonicTheHedgehog-	LabyrinthZone.Act1	0,5	2000	5000	-311,5757576	10,15909091	1880,763523	1264	1264	4	32	0,00005	FALSCH
14	1	SonicTheHedgehog-	GreenHillZone.Act3	0,5	2000	5000	-311,5757576	10,15909091	2549,027671	2101	2101	4	32	0,00005	FALSCH
15	1	SonicTheHedgehog-	StarLightZone.Act3	0,5	2000	5000	-311,5757576	10,15909091	198,6883117	69	69	4	32	0,00005	FALSCH
16	1	SonicTheHedgehog-	MarbleZone.Act2	0,5	2000	5000	-311,5757576	10,54273504	1194,185385	792	792	4	32	0,00005	FALSCH
17	1	SonicTheHedgehog-	GreenHillZone.Act2	0,5	2000	5000	-311,5757576	10,54273504	1142,717463	346	346	4	32	0,00005	FALSCH
18	1	SonicTheHedgehog-	StarLightZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	1464,422317	841	841	4	32	0,00005	FALSCH
19	1	SonicTheHedgehog-	MarbleZone.Act1	0,5	2000	5000	-311,5757576	17,53667954	1611,645641	746	746	4	32	0,00005	FALSCH
20	1	SonicTheHedgehog-	MarbleZone.Act3	0,5	2000	5000	-311,5757576	17,53667954	2264,31045	1073	1073	4	32	0,00005	FALSCH
21	1	SonicTheHedgehog-	StarLightZone.Act3	0,5	2000	5000	-311,5757576	17,53667954	800,268335	261	261	4	32	0,00005	FALSCH
22	1	SonicTheHedgehog-	StarLightZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	1719,141107	964	964	4	32	0,00005	FALSCH
23	1	SonicTheHedgehog-	SpringYardZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	2009,994039	1012	1012	4	32	0,00005	FALSCH
24	1	SonicTheHedgehog-	SpringYardZone.Act1	0,5	2000	5000	-311,5757576	17,53667954	1185,159823	495	495	4	32	0,00005	FALSCH
25	1	SonicTheHedgehog-	SpringYardZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	2341,913588	805	1012	4	32	0,00005	FALSCH
26	1	SonicTheHedgehog-	StarLightZone.Act3	0,5	2000	5000	-311,5757576	17,53667954	1358,275291	428	428	4	32	0,00005	FALSCH
27	1	SonicTheHedgehog-	StarLightZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	2011,655006	1745	1745	4	32	0,00005	FALSCH
28	1	SonicTheHedgehog-	GreenHillZone.Act2	0,5	2000	5000	-311,5757576	17,53667954	2263,736866	1460	1460	4	32	0,00005	FALSCH
29	1	SonicTheHedgehog-	LabyrinthZone.Act3	0,5	2000	5000	-311,5757576	17,53667954	5568,46155	2197	2197	4	32	0,00005	FALSCH
30	1	SonicTheHedgehog-	LabyrinthZone.Act3	0,5	2000	5000	-311,5757576	17,53667954	3008,936799	2197	2197	4	32	0,00005	FALSCH
31	1	SonicTheHedgehog-	LabyrinthZone.Act2	0,5	2000	5000	-311,5757576	19,33273165	2644,810758	1353	1353	4	32	0,00005	FALSCH
32	1	SonicTheHedgehog-	StarLightZone.Act3	0,5	2000	5000	-311,5757576	19,33273165	1959,114922	1599	1599	4	32	0,00005	FALSCH
33	1	SonicTheHedgehog-	MarbleZone.Act2	0,5	2000	5000	-311,5757576	19,33273165	1653,95359	784	792	4	32	0,00005	FALSCH
34	1	SonicTheHedgehog-	ScrapBrainZone.Act1	0,5	2000	5000	-311,5757576	19,33273165	1353,371212	365	1765	4	32	0,00005	FALSCH
35	1	SonicTheHedgehog-	SpringYardZone.Act1	0,5	2000	5000	-311,5757576	19,33273165	1081,600283	747	747	4	32	0,00005	FALSCH
36	1	SonicTheHedgehog-	SpringYardZone.Act3	0,5	2000	5000	-311,5757576	19,33273165	1258,909752	581	581	4	32	0,00005	FALSCH
37	1	SonicTheHedgehog-	LabyrinthZone.Act2	0,5	2000	5000	-311,5757576	19,33273165	2103,372407	633	1353	4	32	0,00005	FALSCH
38	1	SonicTheHedgehog-	GreenHillZone.Act3	0,5	2000	5000	-311,5757576	19,33273165	917,0772443	247	2101	4	32	0,00005	FALSCH

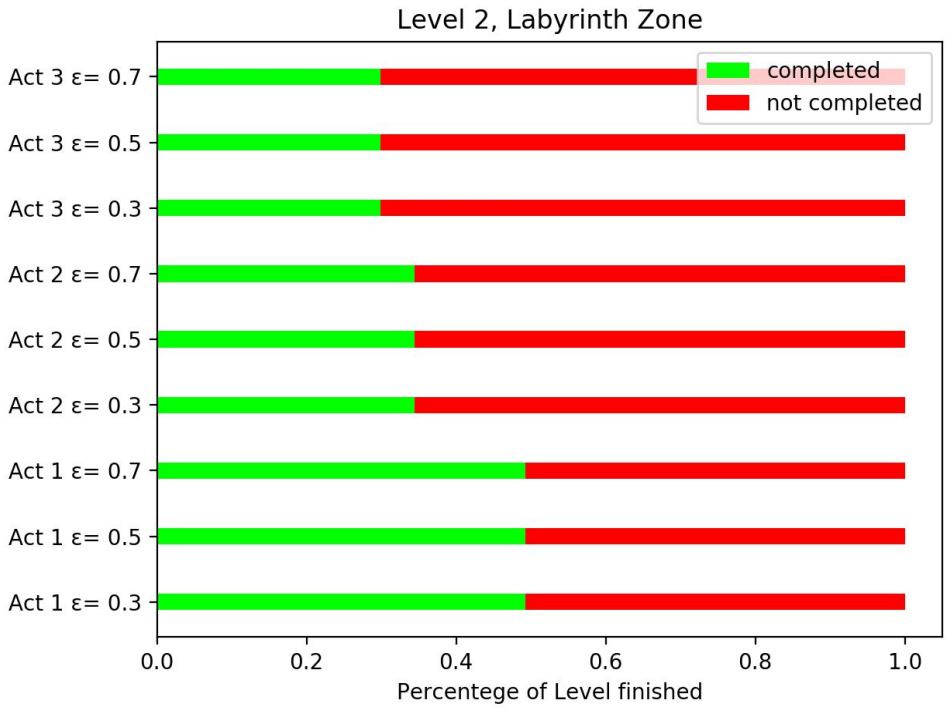
III. Epsilonevaluierung

Ein wesentlicher Teil der Evaluierung war das herausfinden eines geeigneten Epsilonwertes mit dem ein Training begonnen wird. Es wurden mit 0.3, 0.5, 0.7 bewusst sehr verschiedene Werte verwendet, um deren Einfluss auf die Lernfähigkeit am besten zeigen zu können.

Konkret wurden die 3 Werte insofern verglichen, als dass für jedes Level analysiert wurde, unter welchem Epsilonwert Sonic am weitesten im jeweiligen Level gelaufen ist. Hierbei ließen sich einige Muster erkennen.

A. Epsilon unabhängige Level

Bei 7 der 16 Level kam Sonic unabhängig vom Epsilonwert jeweils nur bis zu einem gewissen Punkt, an dem er nicht weiterkam. So waren hier die Level Labyrinth-Zone (Act 1, Act 2, Act 3), Marble-Zone(Act 1, Act 2, Act 3), sowie Spring Yard-Zone (Act 3) betroffen. Es handelt sich bei allen, um sehr schwierige Passagen des Spiels, für deren Überwinden ein wesentlich ausführlicheres Training nötig gewesen wäre oder aber das jeweilige Level einzeln trainiert werden hätte müssen. Diese Form des Level auswendig lernen wollten wir allerdings nicht. Nachteil dieser Methodik sind die schwierigen Passagen der Level, in der eine Level- spezifische Taktik nötig wäre und eine allgemeine Strategie nicht in der Lage ist, weiterzukommen.



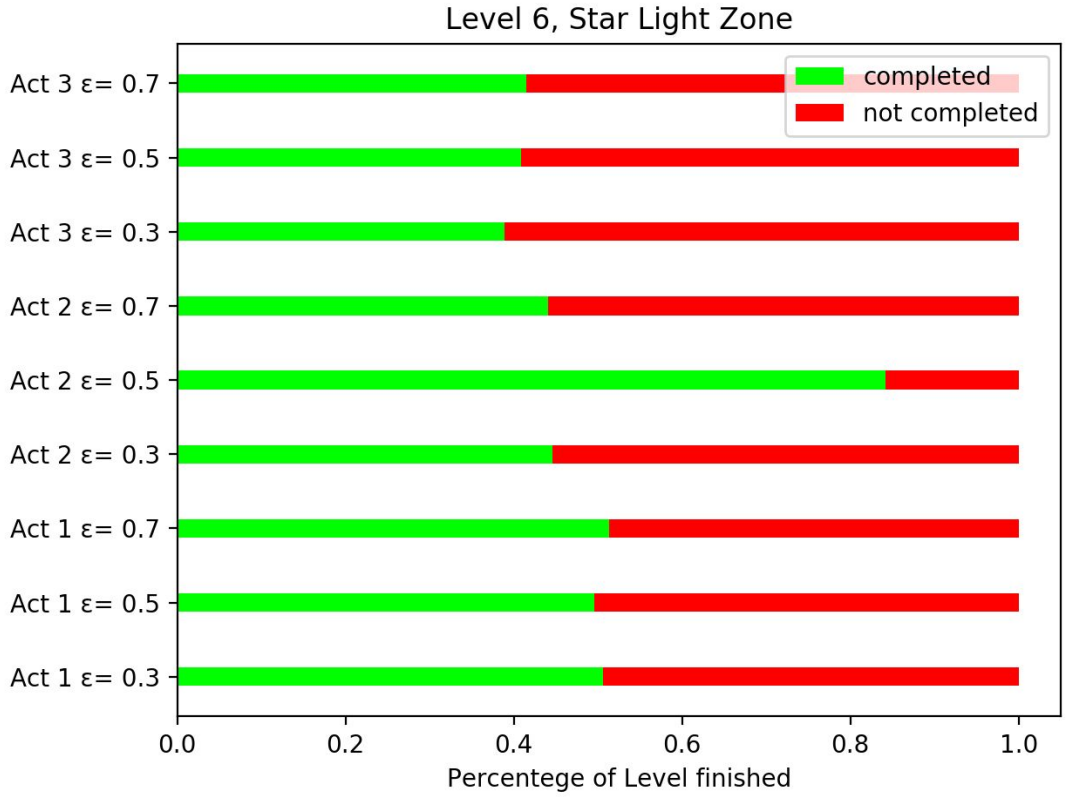
Im obenstehenden Diagramm sind beispielsweise die Fortschritte die Sonic in den Leveln Labyrinth-Zone Act 1,2,3 gemacht hat, zu sehen. Es lässt sich ablesen, dass unabhängig von Epsilon Sonic an gewissen Punkten im Leveln zu scheitern scheint.

Hier zum Beispiel ein Level aus Marble-Zone. Die Wand lässt sich nicht überwinden. Eine Taktik aus Zurücklaufen und woanders nach oben Springen wäre nötig, um an dieser Stelle weiterzukommen.

B. Epsilon Abhängigkeit

Für die übrigen 9 Level konnte keine fixe maximale x-Richtungsreichweite von Sonic festgestellt werden, sondern sie unterschied sich für verschiedene Epsilonwerte. Zwar gab es auch hier einige Level, bei denen sich ein Punkt abzeichnete, es aber Ausreißer für verschiedene Epsilon-Werte gab.

Ein gutes Beispiel hierfür ist im nachfolgenden Diagramm für das Level Star Light Zone, Act 2 zu sehen. Wurden mit Epsilon 0.3 und 0.7 jeweils nur etwa 50% des Levels durchspielt, so schaffte konnte Epsilon 0.5 fast 90% des Levels absolvieren, war also schon sehr nah am Ziel.



Eine klare Tendenz, welches Epsilon am besten geeignet war, ließ sich allerdings trotzdem nicht final überprüfen. So war es für die 9 verbliebenen Level 4 Mal Epsilon 0.5, 3 Mal 0.3 und 2 Mal 0.7, welches das Rennen machte.

Wir gehen davon aus, dass unsere Trainingszeit im Endeffekt nicht lang genug war, um klare Unterschiede in der Explorationswahrscheinlichkeit feststellen zu können.

IV. Vernachlässigte Evaluierungshyperparameter

Aus zeitlichen Gründen konnten 2 weitere interessante Hyaperparameter in der Evaluierung nicht berücksichtigt werden. Diese werden im Folgenden zur Vollständigkeit kurz erläutert.

A. Mini-Batch Größe

Da es keine allgemeine Regel zur Wahl der Mini Batch Größe gibt, entschieden wir uns die Größe von 32 zu wählen. In “Efficient Mini-batch Training for Stochatic Optimization” (https://www.cs.cmu.edu/~muli/file/minibatch_sgd.pdf) wird gesagt das größere Mini-batches schneller lernen, aber kleinere besser generalisieren können. Diesen Effekt wollten wir auch an unserem Projekt testen, aber aus zeitlichen Gründen konnte dieser Teil der Evaluierung nicht fertiggestellt werden.

B. Größe des Frame Stacks

Die Größe des Frame Stacks, also der Anzahl an Bildern die dem Neuronalem Netz als Input übergeben werden, wurde von uns auf 4 festgesetzt. Interessant wäre es gewesen zu analysieren, welchen Einfluss die Variation dieser Zahl auf den Lernerfolg von Sonic gehabt hätte. So war der ursprüngliche Plan es auch mit 6 an Stelle von 4 Bildern zu probieren. Zeitliche Gründe zwangen uns dazu diese Analyse zu vertagen.

V. Testset

Es wurde mit allen Levels außer dem ersten trainiert, damit dieses als Testset benutzt werden konnte.

Um Training und Test klar zu trennen, ließen wir nach dem Training das erste Level mit den Gewichten des neuronalen Netzes spielen. Dafür hatten wir ein kleines Skript namens run_level.py implementiert.

Beim ersten Level hat Sonic mit Epsilon 0,5 über 4000 Schritte nach vorne erzielt. Dies zeigt, dass unseres Modell gut verallgemeinern kann und die Trainingsset nicht auswendig lernte.

Ein Video vom Gameplay des ersten Level ist auf der Google Drive vorhanden.

Challenges

I. Ideale Rewardfunktion

Das Finden einer idealen Rewardfunktion stellte sich aus verschiedenen Gründen als äußerst problematisch dar.

Das erste Problem war die Diversität der Level. So gab es Level mit vielen Gegnern, Level in denen man nicht einfach nur nach rechts laufen musste, sondern im Zickzack (Labyrinth-Level). Einige Level hatten viele Schluchten, etc.. So wurde zum Beispiel im Labyrinth Level zunächst viel Reward gegeben, da das ständige von links nach rechts laufen einen hohen Reward für das vorankommen in x-Richtung brachte. Dies wirkte sich natürlich auf die anderen Level aus, in denen eine andere Strategie evtl. besser gewesen wäre. Wir haben auch versucht nur für die Labyrinth-Levels Sonic anders zu belohnen, was aber nicht viel brachte und als Schummeln betrachtet werden könnte.

Ursprüngliche Ansätze, in denen schlechte Aktionen wie Sterben oder Ringe durch Gegner verlieren hoch bestraft wurden, stellten sich als problematisch dar, da Sonic aus “Angst” vor Bestrafung nicht vorwärts kam.

Den größten Einfluss hatte das ständige Anpassen der Rewardfunktion allerdings auf die insgesamten Trainingserfolge, da es sehr lange dauerte eine neue Rewardfunktion zu testen. So wurden bis zuletzt immer wieder Anpassungen gemacht und das eigentliche Training und die Evaluierung mussten warten, bzw. wurden immer wieder von vorne begonnen.

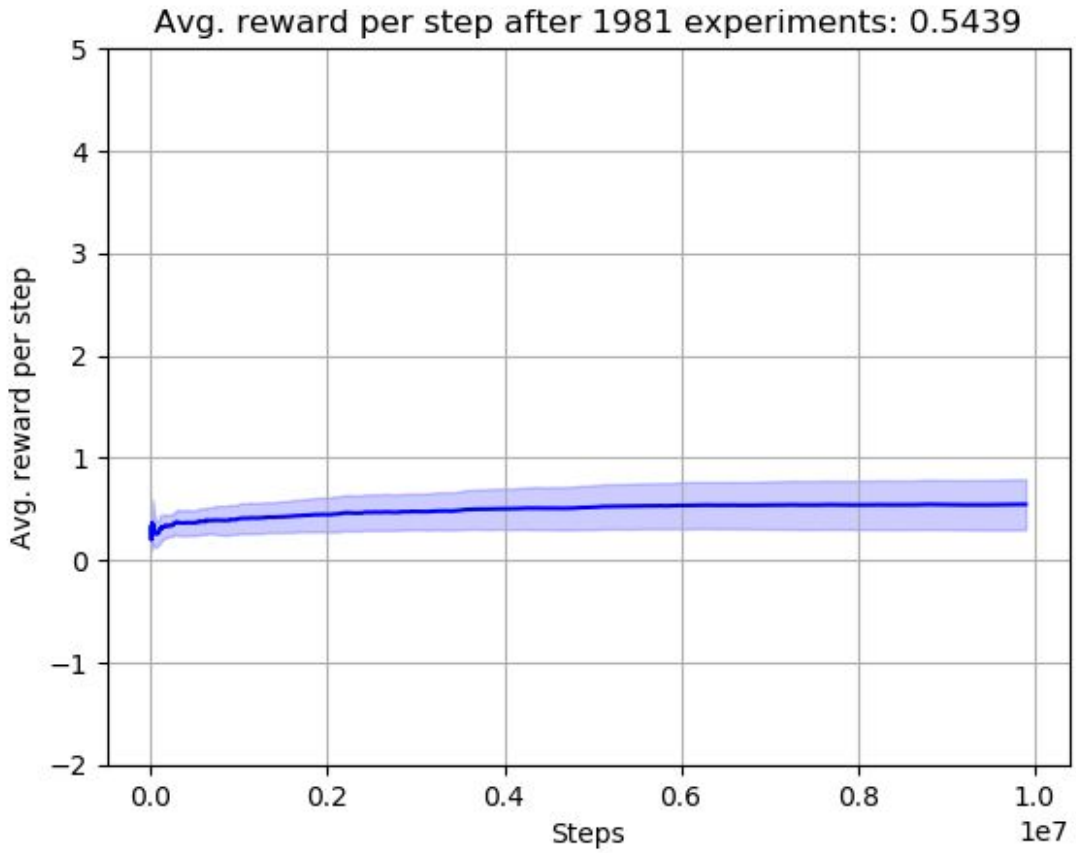
II. Average Reward per Step

Für jedes Training wird alle 20 Experimente ein Graph erstellt, der den Average Reward mit entsprechendem Konfidenzintervall darstellt.

Hierfür wurden der durchschnittliche Reward und die Standardabweichung nach jedem Schritt neu berechnet. Mit dem Ziel nicht alle in die Millionen gehende Anzahl an Schritten im Hauptspeicher halten zu müssen, wurde hier die Additivität von Durchschnitt und Varianz eingesetzt.

Generell ließ sich in allen Trainings feststellen, dass der durchschnittliche Reward mit steigender Zahl an Experimenten zwar anstieg (siehe z.B.: Diagramm unten (Epsilon = 0.5, Experimente =2000)), jedoch sehr langsam und relativ konstant. Dies und die Tatsache, dass es für sämtliche Trainings ähnlich flach verlief, lässt darauf schließen, dass entweder die Rewardfunktion noch nicht ideal ermittelt wurde oder aber die Anzahl an Schritten innerhalb eines Experimentes bzw. die Anzahl an

Experimenten innerhalb eines Trainings noch zu gering war um klare Trainingsfortschritte zu erzielen und ein längeres Training nötig gewesen wäre.



1e7 * 1.0 = 10 Millionen Schritte

III. Rechenkapazität und Zeit

Vor ein großes Problem stellte uns die uns zur Verfügung stehende begrenzte Rechenkapazität.

Wir versuchten schon beim Entwurf unseres Algorithmus darauf Rücksicht zu nehmen und wählten beispielsweise den Double DQN anstatt des PPO Algorithmus, der dazu in der Lage sein sollte aus weniger Training mehr Trainingserfolg zu generieren. Auch bei der Evaluierung schränkte uns die lange Rechenzeit sehr ein, da wir nicht lange genug trainieren konnten um alle Hyperparameter zu untersuchen, die wir für interessant gehalten hätten.

In der finalen Phase hatten wir 5 Laptops auf denen wir parallel lernten, doch auch diese reichten nicht wirklich aus, um die gewünschten Resultate zu erhalten.

Wir haben versucht das Problem durch das Erstellen eines Dockers zu lösen den wir mittels Slurm Workload Manager auf den CIP-Pool Rechnern laufen lassen wollten. Leider funktionierte das nicht.

Aufgabenverteilung

Im folgenden Abschnitt wird auf die Aufgabenverteilung innerhalb des Projektes eingegangen. Grundlegend lässt sich sagen, dass die Arbeit im Team stets harmonisch war und die Aufgabenverteilung sich erst im Laufe des Projektes basierend auf den jeweiligen Vorkenntnissen und Stärken herauskristallisierte.

I. Ofek Lewinsohn

Ofek fing relativ früh an sich Gedanken über die Architektur des Algorithmus Gedanken zu machen und schaffte einen ersten Grundstock an implementierten Code auf dem wir im Laufe des Projekts aufbauten.

II. Mauritius Klein

Sowie Ofek mit der Implementierung anfing, so startete Mauritius früher damit sich Gedanken zur Dokumentation zu machen und suchte die entsprechenden Paper raus, um aus der großen Menge an Input, die im Laufe des Projektes entstand, die wesentlich wichtigen Konzepte herauszufiltern und zu dokumentieren.

Auch wenn der eine etwas früher hier und der andere etwas früher dort angefangen hat zu arbeiten, so waren beide Teammitglieder trotzdem zu jedem Zeitpunkt des Projekts in allen Projektteilen involviert. Es herrschte ein reger Austausch, ob über Implementierungsfragen, oder aber über das Festlegen der Evaluierungskriterien etc.. Sämtliche Entscheidungen wurden vorher abgesprochen und diskutiert. Beide haben entsprechend ihrer Fähigkeiten zu jedem Zeitpunkt sehr viel Zeit in das Projekt investiert, sodass sich nicht sagen lässt, dass dieser oder jener Teil des Projekts mehr von dem Einen oder mehr dem Anderen stammt.

III. Julian Titze

Julian war im Rahmen der Abgaben während des Semesters sehr aktiv. Allerdings nahm er nur rege, bzw. gar nicht an der Projektarbeit teil. Auf die doch sehr intensiven Whatsappgruppenverläufe der anderen Teammitglieder reagierte er nicht, bzw. nur selten. Zum Kickoff Meeting kam er wegen eines Fahrradunfalls nicht, in den folgenden Wochen bis zur Abgabe waren die einzigen Nachrichten eine Bitte zum erneuten Hinzufügen zum Github Projekt, sowie ein zweimaliges Versichern jetzt mit der Arbeit anfangen zu wollen. Eine Woche vor Abgabe meldete sich Julian zuletzt, um sich zu entschuldigen und erzählte, dass er wegen des Unfalls operiert wurde. Wir wussten davon bis dahin nichts und zeigten Verständnis, allerdings einigten wir uns trotzdem einstimmig darauf, dass es keinen Sinn macht Julian in der letzten Projektwoche noch zu integrieren.

Fazit und Ausblick

Sonic the Hedgehog war ein spannendes Spiel und eine schöne Möglichkeit die theoretischen Konzepte des Reinforcement Learnings praktisch anzuwenden.

Auch wenn die Ergebnisse nicht ganz unseren Wünschen entsprechen, sind wir von unserem Ansatz sehr überzeugt und sehr enttäuscht, dass es zeitlich nicht mehr geklappt hat, soweit zu trainieren, als dass die Vorteile unseres Ansatzes zum Vorschein gekommen sind.

Ein Vergleich mit den anderen vorgestellten Konzepten der Teams aus dem OpenAI Contest wäre eine sinnvolle und spannende Fortführung unseres Projektes.

Wir sind trotz der überschaubaren Trainingserfolge, aber auch gerade wegen der vielen Zeit, die wir in das Projekt gesteckt haben stolz auf unsere Arbeit, denn der Ansatz ist in dieser Form neu und wie das bei neuen Ansätzen eben ist, kann man auch mal “auf die Schnauze fliegen”.

Außerdem hat uns das Projekt und die dazugehörende Recherche bzw. das Lesen von Papern etc. in unserem konkreten Wissen zum Reinforcement Learning auf ein anderes Level gehoben. Zwar haben wir in den Vorlesungen unter dem Semester schon viel gelernt und die Konzepte auch praktisch in den Abgaben umgesetzt, aber ein so großes Projekt von null zu starten war eine Herausforderung aus der wir beide sehr viel mitgenommen haben.

Vielen Dank für das Anbieten des Praktikums, wir werden es auf jeden Fall weiterempfehlen!