Deep-Q-Learning mit 'Super Mario Bros' und A3C.

Jan Bernd Gaida
jan.gaida@hof-university.de
Hochschule für Angewandte Wissenschaften Hof
Hof an der Saale, Bavaria, Germany

ZUSAMMENFASSUNG

In dieser Studienarbeit wird die Performance unterschiedlicher strukturierter Reinforcement-Learning (kurz: RL) Modelle innerhalb einer *Super Mario Bros.* (1985) [7] (kurz: SMB) Umgebung hinsichtlich ihrer Lernperformance und -stabilität auf Basis eines Experiments untersucht.

Konkret werden unterschiedliche RNN-Architekturen mit unterschiedlichen *CNN*-Architekturen hinsichtlich ihres Performance bei limitierter Hardware verglichen. Dazu musste sich zunächst auf einen RL-Algorithmen geeinigt werden und anschließend das Neuronale Netzwerk und dessen Hyperparameter optimiert werden.

Abschließend wird das Ergebnisse des Projekts diskutiert.

Der Quellcode für das Projekt und sowie dessen Ergebnisse stehen öffentlich als Repository zur Verfügung [10].

1 EINFÜHRUNG

Maschinelles Lernen ist die Wissenschaft Computer indirekt zu programmieren. In der vergangen Dekade ist der Einsatz solcher Programme so allgegenwärtig geworden, das Vielen ihre eigentliche Verwendung nicht bewusst ist obwohl Sie mehrmals täglich verwendet wird. [18]

Reinforcement-Learning ist ein junger Teilbereich des Maschinelle Lernens, bei dem das Programm eine Umgebung eigenständig erkundet und in Abhängigkeit des Problems eine konkrete Lösung definiert. Dieses Verständnis erlangt das Programm mittels '*Reinforcement*' (deutsch: bekräftigende, bestätigende) Mechanismen.

Medienwirksame Aufmerksamkeit erlangt dieser Teilbereich durch beeindruckende Rekorde oder Siege in sehr komplexen Spielen wie beispielsweise '*AlphaGo*', das erste Computerprogramm das einen menschlichen Spieler im dadurch berühmten Brettspiel **Go** besiegte [8]. Oder beispielsweise die künstliche Intelligenz '*OpenAI Five*' die das Multiplayerspiel *Dota 2* auf Experten Niveau spielt [22].

Auch wenn dieser Bereich scheinbar keine konkreten Verbesserungen für das alltägliche Leben birgt, bietet das experimentieren mit solchen komplexen künstlich geschaffenen Umgebungen ein großer Forschungspotential. Dies wird langfristig dazu führen, dass vor allem Roboter in weniger komplizierten Anwendungsbereichen, wie beispielsweise dem korrekten Sortieren von Objekten von Laufbändern, großer Fortschritt machen wird [9].

Das Potential von Reinforcement-Learning ist also groß und wird uns sicherlich auch weiterhin dazu Anregen, dieses weiter zu entwickeln und im Alltag nutzbar zu machen.

2 HARDWARE

Das Training des Modells fand auf einem *Consumer-Pc* statt. Konkret standen hierfür 6 GB VRAM (*GeForce GTX 1060*) erreichbar mittels *CUDA*-Treiber, sowie 16 GB RAM (DDR3 im Dual-Channel Modus bei ca. 800 MHz Frequenz) unterstützt durch eine 4 Kern CPU getaktet auf ca. 3.8 GHz (*Intel Core i5 4690K*). Genannte Hardware wurde zum Trainigs- und Testzeitpunkt moderat übertaktet.

3 LERN- UND TESTUMGEBUNG

Als Laufzeit- und Entwicklungsumgebung wurde Python 3.8 (offizielle Releaseversion) gewählt. Nachfolgend werden die wesentlichen benutzten Frameworks kurz aufgeführt, welche allesamt mittels Pip-Paketmanager installierbar sind.

3.1 OpenAI Gym und SMB-Gym

OpenAI Gym [6] ist ein bekanntest Framework zum trainieren und testen unterschiedlicher RL-Algorithmen. Ziel der Entwickler ist es hierbei für diesen Algorithmentypus eine standardisierte Umgebung, meist als Enviorment bezeichnet, zur bessere Vergleichbarkeit mit ähnlichem anzubieten. Dadurch eignet sich dieses Framework bestens für sog. Benchmarktests mit weiteren RL-Algorithmen, mit denen Vor- und Nachteile dieser Algorithmen analysiert und interpretiert werden können.

Durch die Machine-Learning-Community gibt es eine große Anzahl an verfügbaren '*Gyms*', wie das in diesem Projekt genutzte: '*Super Mario Bros for OpenAI Gym*' [15]. Dieses ist ein von Christian Kauten zur Verfügung gestelltes SMB-Enviroment, welches die letzte Version von *OpenAI Gym* (Stand: 2020) unterstützt und weitere Features beinhaltet. So werden in diesem *Gym* unterschiedliche Render-Versionen und vordefinierte Eingabekombination (sog. ActionSpace) angeboten (vgl. 5.1).

Zur Verfügung steht dieses Enviorment aus urheberrechtlichen Gründen nur für Bildungszwecke, vgl. *Digital Millennium Copyright Act* (kurz: *DMCA*) [2].

3.2 PyTorch

Als Machine-Learning-Framework wurde *PyTorch* [4], ein bekanntes und speichereffizientes open source Framework, verwendet. Dieses wurde ursprünglich von einem *Facebook*-Forschungsteam für künstliche Intelligenz entwickelt und basiert auf der in *Lua* geschriebenen Bibliothek *Torch* [3]. [19]

3.3 Tensorboard und Ähnliche Bibliotheken

Zur Visualisierung der Lernergebnisse wurde *Tensorboard* [1] verwendet. Hierbei handelt es sich um ein *Tensorflow*-Toolkit, welches mit kleineren Einschränkungen ebenfalls mit *PyTorch* verwendbar ist. Es verfügt über einige diagrammspezifische Funktionalitäten und wird hauptsächlich für die Interpretation von ML-Experimenten verwendet [11].

Für einen größeren Funktionsumfang von *Tensorboard* mit *PyTorch* wurde die *TensorboardX*-Bibliothek verwendet. Diese bietet

neben einigen zusätzlichen Funktionen, auch Lösungen für Probleme, die aus fehlenden Abhängigkeiten zwischen diesen beiden Bibliotheken entstehen.

Für eine verbesserte Darstellung innerhalb der Commandline wurde die *TorchSummaryX*-Bibliothek verwendet. [5]

3.4 OpenCV

Für das sog. *Preprocessing* (deutsch: vorverarbeiten) (vgl. 5.3) wurde die mächtige open source Bildverarbeitungsbiliothek *OpenCV* verwendet. Diese verfügt neben der Unterstützung von unterschiedlichen Bild- und Farbformaten auch über performante Bildbearbeitungsfunktionen, wie bspw. das Umwandeln des Farbschemas, das Ändern von Helligkeit und Kontrast, sowie weiteren fortgeschritteneren Bildbearbeitungsfunktionalitäten. [14]

4 REINFORCEMENT-LEARNING-ALGORITHMUS

Für das Experiment selbst musste sich zunächst für einen Algorithmus entschieden werden. Dieser sollte möglichst unkompliziert, als auch auf der zur Verfügung stehenden Hardware (vgl. 2) möglichst performant sein, sodass der Trainingsprozess für ein SMB-Level wenige Stunden dauert. Hierfür wurde drei inkrementell komplexere Algorithmen implementiert und miteinander verglichen. Diese werden nachfolgend kurz erläutert.

4.1 Deep-Q-Network-Ansatz

Ausgehend von dem *Deep-Q-Network-Algorithmus* (kurz: DQN) zeichnete sich die Wichtigkeit der genannten Ziele bei der Auswahl des Algorithmus ab. Nach zwei Stunden Training kam es hier zu keinem Lernerfolg: Mario bewegte sich nicht, oder in die falsche Richtung. Grundlegend wird hier eine sogenannte *Q-Table* erzeugt, bei der den Zuständen die bestmöglichste Aktion zugeordnet wird (s.h. Abb. 1). [17]

Auch wenn davon auszugehen ist, dass mit diesem Algorithmus das Lernziel erreichbar ist, ist der Traingszeitrahmen für alle konkreten Netzwerke des Experimentes deutlich größer, wodurch die Verwendung dieses Algorithmus im Rahmen dieser Studienarbeit ungeeignet scheint.

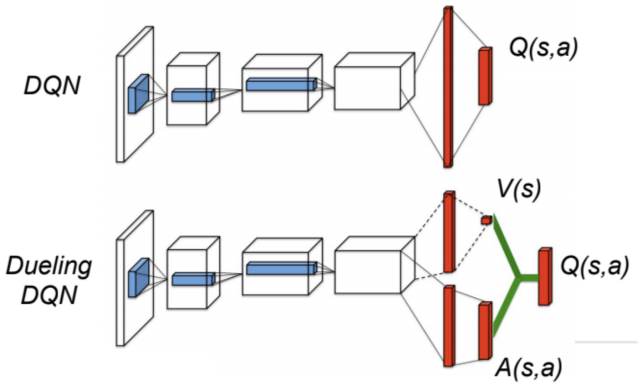

Abbildung 1: Aufbau DCN und DDCN [16]

Anschließend wurde die Tauglichkeit des *Duelling Deep-Q-Network-Algorithmus* (kurz: D-DQN) untersucht. Hierbei handelt es sich, wie der Name bereits andeutet, um eine Weiterentwicklung des vorherigen *DQN-Algorithmus*: der zuvor erläuterten *Q-Table* wird hierbei nun auch ein Wert hinzugefügt, welcher den aktuellen Zustand bewertet (s.h. Abb. 1).

Es stellte sich allerdings auch hier nach zwei Stunden Training kein sichtbarer Lernerfolg ein, wodurch auch dieser im Rahmen dieser Studienarbeit ungeeignet ist.

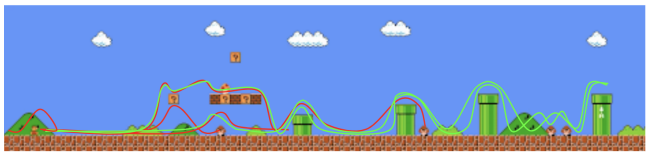

Abbildung 2: Mögliche Laufwege in SMB (Org.: [23])

Ursächlich für den geringen Lernerfolg ist hierbei die Komplexität des ursächlichen Problems bzw. der Aufgabe: Betrachtet man die Menge an Zuständen, so fällt auf, dass diese potentiell gegen unendlich läuft, ebenso ist es bei der Menge an möglichen Aktionen.

Betrachtet man Abbildung 2 so sind ein paar Laufwege von Mario eingezeichnet. Mit grünen Linien sind die erfolgreichen Laufwege markiert, in rot hingegen sind erfolglose Laufwege markiert.

Genannte gegen unendlich laufende Menge an Zuständen entspricht in der Grafik jeden Punkt auf einer Linie, die innerhalb des SMB-Enviorments auch gerendert werden kann. Die ebenfalls gegen unendlich laufende Menge an Aktionen entspricht hierbei der Anzahl an zeichenbaren Laufwegen, welche den Regeln des SMB-Enviroments folgen.

Diese beiden Mengen entsprechen den *Input* und den *Output* des Modells und definieren dadurch wichtige Eigenschaft des zur Lösung geeigneten Algorithmus.

Dies in Kombination mit der Tatsache des Trainierens auf einem einzigen Thread erklärt die Überlegenheit des nachfolgenden Algorithmus. [17], [24]

4.2 A3C-Ansatz

Ein *Reinforcement-Learning-Algorithmus*, der hingegen besser Performance bei dieser Komplexität verspricht ist der *Asynchronous-Advantage-Actor-Critic-Algorithmus* (kurz: A3C).

Im Gegensatz zur den vorher genannten Algorithmen wird hier keine *Q-Table* erzeugt. Stattdessen wird unter Berücksichtigung einer geschätzten Bewertung des aktuellen Zustandes vom sog. *Critic* (deutsch: Kritiker), eine Aktion ausgeführt vom sog. *Actor* (deutsch: Akteur). Der *Actor* und der *Critic* werden hierbei als separate Neuronale-Netzwerke definiert, die den exakt gleichen Input erhalten müssen.

Die Idee des A3C-Algorithmus lässt sich also mit folgender Metapher gut erklären.

> Let['s] imagine a small mischievous child (actor) [which] is discovering the amazing world around him, while his dad (critic) oversees him, to make sure that he does not do anything dangerous. Whenever the kid does anything good, his dad will praise and encourage

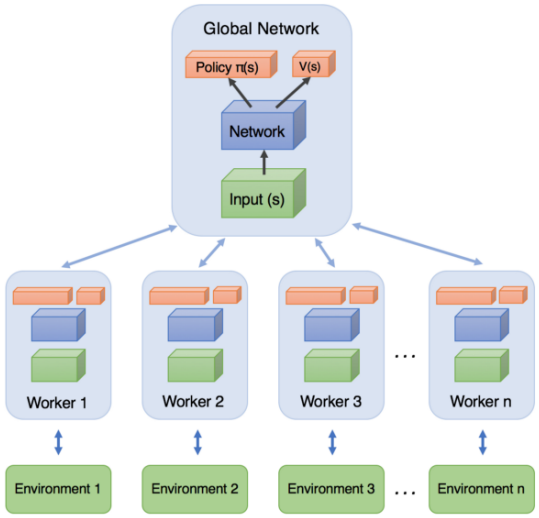

Abbildung 3: Aufbau A3C [13]

> him to repeat that action in the future. And of course, when the kid does anything harmful, he will get [a] warning from his dad. The more the kid interacts [with] the world, and takes different actions, the more feedback, both positive and negative, he gets from his dad. The goal of the kid is, to collect as many positive feedback as possible from his dad, while the goal of the dad is to evaluate his son's action better. In other word, we have a win-win relationship between the kid and his dad, or equivalently between actor and critic.[20]

Dieser Ansatz wird durch parallel agierenden Modelle erweitert, welche wiederum regelmäßig (in der Regel abhängig von der definierten Batchgröße) ein gemeinsames Model (sog. *globales Model*) aufbauen bzw. trainieren und sich auf dessen Lernstand selbst aktuell halten.

Mit diesem Algorithmus kam es bereits nach wenigen Minuten zu einem sichtbaren Lernerfolg, nach 2 Stunden wurde so das Lernziel des Experiment, das Abschließen eines Level, bereits erreicht. Es zeichnete sich also ab, dass dieser Algorithmus bestens geeignet ist für das eigentliche Experiment. [21], [13]

5 ENVIROMENT

Wie bereits erwähnt kommt in diesem Experiment ein SMB-Enviroment zum Einsatz (vgl. 3.1), nachfolgend wird die Konfiguration und der Einsatz dieses SMB-Enviroment genauer betrachtet.

5.1 Konfiguration

Auf Basis von Vorab-Experimenten wurde sich für die Standard Render-Version entschieden, welche eine minimal schlechtere Lernperformance - in diesem Fall das erstmalige Erreichen des Zieles - als die anderen bereitgestellten grafisch reduzierten Render-Versionen aufwies.

Ausgehend von den selben Vorab-Ergebnissen wurde sich für eine leicht reduzierte Eingabemenge (im '*Super Mario Bros for OpenAI Gym*' [15] 'Complex' genannt) entschieden. Diese beinhaltet im wesentlichen neben einer leeren Eingabemenge auch Kombinationen aus den Nach-Links- oder Nach-Rechts-Tasten mit der A- oder bzw. und B-Taste des *NES*-Controllers. Die weiteren reduzierten Eingabekombination hatten einen zustandsabhängigen Einfluss auf das Ergebnis, wodurch einige Level des Enviroments merklich schneller andere merklich langsamer erlernt wurden.

5.2 Weitere Modifikationen

Mittels *Wrapper-Pattern* wurde der *Output* des ursprünglichen Enviorments modifiziert. Nachfolgend werden diese Modifikationen von innen nach außen kurz erläutert.

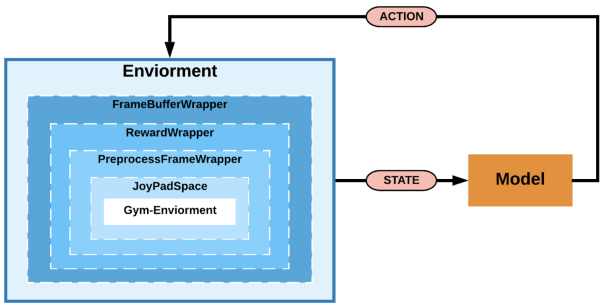

Abbildung 4: Schematischer Kontext des konkret implementierten Enviorments

5.2.1 *JoypadSpace.*

Der *JoypadSpace-Wrapper* ist eine von *OpenAI* zur Verfügung gestellte Implementation des *OneHot-Encodings* für das *NES*.

5.2.2 *PreprocessFrameWrapper.*

Der *PreprocessFrameWrapper* vorverarbeitet den ursprünglichen berechneten Frame (vgl. 5.3).

5.2.3 *RewardWrapper.*

Der *RewardWrapper* überschreibt den ursprünglich berechneten Rewardwert (vgl. 5.4).

5.2.4 *FrameBufferWrapper.*

Der *FrameBufferWrapper* fasst einige berechnete Frames in ein Array zusammen, um so dem Model die Chance zu bieten Bewegungen zu erkennen.

5.3 Preprocessing

Das *Preprocessing* dient zur Vorverarbeitung des Frames. Dadurch soll das Bild so angepasst werden, dass das zutrainierende Model damit bestmöglich rechnen kann. In der konkreten Implementation kann dieser Schritt in wiederum vier Teilschritte unterteilt werden, die nachfolgend kurz begründet werden.

5.3.1 *Zuschneiden.*

Zuerst wird der originale 240x256 Pixel große Frame auf 200x256 Pixel verkleinert. Konkret werden einige Pixel oben und unten weggeschnitten, welche (fast) keine Relevanz für den Spielverlauf haben.

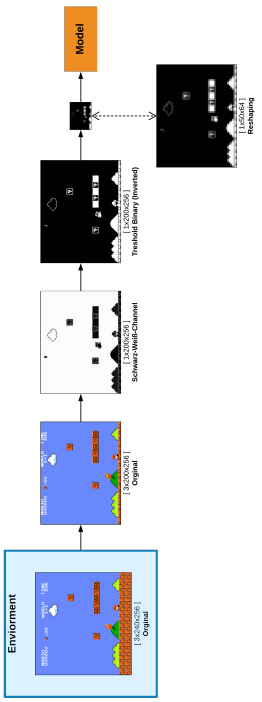

Abbildung 5: Schematische Preprocessing

5.3.2 Schwarz-Weiß.
Im nächsten Schritt wird der Frame zu einem verlustfreien Schwarz-Weiß-Bild vereinfacht.

5.3.3 Schwarz-Oder-Weiß.
Anschließend wird der Frame mittels *Binary-Treshhold* zu Schwarz- oder Weißenfarbtönen vereinfacht. Damit diese sich auch optisch von dem vorherigen Schritt leicht unterscheiden lassen, werden hierbei die Farben invertiert.

5.3.4 Verkleinern.
Abschließend wird das Bild im korrekten Seitenverhältnis verkleinert. Dies hat u.a. Einfluss darauf wie viel Speicher das Model in allen Ebenen benötigt und wurde im Rahmen des Experimentes auf 50x64 Pixel gesetzt (vgl. 2) - dies entspricht nahezu der minimalen informationsverlustbehafteten Verkleinerung des Bildes, die im Rahmen diverser Experimente festgelegt worden ist.

5.4 Reward

Der *Reward* definiert das mathematisch entstehende Verständnis des Modells über das Enviorment. Hierfür gibt es in der Regel zwei unterschiedliche Ansätze, wie dieser *Reward* vergeben werden kann: Per *Sparse Reward* oder mittels *Reward Shaping*. Beim sog. *Sparse Reward* wird der *Reward* nur beim Erreichen wesentlicher Ziele vergeben. Beim *Reward Shaping*, welches für das Experiment gewählt wurde, wird hingegen durch kontinuierliches Vergeben des *Reward* das mathematische Verständnis des Modells sehr stark beeinflusst, wodurch eine starke Abhängigkeit zwischen der Qualität der *Reward-Funktion* und dem Lernergebnisses ensteht.

Im der konkreten Implementation besteht der *Reward* aus 4 Werten, die nach einer Addition in einen geringeren Wertebereich verschoben werden:

$$F_{Reward} = (\Delta X + \Delta Zeit + R_{Ziel} + R_{Leben}) \ / \ 10 \tag{1}$$

Dem Delta der zurückgelegten Strecke addiert mit der verstrichenen Zeit, zusätzlich den fest definierten Belohnungswerten für das erreichen des Zieles bzw. für das verlieren eines Lebens. Diese Variablen lassen sich wie folgt definieren:

$$\Delta X = X(n+1) - X(n) \tag{2}$$

$$\Delta Zeit = T(n+1) - T(n) \tag{3}$$

$$R_{Ziel} = 45 \ if \ goal \ achieved \ else \ 0 \tag{4}$$

$$R_{Leben} = -45 \ if \ life \ lost \ else \ 0 \tag{5}$$

Ziel des Modells für die *Reward-Funktion* ist es also eine möglichst große Distanz in geringer Zeit zu erreichen, bei denen im bestenfalls das Ziel erreicht wird und idealerweise kein Leben verloren wird.

Experimente mit weiteren Faktoren wie etwa einer Belohnung für das Sammeln von Münzen oder dem Verhindern des 'Stehenbleibens' hatten meist negative Auswirkungen auf das eigentliche Hauptziel.

6 NETZWERK-ARCHITEKTUR

Die zugrundeliegende Netzwerk-Architektur für den *A3C-Algorithmus* erfordert zwei separate Netzwerke (*Actor* und *Critc*), die wiederum sich den selben *Input* teilen. Dieser *Input* entsteht durch visuelle Verarbeitung mittels *Convolutional-Netzwerkes* (kurz: CNN) und anschließend durch rückgekoppelte Verarbeitung mittels *Rekurrentem-Netzwerkes* (kurz: RNN) mit dem Ziel der reihenfolgeabhängigen Verarbeitung. Auf all diese Bestandteile des implementierten Neuronalen Netzwerkes wird nachfolgend eingegangen (vgl. Abb. 10).

6.1 Convolutional-Netzwerk

Wie erwähnt dient das *CNN* dazu grafische nicht näher definierte Muster innerhalb des Frames zu deuten. Es wurden hierbei zwei unterschiedliche Architekturen definiert, die nachfolgend kurz erklärt werden. Beide Ansätze hatten zum Ziel die *Tensorgröße* zu minimieren, um dadurch die limitierte Hardware (vgl. 2) bestmöglichst auszunutzen.

6.1.1 Naive CNN.
Bei dem Naiven Ansatz des *CNN* wird ein initial 320 Channel großer Tensor konstant um 80 Channel, also ursprünglich um 25%, verkleinert. Hintergrund hierfür ist, neben dem Ziel den Tensor klein zuhalten, dass mehrere Frame-Features zu einem einzigen Signal inkrementell zusammengefasst werden sollen. Als Kernelgröße wurde 3x3 gewählt, bei einem Stride von 2, welcher u.a. den Tensor konstant reduziert.

Während der Entwicklungsphase gab es keine wesentlichen Unterschiede bei ähnlich hoher Start-Channel-Anzahl und Channel-Reduktionsfaktor, weshalb die Größen so gewählt wurden, dass der verfügbare Arbeitsspeicher voll ausgelastet wurde.

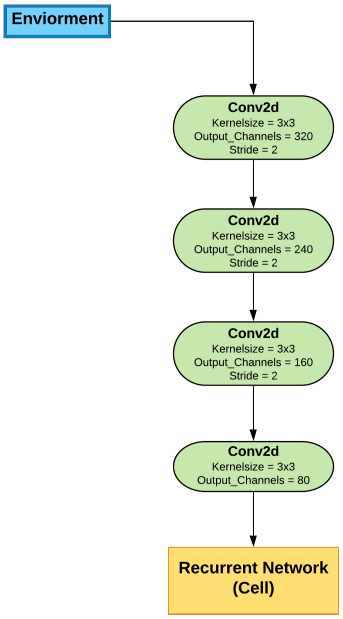

Abbildung 6: Naive CNN-Architektur

6.1.2 Deep(ish)-CNN.

Ein weiterer Ansatz für den *CNN*-Teil ist inspiriert von den *GoogLeNet*-Blöcken [12]. Im Gegensatz zu den vorherigen *CNN*-Ansatz kamen hier nach jedem *CNN*-Layer eine *ReLU*-Aktivierungsfunktion zum Einsatz. Wie auch in dem naiven *CNN*-Ansatz wird hier mittels Stride die Tensorgröße effektiv reduziert.

Abweichend zu *GoogLeNet* wurden hier im zweiten Branch drei 3x3 große Kernel und im dritten Branch zwei 5x5 große Kernel verwendet. Eine Reduktion der Rechenleistung durch das Ersetzen der 5x5 Kernel mittels zwei 3x3 Kernel ist hierbei aufgrund der Strides nicht möglich. Die Channelanzahl der einzelnen Branches wird unterschiedlich stark auf 32 reduziert, wodurch im letzten Verknüpfungschritt 128 Channel erreicht werden.

Die wesentlichen Abweichungen hinsichtlich der Kernelgröße und -anzahl sind hauptsächlich experimentell zu erklären. Geringe Änderungen hierbei hatten meist eine längeren Lernzeit für ähnliche Lernergebnisse zufolge. Die Anzahl an Channels ist ähnlich wie bei dem Naiven *CNN*-Ansatz mit der Hardware-Ausnutzung begründet (vgl. 2).

6.2 Rekurrent-Netzwerk

Hinsichtlich des Rekurrent-Netzwerk stellte es sich als problematisch heraus, dass zu dem Zeitpunkt, an dem ein *CNN-Output* generiert wurde das Training noch nicht abgeschlossen wurde, weshalb kein allgemeines *RNN* implementiert wurde. Stattdessen wurde

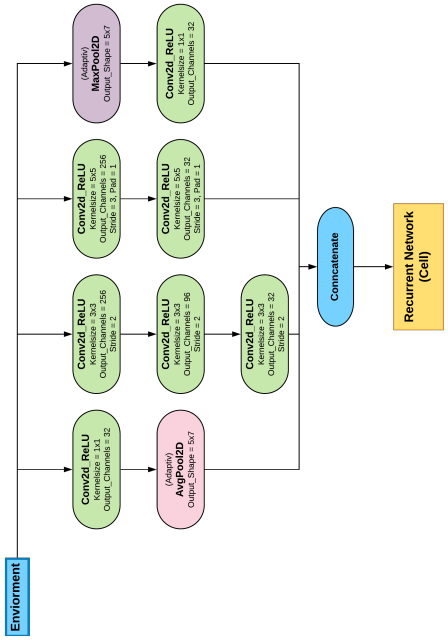

Abbildung 7: Deep(ish)-CNN-Architektur

eine Zelle implementiert, die als Teilinput den Output der selben Zelle zu dem vorherigen Schritt des Enviorments erhält.

Da der *RNN*-Teil für das absolvieren SMB ähnlicher Enviormente eine herausragende Rolle spielt, wurde die Channelanzahl innerhalb des kompletten Netzwerk maximiert. Da dies allerdings dazu führt, das bei der zur Verfügung stehenden limitierten Hardware, ein stark ansteigender Arbeitsspeicherverbrauch entsteht, wodurch diverse Abhängigkeiten innerhalb des Netzwerks und dem Enviroment ensteht, fehlten zum Zeitpunkt des Experimentes weitere Untersuchungen hinsichtlich der optimalen Werte.

6.2.1 LSTM-RNN.

Ein Ansatz für das *RNN* war die Verwendung der *LSTM-Zelle*. Hier wird im Vergleich zum folgenden *GRU*-Ansatz zusätzlich der *Hidden-State* ebenfalls pro Berechnungsschritt übergeben. (Die Input-Channelanzahl ist Abhängig von der Output-Channelanzahl des *CNN*-Teils.)

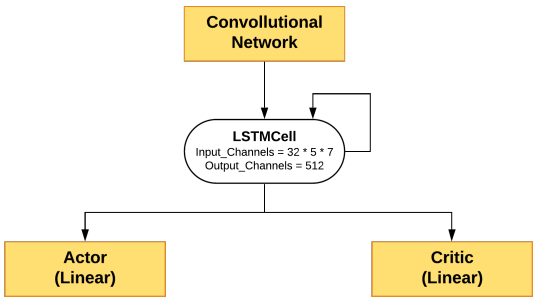

Abbildung 8: LSTM-Architektur

6.2.2 GRU-RNN.

Ähnlich wie die *LSTM-RNN*-Variante wurde die *GRU*-Zelle verwendet. (Die Input-Channelanzahl ist Abhängig von der Output-Channelanzahl des *CNN*-Teils.)

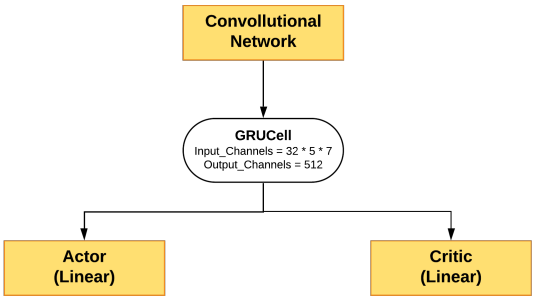

Abbildung 9: GRU-Architektur

6.3 Actor und Critic

Der *Actor*- und *Ciritc*-Teil des neuronalen Netzwerkes spielt eine entscheidende Rolle für das Bestimmen der Aktionen. Als *Linearer-Layer* implementiert unterscheiden sich die beiden hauptsächlich im Output: Der *Kritiker* berechnet einen einzigen Wert - die Bewertung des aktuellen Zustandes (vgl. 3).

Der *Akteur-Output* hingegen entspricht der Anzahl der verfügbaren Aktionen (vgl. 5.1 und beinhaltet die Wahrscheinlichkeit das diese die bestmöglichste Aktion ist. Diese wird im weiteren Verlauf nach der wahrscheinlichsten Aktion gefiltert und zurück zu einer, für das Enviorment verständlichen Aktion, umgewandelt.

6.4 Loss-Funktion

Damit sich die zuvor aufgeführten *Linearen-Layers* während des Trainings unterschiedlich entwickeln, werden diese in der Verlustfunktion mit unterschiedlichen Vorzeichen addiert abzüglich des konstanten Anteils des vorherigen Verlusts. Dieses Ergebnis wird innerhalb des Netzes schließlich back propagiert.

$$loss = -loss_{actor} + loss_{critic} - (beta * loss_{entropy}) \quad (6)$$

6.5 Hyperparameter

Für das komplette Experiment wurden im üblichen Wertebereich liegende *Hyperparamter* verwendet:

$$lr = 0.0001 \quad (7)$$

$$gamma = 0.9 \quad (8)$$

$$beta = 0.01 \quad (9)$$

7 EXPERIMENT UND ERGEBNISSE

Für das folgende Experiment wurden auf Basis der Hardware (vgl. 2) fünf parallel Trainings-Threads gestartet. Diese arbeiteten jeweils eine vorgegebene Anzahl von Epochen ab. Zusätzlich zu den

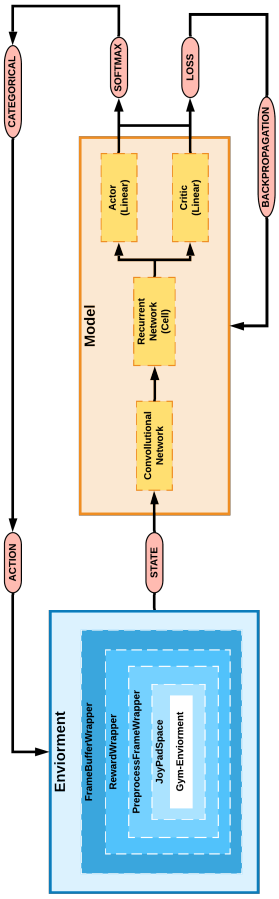

Abbildung 10: Vollständiger Kontext des Modells

Trainings-Thread, wurde ein Thread gestartet, welcher den aktuellen Zustand des globalen Modells wiederspiegelte. Während des Trainings lief zudem zur Live-Ansicht der Lernergbnisse im Hintergrund der *Tensorboard-Service*.

In den nachfolgenden Abbildungen 12, 13 und 14 sind die Farben jeweils nach der Legende (Abb. 11) gewählt. Zusehen sind hier die Ergebnisse nach 3500 Trainingsepisoden bei 5 Trainingsthread für das allererste Super Mario Bros. Level. Insgesamt wurden also 17500 Episoden pro Netzwerk trainiert. Auf den X-Achsen ist die verstrichene Zeit aufgeführt, auf der Y-Achse der für das Diagramm spezifisch erreichte Werte. Die Graphen sind zur besseren Verständlichkeit mit einem Smoothingfaktor von 85% dargestellt.

7.1 Direkter Vergleich von Netzwerken

Nachfolgend werden die Ergebnisse der jeweiligen Netzwerke kurz aufgeführt.

7.1.1 DCN + GRU.

Das *Deep(ish)-Convolutional-Netzwerk* in Kombination mit der *GRU-Zelle* erwies sich als am schnellsten lernende Kombination. Bereits nach ca. 27 Minuten wurde das Ziel erstmalig erreicht; das Lernziel war hier anschließend nach ca. 34 Minuten erreicht.

Im weiteren Verlauf verließ dieses Netzwerk allerdings zweimal den zielführenden Weg für ca. 3 Minuten und Aufgrund des Zeitpunkt nicht näher definierbaren Zeitrahmen, innerhalb des ca. einstündigen Trainingszeitraum.

Auffällig für dieses Netzwerk ist die frühe stabile starke Lernperformance mit einem Vorsprung von bis zu ca. 5 Minuten vor den anderen Kombinationen.

7.1.2 CNN + GRU.

Das einfache *CNN* in Kombination mit der*GRU-Zelle* erwies sich hingegen als langsam lernende Kombination. Auch wenn nach 29 Minuten das Ziel erstmalig erreicht wurde, wurde das Lernziel erst nach ca. 35 Minuten erreicht.

Im weiteren Verlauf verließ dieses Netzwerk allerdings dreimal den zielführenden Weg für insgesamt 7 Minuten.

Auffällig für dieses Netzwerk ist die bereits früh auftauchende Instabilität.

7.1.3 CNN + LSTM.

Als äußerst langsam lernendes Netzwerk erwies sich die Kombination aus naiven *CNN* mit *LSTM-Zelle*. Auch wenn nach 27 Minuten das Ziel erstmalig erreicht werden konnte, wurde das Lernziel erst nach ca. 36 Minuten erreicht.

Im weiteren Verlauf erwies es sich zudem als tendenziell instabil und wich dreimal von zielführenden Weg ab für insgesamt ca. 5 Minuten.

7.1.4 DCN + LSTM.

Die Kombination aus Deep(ish)-Convolutional-Netzwerk mit *LSTM-Zelle* erwies sich als langsamstes lernendes Netzwerk. Nach ca. 33 Minuten wurde das Ziel erstmalig erreicht, nach 39 Minuten wurde das Lernziel erreicht.

Im weiteren Verlauf verließ dieses Netzwerk kein einziges Mal den zielführenden Weg.

Auffällig für diese Kombination ist die für den Trainingszeitraum perfekte Stabilität und die ca. 58 Minuten auftrettende wachsende Führung des insgesamt erreichten Rewardes.

Abbildung 11: Legende für die Ergebnisse

7.2 Zwischenfazit

Ausgehend von vorherigen Test in Kombinationen mit weiteren kleineren Experimenten lässt sich feststellen, dass die Lernstabiltät eine übergeordnete Rolle zum lösen des SMB-Problemes aufweist. Denn testet man schnelle Netzwerke wie die Kombination von *DCN* und *GRU-Zelle* so lässt sich feststellen, dass nicht immer das Ziel erreicht werden kann, ähnlich gut bzw. schlecht verhalten sich die anderen beiden Netzwerkarchitekturen. Hierfür ist u.a. die Struktur der Level, bestehend aus Sackgassen, Engstellen, Fallen, Sprungpassagen oder ähnliche in Kombination mit dem Rewardshaping verantwortlich.

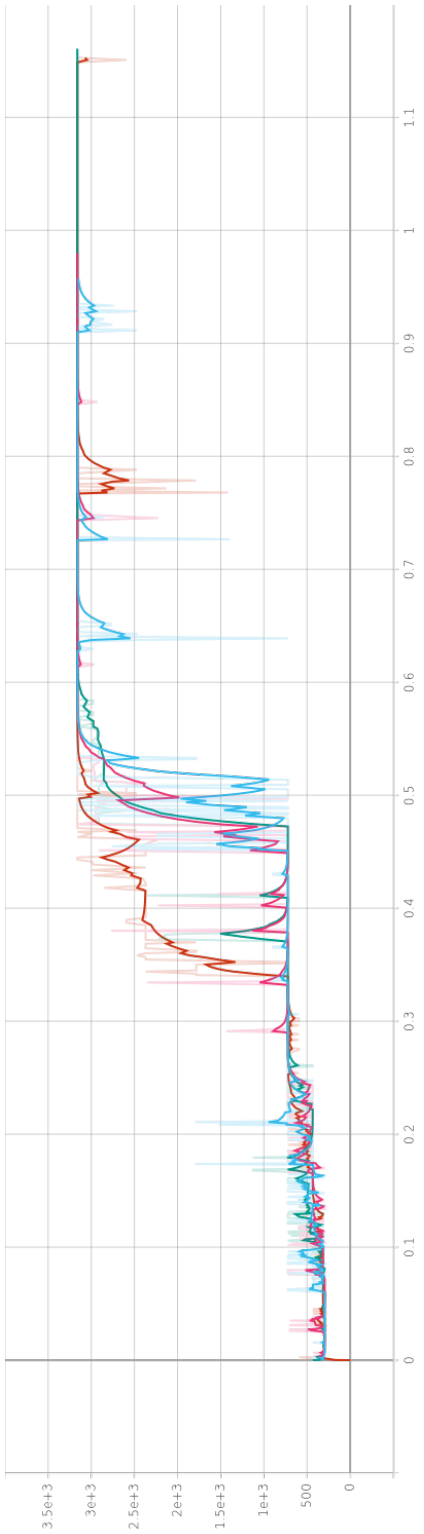

Abbildung 12: Ergebnis: Erreichte X-Position

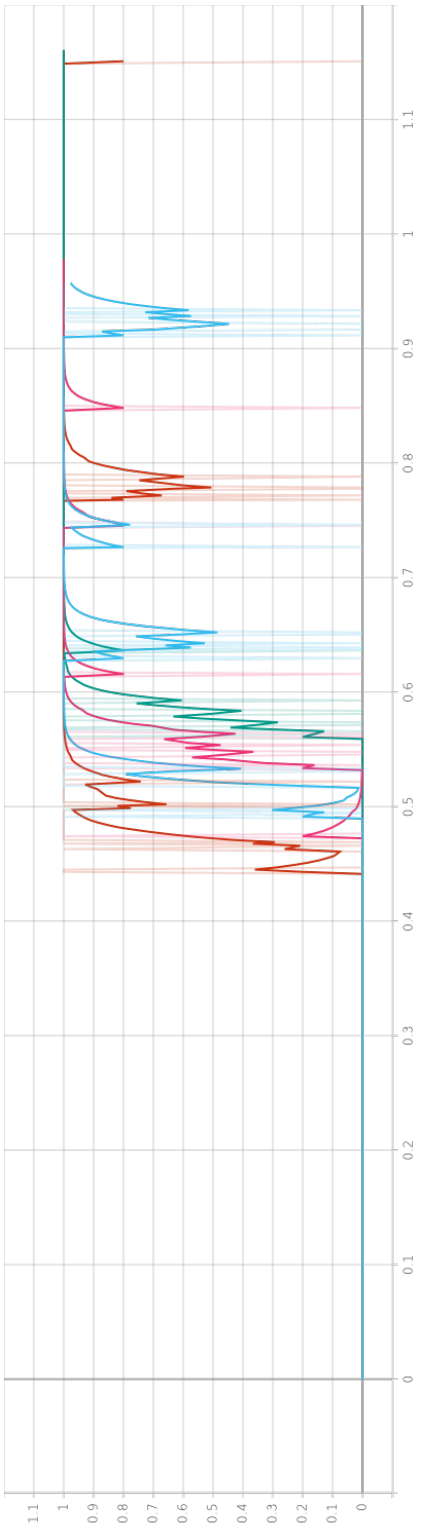

Abbildung 13: Ergebnis: Erreichte Flagge

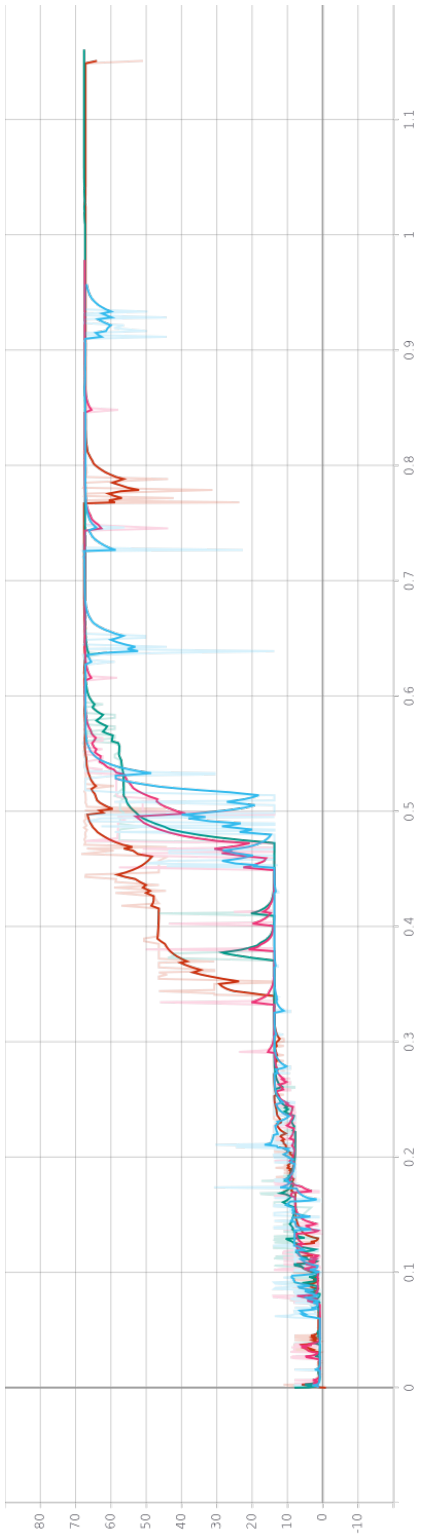

Abbildung 14: Ergebnis: Insgesamt erreichter Reward

Vergleicht man die Aufnahmen der Trainingsdurchgänge (zu finden im Projektrepository [10]) so fällt auf, dass es auch nach dem Training für alle Netzwerkkombination noch Verbesserungspotential innerhalb von wenigen Millisekunden besteht, geschuldet dadurch das Mario an einzelnen Hindernissen kurz zum stoppen kommt.

Abschließend wurden weitere Test mit dem erfolgversprechendem Netzwerk, bestehend aus *Deep(ish)-Convolutional-Netzwerk* und *LSTM-Zelle*, durchgeführt um weitere Aussagen hinsichtlich der Lernperformance zu treffen bzw. zu bestätigen.

7.3 Weiterführende Beobachtungen

Nachfolgend werden die restlichen drei Level der ersten SMB-Welt mit *DCN* und *LSTM-Zellen* aufgeführt.

7.3.1 Level 2.

Beim zweiten SMB-Leve erreichte keines der Netzwerkkombinationen aus dem ersten Experiment (vgl. 7.1) das Ziel. Alle blieben an der selben Stelle hängen: Einer Sackgasse, wo der eigentliche Lösungsweg durch mehrere Gegner für eine geringe Zeit blockiert wird.

Ursächlich hierfür ist das *Rewardshaping*, welches den Lösungsweg nicht mit längeren Wartezeiten definiert. Es zeigte sich, dass mit einem weitestgehend randomisierten Reward diese Stelle über einen entsprechend längeren Zeitraum von den meisten Kandidaten lösbar war.

7.3.2 Level 3.

Nachfolgend werden die Ergebnisse aus Abbildung 15 kurz analysiert. Es wurden 5 Threads mit jeweils 2500 Episoden trainiert.

Level 3 besteht hierbei im wesentlich aus Sprungpassagen unterschiedlicher Schwierigkeitsgrade, welche dem ersten Level tendenziell ähnelt, wodurch sich die Ähnlichkeit der Lernperformance erklären lässt.

Nach ca. 31 Minuten wurde erstmalig das Ziel erreicht, nach weiteren 3 Minuten war das Lernziel erreicht. Ähnlich wie in dem vorherigen Experiment fällt die Lernstabilität auf. Es ist auch anzumerken, dass hier ebenfalls nach Erreichen des Lernzieles ein weiterer Lernerfolg messbar ist in Form des insgesamt verdienten Rewardes.

7.3.3 Level 4.

Nachfolgend werden die Ergebnisse aus Abbildung 16 kurz analysiert. Es wurden 5 Threads mit jeweils 7500 Episoden trainiert.

Das Level 4 beinhaltet das erste Auftreten des *Bowser* Gegners und besteht aus einigen sehr schwierigen Passagen wie bspw. schwierigen Sprungpassagen, tödlichen Sackgassen, feindliche Projektile und sich bewegende Hindernisse.

Nach ca. 54 Minuten wurde das Ziel erstmalige erreicht, nach ca. 72 Minuten stellte sich das Lernziel ein, unterbrochen durch einen ca. 5 Minütigen Misserfolg.

Auffällig ist hierbei die relativ instabile Lernrate zu Beginn des Trainings, welche auf die Schwierigkeit des Levels zurückzuführen ist. Auffällig ist auch die minimal instabile Rewardrate nach dem Erreichen des Lernzieles.

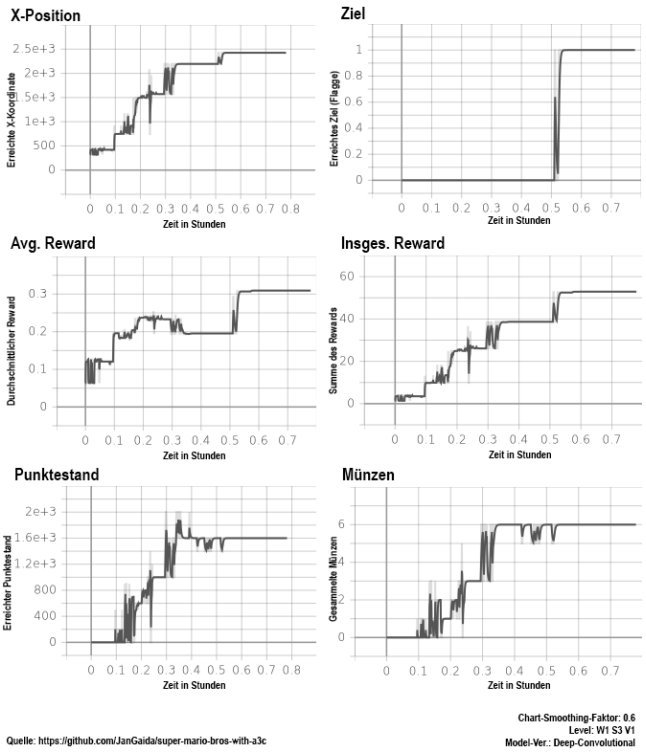

Abbildung 15: Ergebnis: Welt 1 Level 3

8 RESÜMEE

Fasst man die Ergebnisse dieser Studienarbeit also zusammen, lässt sich zunächst erst einmal feststellen, dass SMB für viele RL-Algorithmen eine Herausforderung darstellt. Es lässt sich zwar vermuten, dass mit leistungsstärkerer Hardware noch schneller das Lernziel erreicht werden kann, dennoch wird aufgrund der tendenziell geringen Explorationsrate bei den getesteten RL-Algorithmen nicht der bestmöglichste Lösungwegs zeitnah gefunden.

Für die Suche nach diesen bestmöglichen Lösungweg erwies sich die Stabilität des Lernerfolgs als äußerst wichtig, auch wenn diese dazu führt, dass das eigentliche Lernziel später erreicht wird. Desweiteren lässt sich feststellen, dass diese Stabilität des Lernerfolgs auch dazuführen kann, dass vorallem schwierige Passagen schneller gelöst werden kann.

Hinsichtlich der verwendeten neuronalen Schichten erwies sich die GRU-Zelle als schnell lernfähig, wohingegen die LSTM-Zelle sich als stabil erwies. Hinsichtlich des CNN erwies sich eine in Branchen unterteilte Variante performanter als eine Serielle Variante.

Auch wenn für ein aussagekräftiges Studienergebnis weitere Tests notwendig sind, lassen doch wichtige Charakteristika der verwendetet neuronalen Netzwerkarchitekturen feststellen, welche wiederum aktuelle Forschungsgebiete streifen.

LITERATUR

[1] [n.d.]. Tensorboard. https://www.tensorflow.org/tensorboard [Online; accessed 16-July-2020].

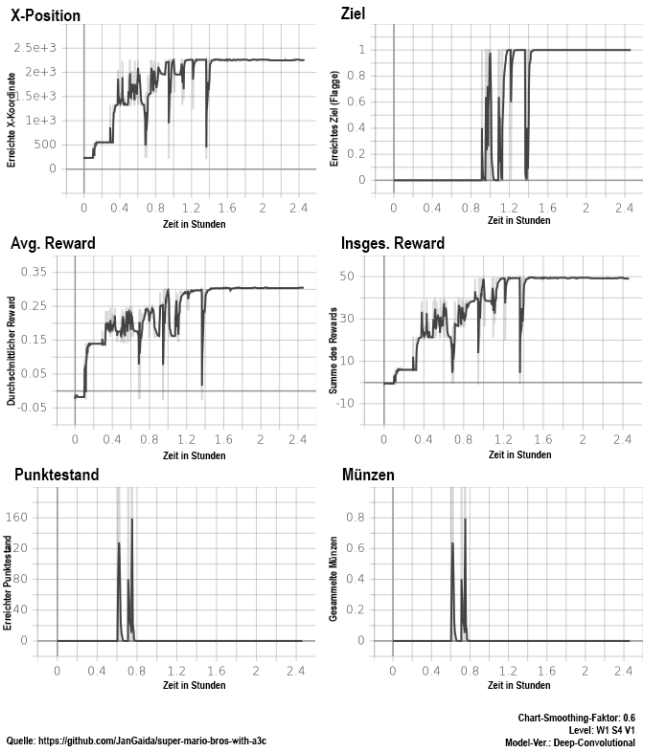

Abbildung 16: Ergebnis: Welt 1 Level 4

[2] 1998. Digital Millenium Copyright Act. https://web.archive.org/web/20000824110859/http://www.eff.org/IP/DMCA/hr2281_dmca_law_19981020_pl105-304.html [Online; accessed 16-July-2020].

[3] 2002. Torch. http://torch.ch/ [Online; accessed 16-July-2020].

[4] 2016. PyTorch. https://pytorch.org/ [Online; accessed 16-July-2020].

[5] Namhyuk Ahn. 2019. torchsummaryX. https://github.com/nmhkahn/torchsummaryX

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *ArXiv* abs/1606.01540 (2016).

[7] Wikipedia contributors. 2020. Wikipedia, The Free Encyclopedia. https://de.wikipedia.org/wiki/Super_Mario_Bros. [Online; accessed 16-July-2020].

[8] DeepMind. 2020. AlphaGo the story so far. https://deepmind.com/research/case-studies/alphago-the-story-so-far

[9] Florian Fallenbüchel. [n.d.]. Anwendungen von Reinforcement Learning. https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/643877180/fallenbuechel_anwendungen-reinforcement-learning-report.pdf

[10] Jan Gaida. 2020. Deep-Q-Learning mit 'Super Mario Bros' und A3C. https://github.com/JanGaida/super-mario-bros-with-a3c

[11] Google. 2020. Why TensorFlow. https://www.tensorflow.org/about

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[13] Arthur Juliani. [n.d.]. Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C). https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2 [Online; accessed 16-July-2020].

[14] Abid K. 2013. Introduction to OpenCV-Python Tutorials. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html#intro

[15] Christian Kauten. 2018. Super Mario Bros for OpenAI Gym. GitHub. https://github.com/Kautenja/gym-super-mario-bros

[16] Volodymyr Mnih. [n.d.]. Deep Q-Networks. https://drive.google.com/file/d/0BxXI_RttTZAhVUhpbDhiSUFFNjg/view [Online; accessed 16-July-2020].

[17] Parsa Heidary Moghadam. 2019. Deep Reinforcement learning: DQN, Double DQN, Dueling DQN, Noisy DQN and DQN with Prioritized Experience Replay. https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

[18] Andrew Ng. 2020. What is Machine Learning? https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v

[19] Chi Nhan Nguyen. 2019. Neuronale Netzwerke mit PyTorch entwickeln, trainieren und deployen. https://entwickler.de/online/machine-learning/neuronale-netzwerke-pytorch-579898880.html

[20] Viet Nguyen. 2019. [PYTORCH] Asynchronous Advantage Actor-Critic (A3C) for playing Super Mario Bros. https://github.com/uvipen/Super-mario-bros-A3C-pytorch

[21] Chris Nicholls. 2017. Reinforcement learning with the A3C algorithm. https://cgnicholls.github.io/reinforcement-learning/2017/03/27/a3c.html

[22] OpenAI. 2019. OpenAI Five. https://openai.com/projects/five/

[23] RB1196. [n.d.]. Deep Q-Networks. https://mario.fandom.com/wiki/World_1-1_(Super_Mario_Bros.)?file=SMB_World_1-1_NES_1.png [Online; accessed 16-July-2020].

[24] Chris Yoon. 2019. Double Deep Q Networks, Noisy DQN and DQN with Prioritized Experience Replay. https://towardsdatascience.com/double-deep-q-networks-905dd8325412

Abbildungsverzeichnis