Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jerom Schult

Deep Q Learning mit Künstlichen Neuronalen Netzen für Markov-Entscheidungsspiele

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jerom Schult

Deep Q Learning mit Künstlichen Neuronalen Netzen für Markov-Entscheidungsspiele

Bachelorarbeit eingereicht im Rahmen der Bachelorarbeit

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck

Eingereicht am: 14. Dezember 2018

Jerom Schult

Thema der Arbeit
Deep Q Learning mit Künstlichen Neuronalen Netzen für Markov-Entscheidungsspiele

Stichworte
Künstliche Neuronale Netze, Q Learning, Deep Learning, Markov-Entscheidungsspiele, CartPole

Kurzzusammenfassung
Im Fokus dieser Thesis steht die Frage inwieweit Künstliche Neuronale Netze ohne Verwendung von Domänenwissen Probleme lösen können. Dazu wird eine Architektur eines Modells, basierend auf Deep Learning mit künstlichen Neuronalen Netzen, zur Lösung des Spiels CartPole aufgestellt und implementiert. Zur Verbesserung der Modellqualität wird das Lernverfahren um Double Q Learning und Memory Replay erweitert. Durch das Erreichen eines Referenzwerts wird gezeigt, dass für die gewählte Problemdomäne eine konkurrenzfähige Lösung erreicht werden konnte und das Modell selbstständig in der Lage ist eine adäquate Strategie zu entwickeln.
Jerom Schult

Title of the paper
Deep Q Learning with Artificial Neural Nets for Markov Decision Processes

Keywords
Artificial Neural Nets, Q Learning, Deep Learning, Markov Decision Processes, CartPole

Abstract
This thesis focuses on the question to what extent Artificial Neural Nets are capable of solving problems without the use of specific domain knowledge. For that an architecture of a model to solve the CartPole game, based on deep learning with Artificial Neural Nets, is designed and implemented. To improve the quality of the built model the learning process is extended by Double Q Learning and Memory Replay. Through reaching a defined reference value it is shown, that it was possible to build a competitive solution for the chosen domain and the model was independently able to learn an adequate strategy.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Künstliche Intelligenz gils als einer der spannendsten und zugleich furchteinflößendsten Teilbereiche der Informatik. Vorstellungen von intelligenten Supercomputern decken dabei das gesamte Spektrum von Utopien bis Dystopien ab. Selbstfahrende Autos, perfekte Übersetzungscomputer im Ohr und fehlerlos arbeitende Androiden-Ärzte für Operationen und Anamnesen in Krankenhäusern stehen Vorstellungen von bösartigen, autonomen Killermaschinen mit der Absicht die Menschheit auszulöschen gegenüber. Auch die Wissenschaft beschäftigt sich bereits seit langem mit der Frage wie eine künstliche Intelligenz zu definieren sei und wie die unterschiedlichen Definitionen zu erreichen sind. Durch die mit der Zeit steigende Rechenkapazität moderner Computer erleben Teilbereiche des maschinellen Lernens, darunter Künstliche Neuronale Netze (KNN), aktuell einen massiven Aufschwung.
Supervised- und unsupervised Learning sind dabei auf große Datenbestände in ausreichender Qualität angewiesen. Beim Reinforcement Learning hingegen lernt das Modell direkt auf der Problemdomäne und ist dabei nicht zwangsläufig auf Expertenwissen angewiesen.
So konnte das Team von DeepMind ein Modell mit Reinforcement Learning trainieren, das in der Lage war klassische Atari-Spiele auf übermenschlichem Niveau zu spielen ohne an die einzelnen, sehr unterschiedlichen Spiele angepasst zu werden (siehe Mnih u. a. (2015)).
Sollte es möglich sein mit allgemeinen, nicht an einzelne Probleme angepassten Modellen immer komplexere Anwendungsfälle der realen Welt zu lösen, könnte das einen großen gesellschaftlichen Wandel mit sich bringen. Ausgehend von den Erfolgen von DeepMing wird diese Arbeit sich mit den benutzten Techniken und der Fähigkeit Künstlicher Neuronaler Netze zur Problemlösung ohne die Verwendung von Domänenwissen auseinandersetzen.

1.2 Zielsetzung

In dieser Thesis wird eine Variante des Lernens direkt auf einer Problemdomäne betrachtet, die ohne Verwendung von Domänenwissen und unter Einsatz von Künstlichen Neuronalen Netzen versucht ein Spiel zu lösen. Bei dem Spiel handelt es sich um das Spiel Cartpole, eine in

der Literatur gängige Simulation einer regelungstechnischen Aufgabe, und als Lernmethode wird Deep Q Learning mit Künstlichen Neuronalen Netzen benutzt.
Der Fokus liegt hierbei auf der Frage inwieweit sich Neuronale Netze als non-lineares Verfahren zur Lösung eines Problems ohne die Verwendung von Domänenwissen eignen. Dazu wird versucht das Spiel CartPole nach der in Kapitel 2.8.2 vorgestellten Definition zu lösen und darauf aufbauend eine konkurrenzfähige Lösung herzuleiten.

1.3 Aufbau

Dazu wird in Kapitel 2 eine Analyse der Problemdomäne vorgenommen und die wichtigsten theoretischen Konzepte erläutert. Darauf folgt eine Erläuterung der zentralen Fragestellung dieser Arbeit.
In Kapitel 3 wird eine Architektur beschrieben, die eine Experimentalumgebung zur Beantwortung der Fragestellung liefert.
In Kapitel 4 wird eine Evaluation der gesammelten Messungen vorgenommen und aufgetretene Probleme beschrieben.
Zuletzt folgt in Kapitel 5 ein Fazit über die Ergebnisse dieser Thesis und ein Ausblick.

2 Q Learning mit Künstlichen Neuronalen Netzen

Seit einigen Jahren kommt Machine Learning wieder eine erhöhte Aufmerksamkeit zuteil. Zum Teil schon lange bekannte Lernverfahren, die früher aufgrund mangelnder Rechenleistung nicht sinnvoll einzusetzen waren, erfreuen sich aktuell einer immer größeren Beliebtheit.

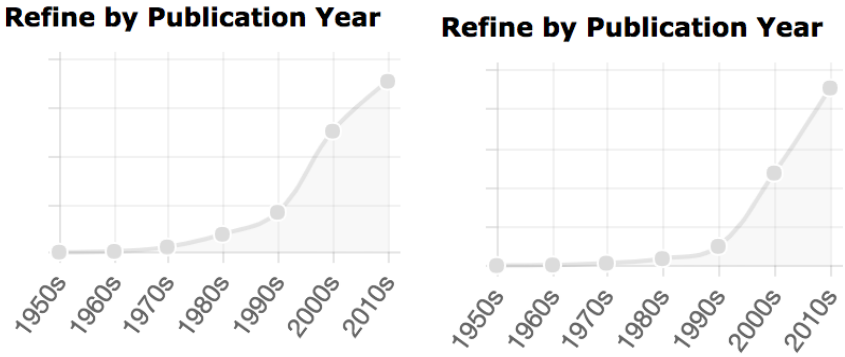

Abbildung 2.1: Suchhäufigkeit nach „Artificial Neural Net“ (links) und „Q Learning“ (rechts) in der ACM Digital Library (vergl. ACM Digital Library (2018))

Entscheidende Fortschritte in der Domäne hat das Team von DeepMind (vergl. DeepMind (2018)) in den letzten Jahren erzielt. Als Erste haben sie es mit Deep Learning, also der Benutzung Künstlicher Neuronaler Netze (KNN) geschafft die besten „Go“-Spieler der Welt zu schlagen (vergl. Silver u. a. (2017)). Das Brettspiel „Go“ ist komplexer als Schach und galt lange Zeit als nicht zu erreichender Meilenstein der computergestützten Intelligenz.
Der zweite Bereich in dem DeepMind überragende Fortschritte verzeichnen sind Videospiele. Mit Deep Learning mit Künstlichen Neuronalen Netzen (KNN) und einer angepassten Version des Q Learning, das bereits 1989 zum ersten Mal erforscht wurde (vergl. Watkins (1989)), gelang es DeepMind unterschiedliche, klassische Atari-Spiele mittels eines generalistischen Ansatzes zu spielen (vergl. Mnih u. a. (2015)).
Dabei wurde das Netz nicht auf jedes einzelne Spiel angepasst, sondern es gelang ein Netz

zu konstruieren, das auf völlig unterschiedlichen Spielen lernen konnte. Dabei erreichte es in vielen Spielen bessere Ergebnisse als menschliche Vergleichsspieler.

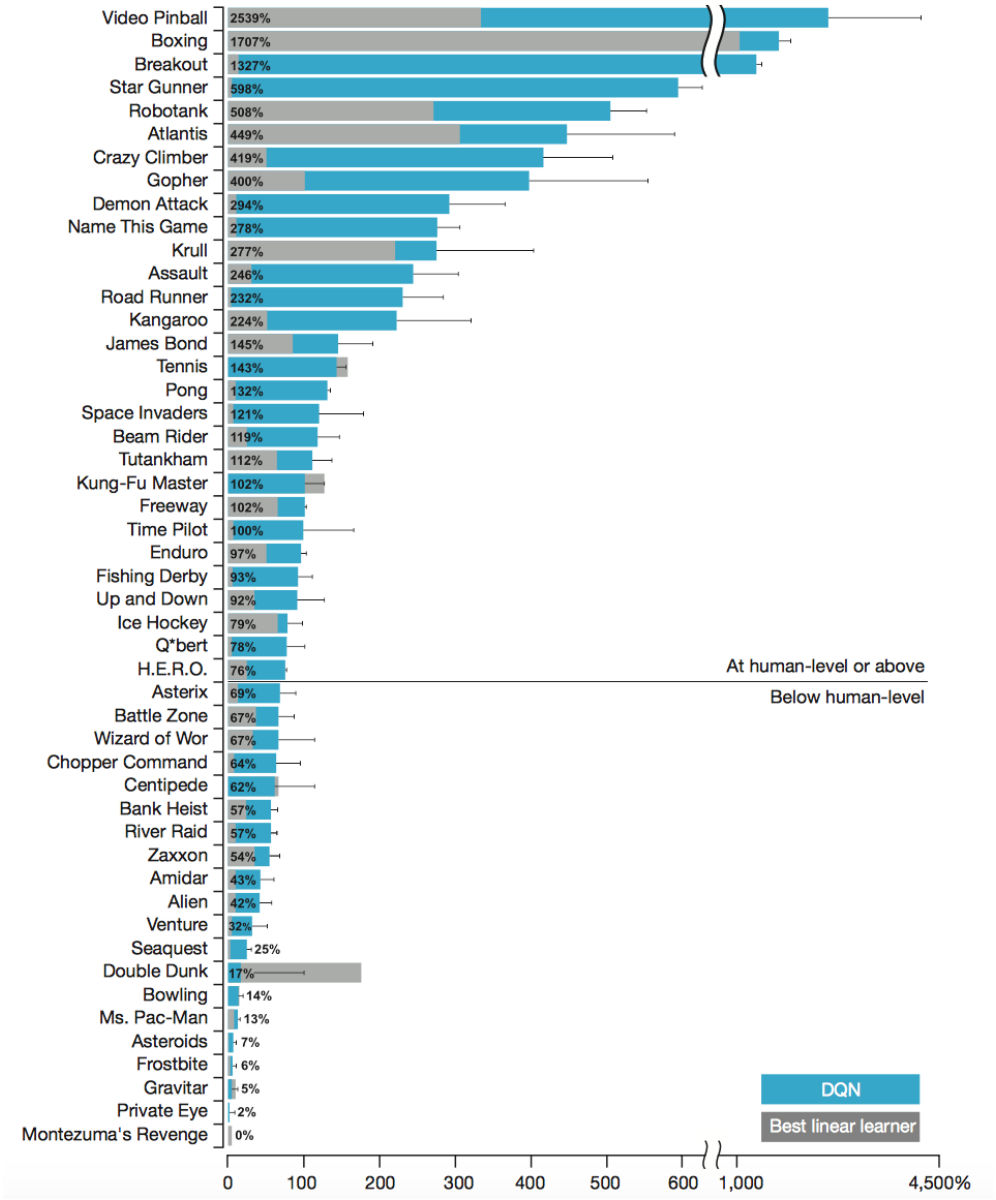

Abbildung 2.2: Vergleich des Netzes von DeepMind mit den damals besten Vergleichswerten aus der Literatur und menschlichen Vergleichswerten für eine Reihe klassischer Atari-Spiele Mnih u. a. (2015)

Um eine höhere Qualität der Lernverfahren zu erreichen setzte das Team von DeepMind so genanntes Experience Replay (Memory Replay) ein (vergl. Liu und Zou (2017)). Memory Replay ist, wie KNN, an Theorien über die Funktionsweise des menschlichen Gehirns angelehnt. Dabei werden Erfahrungen (z.B. absolvierte Spielschritte) in einer Memory-Liste gespeichert. Das Neuronale Netz lernt nun nicht mehr direkt auf den erfolgten Erfahrungen, sondern holt für jeden Lernschritt einen Batch an Erfahrungen aus der Memory und führt die in 2.5 vorgestellte Backpropagation darauf durch.
In dieser Arbeit wird ein Versuchsaufbau beschrieben, der das Q Learning mit Künstlichen Neuronalen Netzen auf ein klassisches Machine-Learning Problem (vergl. Barto u. a. (1988)) anwendet.
Dazu habe ich mir eines der sogenannten Markov-Entscheidungsspiele ausgesucht, da bei ihnen in jedem Schritt eine vollständige Information besteht (vergl. Sutton und Barto (1998)) und sie deswegen für ein erstes Eintauchen in die Materie sinnvoll sind.
Lernmethode ist das Q Learning nach Vorbild von DeepMind, das sich für Markov-Entscheidungsspiele besonders eignet, weil bei Zustandsübergängen eine vollständige Information besteht, und es somit für jeden Zustand einen „besten“ Folgezustand gibt.
Das Q Learning wird, ebenfalls an die Paper von Deepmind angelehnt, um Memory Replay und Double Q Learning erweitert, um die Lernergebnisse zu verbessern.
Folgend werden die theoretischen Konzepte vorgestellt, die für die Beantwortung der Fragestellung dieser Thesis notwendig sind.

2.1 Agent & Umgebung

Ein Reinforcement-Learning Problem lässt sich durch zwei Komponenten darstellen. Die Komponente, die Entscheidungen trifft und versucht aus ihnen zu lernen ist der „Agent“. Der Agent interagiert mit der zweiten Komponente, der „Umwelt“, die alles außerhalb des Agenten darstellt.
[1]Der Agent kennt zu jedem Zeitpunkt t den aktuellen Zustand s_t der Umgebung. Nun wählt der Agent eine Aktion a_t aus den im aktuellen Zustand möglichen Aktionen und kommuniziert diese an die Umgebung. Die Umgebung reagiert auf die gewählte Aktion und gibt, abhängig von der gewählten Aktion a_t eine Belohnung $r_t + 1$ und einen Folgezustand $s_t + 1$ zurück. Der jeweils letzte Zustand, den der Agent von der Umgebung erhalten hat, ist nun wieder der Zustand s_t.

[1]Der Absatz basiert auf Sutton und Barto (1998)

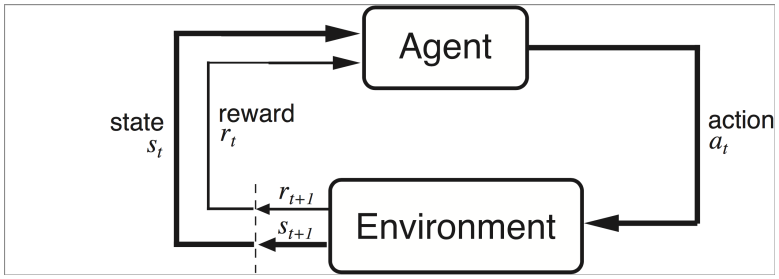

Abbildung 2.3: Darstellung der Interaktion von Agent und Umgebung nach Sutton und Barto (1998)

Wie man sieht ist die Interaktion zyklisch. In jedem Zeitschritt errechnet der Agent Wahrscheinlichkeitswerte für das Wählen einer bestimmten Aktion und bildet diese in einer Map ab. Dieses mapping nennt man die Policy π des Agenten.

2.2 Markov-Entscheidungsspiele

Die Markov-Eigenschaft ist gegeben, wenn die Wahrscheinlichkeit Pr einen Folgezustand s' von s aus zu erreichen nur von s abhängig ist, und nicht von Vorgängern vom Zustand s.

$$Pr\{s_{t+1} = s', r_{t+1} = r \quad | \quad s_t, a_t\} \tag{2.1}$$

Formale Definition der Markov-Eigenschaft nach Sutton und Barto (1998)

Das bedeutet, dass es für die Auswirkungen einer Aktion aus einem Zustand heraus unbedeutend ist durch welche Aktionen dieser Zustand erreicht wurde. Ein Beispiel dafür ist Schach. Kennt ein Agent die Positionen der Figuren auf dem Schachbrett ist das Ergebnis einer von ihm getätigten Aktion nicht vom vergangenen Spiel abhängig.

2.3 Künstliche Neuronale Netze

KNN sind Netze aus so genannten künstlichen Neuronen. Dabei werden viele einzelne Neuronen über gewichtete Kanten miteinander verbunden und mit bestimmten mathematischen Verfahren trainiert.

2.3.1 Neuronen

Die einfachste Form eines Neurons ist ein einzelner Knoten, der eine mathematische Funktion darstellt.

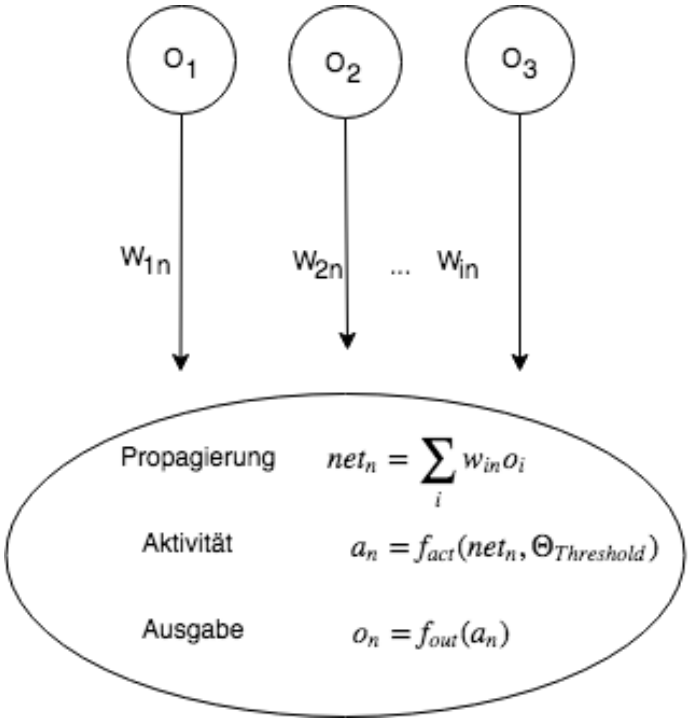

Abbildung 2.4: Darstellung eines Neurons nach Jürgen Cleve (2014) mit Propagierungsfunktion, Aktivierungsfunktion und Ausgabefunktion

Die Propagierungsfunktion $net_n = \sum_i w_{in} o_i$ nimmt die Summe über alle gewichteten Eingaben, wobei o_i der Input an Verbindung i ist und w_{in} das Gewicht an Verbindung i ist. Die Aktivierung eines Neurons ist also vom Output und der jeweiligen Gewichtung seiner Vorgänger abhängig.
Die Aktivierungsfunktion entscheidet ob das Neuron „aktiviert“ wird oder nicht. Dabei wird ein Schwellwert $\Theta_{Threshold}$ definiert, von dem die Aktivierung des Neurons abhängt.
Das einfachste Beispiel einer Aktivierungsfunktion ist die lineare Funktion $f(x) = x$. Hier ist der Output des Neurons proportional zum Input. Dies entspricht einer direkten Weiterreichung der Informationen durch das Neuron.

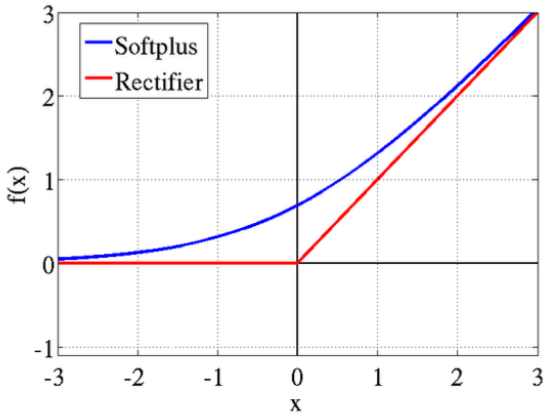

Abbildung 2.5: Verlauf einer „Rectifier“-Funktion nach Glorot u. a. (2011)

Die Aktivierungsfunktion $f(x) = max(0, x)$ hat sich in der Literatur durchgesetzt (vergl. Glorot u. a. (2011)) und wird sehr häufig für Deep Learning Tasks benutzt.
Wie man sieht werden alle negativen Input-Werte zu 0 zusammengefasst, während für alle anderen Werte der Output proportional zum Input ist. Dies verringert die Komplexität der Berechnungen im weiteren Verlauf des Lernverfahrens und beschleunigt dadurch das Training.
Die Ausgabefunktion bestimmt abhängig vom Ergebnis der Aktivierungsfunktion welcher Wert weitergereicht wird. Hier lässt sich bei Bedarf eine Normalisierung des Wertes vornehmen.

2.3.2 Mehrschichtiges Netz

Durch Zusammenschluss vieler Knoten erhält man ein Neuronales Netz. Dabei wird ein Netz unterteilt in den Input-Layer, beliebig viele Hidden-Layer und einen Ouput-Layer. Die einzelnen Layer können beliebig viele Knoten beinhalten und unterschiedlich stark miteinander vermascht sein. Die Anzahl und Dichte der Layer ist einer der vielen Parameter, die bei der Arbeit mit Künstlichen Neuronalen Netzen optimiert werden können.

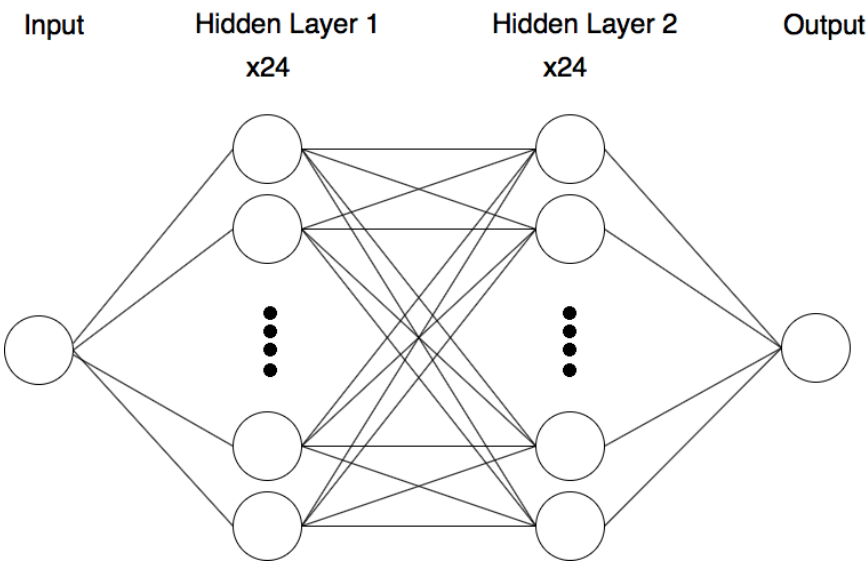

Abbildung 2.6: Beispielhafte Netzarchitektur mit Input-, Output- und zwei Hidden-Layern

Man beachte, dass hier die Verbindungen gerichtet sind. Informationen fließen immer Richtung Output-Schicht. Es gibt keine rückgeführten Zyklen.
Die Verbindungen werden durch eine Gewichtsmatrix $W_{i,j}$ dargestellt, die mit der Matrix der Eingabewerte verrechnet wird um die in Abbildung 2.4 gezeigte Funktion abzubilden. Dabei sind i und j die direkt verbundenen Knoten des Netzes und W eine Matrix mit je einem Wert für jede abgebildete Dimension. Eine Dimension bildet je ein Attribut der Entitäten des zu verarbeitenden Datensatzes.
Der Input-Layer ist immer von den zu verarbeitenden Daten abhängig. So macht es bei einem Spiel Sinn den Input-Layer an die Form der Zustände anzupassen. Wird beispielsweise der Spiel-Zustand durch einen Vektor von 4 Werten (z.B. Koordinaten) dargestellt, macht es Sinn 4 Eingabeneuronen zu verwenden, die jeweils eine der Zahlen des Eingabevektors erhalten und verarbeiten.
Es gibt mittlerweile einige verschiedene Methoden um eine optimale Anzahl und Größe der Hidden Layer zu ermitteln. Neben dem Trial and Error Verfahren haben sich unter anderem verschiedene Heuristiken, Pruning und die Brute-Force-Suche in der Literatur ergeben (vergl. Stathakis (2009)).
Die Form der Ausgabeschicht hängt von der darzustellenden Umgebung ab. Grundsätzlich besitzt ein Künstliches Neuronales Netz für jede abzubildende Kategorie einen eigenen Knoten in der Ausgabeschicht. Besteht die Aufgabe des Netzes beispielsweise darin die Zahlen von

0-9 zu erkennen, ergibt es Sinn eine Ausgabeschicht mit 10 Knoten zu wählen. Wird nun eine Eingabe durch das Netz gereicht, wird in der Ausgabeschicht das entsprechende Neuron aktiv, bzw. am stärksten aktiv, welches der „erkannten“ Zahl entspricht.

2.4 Kosten

Das Ziel eines Künstlichen Neuronalen Netzes ist in den meisten Fällen eine möglichst genaue Vorhersage für unbekannte Daten zu tätigen. Für das Lernverfahren braucht es dafür eine Objektivierung von „richtigen“ und „falschen“ Vorhersagen. Die Differenz zwischen einer getätigten Vorhersage und dem gewünschten Ergebnis nennt man „Kosten“, die Funktion „Kostenfunktion“.
Im Beispiel von Mnih u. a. (2015) wurde als Kostenfunktion der Mean Squared Error benutzt.

$$MSE = \frac{1}{m} \sum_{i} (\hat{y} - y)^2 \tag{2.2}$$

Kostenfunktion Mean-Squared-Error nach Goodfellow u. a. (2016)

Dabei wird der vom zu trainierenden Modell vorhergesagte Wert y vom Target $\hat{y}$ abgezogen und danach quadriert. Der Target $\hat{y}$ ist der Soll-Wert, der sich klassicherweise aus den Trainingsdaten ergibt. Durch das Quadrieren wird verhindert, dass sich bei der Bildung der Summe positive und negative Kosten annulieren.
Der Durchschnitt der errechneten Werte bildet die Kosten, die durch die Backpropagation durchs Netz propagiert werden.

2.5 Backpropagation & Gradientenabstiegsverfahren

Das Lernverfahren von Künstlichen Neuronalen Netzen basiert auf den zwei Grundkonzepten Backpropagation und Gradientenabstiegsverfahren.
Backpropagation ist das Verfahren um die errechneten Kosten vom Ende des Künstlichen Neuronalen Netzes nach vorne zu propagieren um eine Errechnung des Gradienten der Kosten zu ermöglichen. Dabei wird unter Verwendung der „Chain Rule“ (Kettenregel) rekursiv für jeden Knoten des Netzes ein Gradient errechnet (vergl. Goodfellow u. a. (2016)).
Beim Gradientenabstiegsverfahren wird nun die Information über den Gradienten jedes einzel-

nen Knoten benutzt um die Gewichte der Verbindungen zwischen den Knoten so anzupassen, dass bei einem erneuten Durchlauf die Kosten am Ende des Netzes verringert werden. Dadurch übernimmt jeder Knoten einen Teil der Korrekturleistung um den Fehler zu minimieren, und reicht den Rest der Kosten weiter nach vorne.
Im Machine Learning wird häufig das „Stochastic Gradient Descent"-Verfahren eingesetzt, bei dem nicht jeder Datensatz zur Anpassung der Gewichte benutzt wird, sondern über Mini-Batches versucht wird das Ergebnis zu approximieren. Es ist gezeigt, dass so auch bei sehr großen Datensätzen ausreichende Ergebnisse erzielt werden können, obwohl nur ein verhältnismäßig kleiner Teil des Datensatzes für das Gradientenabstiegsverfahren benutzt werden muss (vergl. Goodfellow u. a. (2016)).
Ziel dieser beiden Verfahren ist die Kosten des Netzes für alle Eingaben zu minimieren, also für jede Eingabe eine möglichst „richtige" Ausgabe erzeugen zu können.

2.6 Q Learning

Das klassische Q-Learning basiert auf einer Funktion, die Belohnungswerte für Zustände approximiert.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \tag{2.3}$$

one-step Q-Learning nach Sutton und Barto (1998)

Q ist dabei die Approximationsfunktion, die Zustand und Aktion als Eingabe bekommt. Das Symbol $\leftarrow$ beschreibt ein Update dieser Q-Funktion. Nach jedem Schritt, den ein Agent in der Umgebung ausführt, wird der alte Funktionswert um einen gewissen Betrag erhöht, bzw. verringert. Dabei ist r_{t+1} die Belohnung für die zuletzt ausgeführte Aktion, auf die die erwartete Belohnung des nächsten Schrittes $\max_a Q(s_{t+1}, a)$ aufaddiert wird.
Der Betrag der erwartete Belohnung wird mit dem Discount-Faktor γ multipliziert, für den üblicherweise gilt: $0 < \gamma < 1$. Über γ kann gesteuert werden wie stark die erwartete, zukünftige Belohnung in die Bewertung eines Zustand-Aktion Paares einfließt, wie weit der Algorithmus also in die Zukunft blickt. Ein γ Wert von 0 entspricht einem geringen look-ahead Wert, ein Wert von 1 dementsprechend einem Algorithmus, der auch sehr weit entfernte zukünftige Belohnungen in Betracht zieht (vergl. Sutton und Barto (1998)).
Von $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$ abgezogen wird die ursprüngliche Bewertung des Zustand-Aktion Paares. Der dabei gebildete Wert entspricht einem Delta zwischen der alten und der neuen Bewertung. Dieser Wert wird nun mit der Lernrate α multipliziert. Für die Lernrate

gilt üblicherweise auch: $0 < \alpha < 1$. Über α wird gesteuert mit welcher Intensität der alte Funktionswert über die erwartete Belohnung aufgrund der neuen Erkentnisse aktualisiert wird. Das ist analog zur Schrittweite beim Bergsteigeralgorithmus (vergl. Russell und Norvig (2016)). Wählt man α sehr groß, überspringt man im Zweifel die lokalen Minima, wählt man es sehr klein, ist es sehr wahrscheinlich, dass man nicht das globale Minimum findet.
Für den klassischen Q-Learning Algorithmus ist bewiesen, dass er zu einer optimalen Lösung konvergiert (vergl. Melo (2001)).

2.7 Q Learning mit KNN

2.7.1 Target

Das von Mnih u. a. (2015) gezeigte Q Learning für KNN baut auf dem in Kapitel 2.6 beschriebenen, klassischen Q Learning auf. Dabei approximiert das Neuronale Netz die Q-Funktion $Q(s, a; \Theta)$, versucht also eine Gewichtsbelegung zu finden, die für eingegebene Zustände die erwartete Belohnung für jede Aktion vorhersagt. Dabei steht Θ für die Gewichte des Netzes. Ausgehend vom klassischen Q Learning wird das Update der Q-Funktion durch das in Kapitel 2.5 beschriebene Gradientenabstiegsverfahren realisiert. Dies benötigt die Kosten einer Eingabe, die beispielweise über die in Kapitel 2.4 vorgestellte Mean Squared Error Kostenfunktion ermittelt werden können.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

Abbildung 2.7: Mean Squared Error für Q Learning mit Künstlichen Neuronalen Netzen nach Mnih u. a. (2015)

In Abbildung 2.7 entspricht $r + \gamma \max_a Q(s', a'; \Theta_i^-)$ dem Zielwert und $Q(s, a; \Theta_i)$ dem aktuell vom Netz vorhergesagten Wert für das Zustand-Aktion Paar. Das s' steht für den Folgezustand von s und a' für den Folgezustand von a. Die unterschiedliche Benennung von Θ basiert auf einer in der Literatur getätigten Erweiterung, dem Double Q Learning.

2.7.2 Double Q Learning

Beim klassischen Q Learning ist der vorhergesagte Wert direkt abhängig von der aktuellen Gewichtsbelegung des Policy-Netzwerks. Wird die Bewertung $Q(s_t, a_t)$ erhöht durch das

durchführen eines Update, erhöht dies häufig auch die Bewertung aller Folgezustände und somit indirekt den Zielwert für folgende Updates. Dies führt dazu, dass die Annäherung an die optimale Gewichtsbelegung oszilliert.
Um dies zu verhindern benutzt man zwei unterschiedliche Netze, von denen eines zur Auswahl der nächsten Aktion benutzt wird (Policy-Netzwerk Θ_i) und das andere zur Berechnung des Ziel-Wertes (Target-Netzwerk Θ_i^-). Alle N Schritte werden die Gewichte des Policy-Netzwerkes benutzt und in das Target-Netzwerk kopiert. Das führt zu einem stabileren Lernverfahren und insgesamt besseren Ergebnissen (vergl. Van Hasselt u. a. (2016)). Der Zielwert $r + \gamma \max_a Q(s', a'; \Theta_i^-)$ basiert auf der für die betrachtete Aktion tatsächlich zurückgegebenen Belohnung und der nach aktuellem „Wissen“ des Netzes erwarteten, zukünftigen Belohnung ausgehend vom neuen Zustand.
Der mit den jeweils aktuellen Gewichtsparametern des Netzes vorhergesagte Wert $Q(s, a; \Theta_i)$ wird vom Zielwert abgezogen, da es sich beim Zielwert um eine womöglich bessere Schätzung handelt (weil Informationen über die tatsächliche Belohnung vorliegen). Das Delta aus beiden Werten entspricht den Kosten für den betrachteten Input, die dann aus den in Kapitel 2.4 genannten Gründen quadriert wird.
Nach Tsitsiklis und Roy (1997) ist bekannt, dass die Ergebnisse einer Funktionsapproximation z.B. durch ein Künstliches Neuronales Netz instabil sind, bzw. teilweise sogar von einer optimalen Lösung divergieren.
Eine Mögliche Lösung dem entgegenzuwirken ist das sogenannte Memory Replay.

2.7.3 Memory Replay

Beim einfachen Q Learning (Kapitel 2.7) wird die Backpropagation nach jeder ausgeführten Aktion durchgeführt. Die Updates des Modells sind daher stark an die Reihenfolge der Erfahrungen gekoppelt und jede Erfahrung wird nur genau einmal verwertet und dann "vergessen". Dies ist vor allem bei wichtigen Erfahrungen, die selten vorkommen ein Problem (vergl. Lin (1992)).
Eine Möglichkeit dem entgegenzuwirken ist laut Lin (1992) das Memory Replay. Memory Replay hat unter anderem bei AlphaGo (vergl. Silver u. a. (2017)), und bei der Lösung von Atari-Klassikern mittels Reinforcement Learning (vergl. Mnih u. a. (2013)) einen entscheidenden Vorteil gebracht.
Dabei wird das Gradientenabstiegsverfahren nicht auf den Kosten des aktuell verarbeiteten Input ausgeführt, sondern auf einem Minibatch von Erfahrungen (s_t, a_t, r_t, s_{t+1}) (in der Abbildung 2.7.4 wird statt mit Zuständen s mit Sequenzen ϕ gearbeitet, da Mnih u. a. (2015) mit Zustandssequenzen arbeiten). Dazu wird erst eine Aktion a getätigt und deren Beloh-

nung r und Folgezustand s_{t+1} im Spiel beobachtet. Diese werden werden zusammen mit dem Ausgangszustand s in einer FIFO-Queue gespeichert. An dieser Stelle wird nun kein Gradientenabstiegsverfahren mit der aktuellsten Erfahrung angestoßen. Stattdessen wird das Gradientenabstiegsverfahren auf einem Batch aus zufällig aus der Queue ausgewählten Erfahrungen durchgeführt.

2.7.4 Algorithmus

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Abbildung 2.8: Kompletter Algorithmus zum Q Learning mit Künstlichen Neuronalen Netzen mit Double Q Learning und Memory Replay nach Mnih u. a. (2015)

Der Algorithmus in Abbildung 2.8 baut auf den zuvor beschriebenen Grundlagen auf. Zuerst wird eine Replay Memory D mit N Plätzen initiiert. Danach wird ein Neuronales Netz Q mit den Gewichten Θ und eine Kopie davon mit den Gewichten Θ_i^- initiiert. Das Lernverfahren wird für M Episoden durchlaufen. Eine Episode ist gleichbedeutend mit einer Partie. In jeder

Episode wird die folgende Schleife für t Schritte absolviert, jeweils bis zum Spielende.
Über eine ϵ-Greedy Policy wird die nächste Aktion vom Neuonalen Netz Q ausgewählt. Im besten Fall fängt ϵ bei 1 an und wird mit der Zeit kleiner. Am Anfang eines Lernverfahrens ist der Zustandsraum unbekannt und durch ein hohes Maß an Exploration werden viele sehr unterschiedliche Zustände durchlaufen (vergl. Kaelbling u. a. (1996)). Mit fortgeschrittener Lerndauer ergibt es Sinn auf Zustände zuzugreifen, die bereits eine Bewertung erhalten haben, um den Agenten zum Ziel zu führen.
Die gewählte Aktion wird vom Agenten ausgeführt. Wie in Kapitel 2.1 erläutert gibt die Umgebung nun den Folgezustand und eine Belohnung zurück.
Das Ergebnis eines Schrittes ist eine Erfahrung (s_t, a_t, r_t, s_{t+1}), die in der Replay-Memory D gespeichert wird. Aus dieser Replay-Memory wird nun ein Minibatch zufällig ausgewählter Erfahrungen genommen und der Reihe nach zum Training der Netzgewichte benutzt. Dazu wird, wie in Kapitel 2.7 erläutert der Zielwert und der tatsächliche Wert gebildet, daraus die Kosten ermitteln und diese als Grundlage eines Gradientenabstiegsverfahrens benutzt.

2.8 CartPole

Bei dem Spiel CartPole handelt es sich um einen klassiker der Machine-Learning Forschung, unter anderem beschrieben von Sutton und Barto (1998).

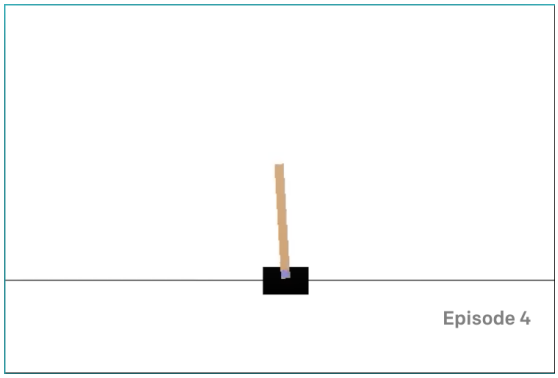

Abbildung 2.9: Spiel Cartpole (vergl. OpenAi (2018a))

„Über ein frei schwingendes Gelenk ist eine Stange an einem Wagen angebracht, der über eine reibungslose Schiene fährt.
Das System wird kontrolliert, indem eine Kraft von +1 oder -1 auf den Wagen angewandt wird. Das Pendel ist zu Beginn des Spiels aufrecht. Das Ziel des Spiels ist das Pendel so lange wie

möglich am umfallen zu hindern.
Die Episode endet, wenn die Stange um mehr als 15 Grad von der Vertikalachse abweicht oder der Wagen sich mehr als 2,4 Einheiten von der Mitte des Spielfeldes entfernt. Für jeden Zeitschritt, den das Spiel läuft, erhält der Agent eine Belohnung von „+1 “ (frei übersetzt nach OpenAi (2018a), auf Grundlage von Sutton und Barto (1998)).

2.8.1 Zustands- & Aktionsraum

Der aktuelle Zustand des Spiels wird in einem 4-Vektor dargestellt. Die Nummern in der folgenden Abbildung beziehen sich auf die Position im Zustands-Vektor.

Tabelle 2.1: Zustandsraum nach OpenAi (2018a)

#	Parameter	Min.	Max.	Beschreibung
0	Cart Position	-4.8	4.8	Ein Wert von 0 beschreibt die Mitte der Spielfläche, negative Werte sind links von der Mitte (und umgekehrt)
1	Cart Geschwindigkeit	$-\infty$	∞	Bei einer negativen Geschwindigkeit fährt das Cart in Richtung des linken Bildschirmrands, bei einer positiven Geschwindigkeit zum rechten Bildschirmrand. Die Geschwindigkeit wird in einer von OpenAi gewählten, relativen Einheit angegeben.
2	Winkel d. Stange	-24°	24°	In Grad Celsius im Verhältnis zur Vertikalen
3	Geschwindigkeit d. Stange an Spitze	$-\infty$	∞	Ebenfalls in relativer Einheit, negative Geschwindigkeitswerte werden durch einen Fall in Richtung des linken Bildschirmrands erzeugt, positive andersherum

Dadurch, dass zwei der möglichen Parameter Werte von $-\infty$ bis ∞ annehmen können, ist der Zustandsraum theoretisch unendlich. Ein Spiel mit unendlichem Zustandsraum ist mit einem Künstlichen Neuronalen Netz besser zu lösen als mit einer Q-Tabelle. Allerdings muss an dieser Stelle erwähnt werden, dass durch eine Zusammenfassung ähnlicher Zustände der Zustandsraum erheblich verringert werden könnte. Darauf wurde in dieser Thesis verzichtet

um zu verhindern, dass Domänenwissen in die Lösung der Aufgabe fließt.
Der Aktionsraum besteht aus genau zwei möglichen Aktionen. Ein Schritt im Spiel mit dem Input „0“ bewegt das Cart nach links, „1“ bewegt das Cart nach rechts. Die Kraft, die bei den jeweiligen Aktionen auf das Cart ausgeübt wird ist dabei als abhängig vom Winkel des Stabes implementiert. Die Implementation ist nachzuverfolgen in Sutton (o.D)

Tabelle 2.2: Aktionsraum nach OpenAi (2018a)

#	**Aktion**
0	Übt Kraft von 10N von rechts nach links auf Cart aus
1	Übt Kraft von 10N von links nach rechts auf Cart aus

Grundsätzlich sieht man an dem Zustands- und Aktionsraum, dass es sich bei dem Spiel um eine stark von der Realität abstrahierte Simulation handelt. Für diese Thesis ist es jedoch vor allem aus folgenden Gründen geeignet:
1) Bei dem Spiel handelt es sich um ein Markov-Decision-Problem, bei dem die Markov-Annahme aus Kapitel 2.2 gilt.
2) In jedem Zustand lässt sich eine Entscheidung über die beste, nächste Aktion treffen, ohne die vorherigen Zustände zu kennen. Es handelt sich also um ein Spiel mit perfekter Information.
3) Der Zustandsraum ist groß genug, als dass sich eine Modellierung mit Künstlichen Neuronalen Netzen lohnt.

2.8.2 Fragestellung

Als Referenzwert für die Eignung Künstlicher Neuronaler Netze für die zuvor definierte Problemstellung bietet sich die von OpenAi (2018b) aufgestellte Definition eines „gelösten Spiels“ an. Dabei gilt der Vorgänger (Cartpole v0) der in dieser Arbeit genutzen Implementation (Cartpole v1) als gelöst, wenn er in 100 konsekutiven Episoden einen durchschnittlichen Reward von >= 195 erlangt. In der Version v1 wird das Spiel erst nach 500 Schritten unterbrochen, wenn vorher kein Spielabbruch durch die in Kapitel 2.8 beschriebenen Abbruchbedingungen zustande kommt. Dies macht es einfacher eine durchschnittliche, gesammelte Belohnung von >= 195 zu erreichen, was im Rahmen dieser Thesis akzeptiert wird.

> CartPole-v0 defines „solving“ as getting average reward of 195.0 over 100 consecutive trials.... (vergl. OpenAi (2018b))

Der Anspruch dieser Thesis ist die Lösung des Cartpole Spiels nach der zuvor genannten Definition.
Gelingt es das Spiel zu lösen, wird versucht eine gute Lösung zu finden, die sich den Referenzergebnissen im OpenAI Leaderboard (vergl. OpenAI (2018c)) annähert. Dabei handelt es sich um eine Auswahl von der Open AI Community ausgewählter Lösungen für Cartpole, mit teilweise sehr guten Lösungen. Der Anspruch ist also nicht die Tabelle anzuführen, sondern eine näherungsweise Lösung zu entwerfen.

User	Episodes before solve
Tom	9
Udacity DRLND Team	13
TeaPearce, nanastassacos	16
MisterTea, econti	24
yingzwang	32
SurenderHarsha	40
n1try	85
khev	96
ceteke	99
manikanta	100
JamesUnicomb	145
Harshit Singh Lodha	265
mbalunovic	306
ruippeixotog	933

Abbildung 2.10: Sammlung konkurrenzfähiger Lösungen für das Spiel Cartpole von OpenAI
OpenAI (2018c)

Eine lineare Lösung angereichert mit Expertenwissen könnte das Spiel noch vor dem ersten Durchlauf lösen. Das ist für diese Thesis unerheblich, da hier erforscht werden soll inwieweit sich Neuronale Netze als non-lineares Verfahren zur Lösung eines Problems ohne die Verwendung von Domänenwissen eignen.

3 Architektur

Um die Frage nach der Eignung Künstlicher Neuronaler Netze für die Lösung des Cartpole Spiels zu beantworten, wird ein Versuchsaufbau benötigt. Dazu wird im folgenden eine Architektur definiert, die die Grundlage einer Experimentalumgebung liefert, mit der im folgenden Messungen durchgeführt werden.
Dieses Kapitel widmet sich der für spätere Messungen notwendigen Architektur und erklärt dafür getroffene Entscheidungen. Zuerst wird die Modellsicht beschrieben, dann die Klassen gefolgt von den wichtigsten Prozessen. Zuletzt werden gewählte Tools erläutert.
Die Software soll ein Lernverfahren über eine Menge an Episoden durchführen können, bei dem auf einer Spielumgebung Aktionen ausgeführt werden, aus denen ein Agent lernt. Der Agent muss nach dem Q-Learning Algorithmus handeln und über ein Modell verfügen, das ein Künstliches Neuronales Netz darstellt. Für das Memory Replay braucht es eine Memory, in der gesammelte Erfahrungen gespeichert werden.

3.1 Komponenten

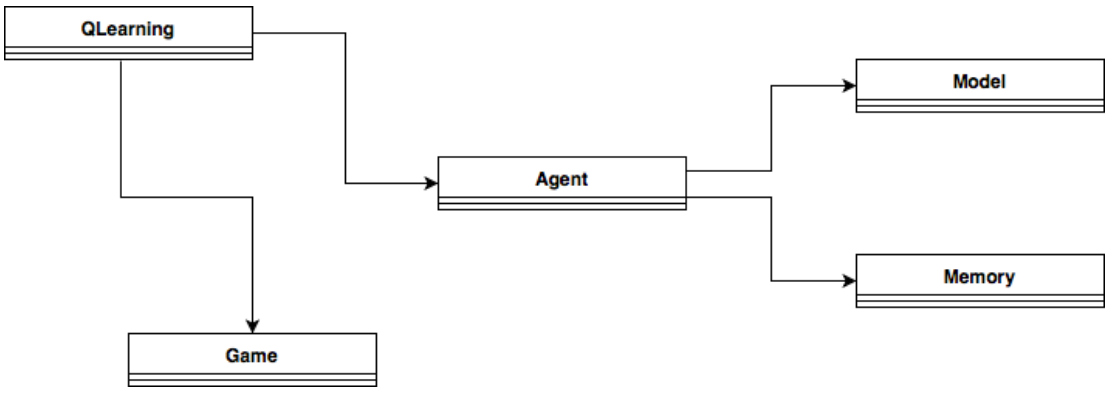

Abbildung 3.1: Übersicht der wichtigsten Komponenten

Die Komponenten übernehmen nach dem von Martin (2009) beschriebenen Single Responsibility Principle immer genau eine Aufgabe.
So hat die QLearning Komponente die Aufgabe das Lernverfahren abzubilden. Dafür fragt sie die nächste auszuführende Aktion von der Agenten-Komponente an und übermittelt diese an die Game-Komponente. Diese hat nur die Aufgabe Aktionen auf die Spielumgebung anzuwenden und die resultierenden Zustände und Belohnungen zu liefern. Sie braucht keine Kenntnis über die restlichen Komponenten.
Für eine klare Trennung der Zuständigkeiten (Separation of Concerns) ist die Interaktion mit dem CartPole-Framework in die Game-Komponente ausgelagert. Soll ein anderes Spiel mit einer anderen Schnittstelle in die Software eingebunden werden, muss so nur die Adapter-Klasse angepasst werden.
Obwohl OpenAi die Belohnung für einzelne Schritte vorgibt, lässt sich in der Game-Komponente bei Bedarf die Belohnung einzelner Zustände anpassen um bessere Lernerfolge herbeizuführen.
Der Agent wählt Aktionen anhand seines trainierten Modells aus. Dafür kommuniziert der Agent mit der Model-Komponente und übergibt ihr Erfahrungen, die von dieser für die Backpropagation benutzt werden. Die Erfahrungen speichert der Agent nicht selbst, sondern überträgt sie an die Memory-Komponente, deren einzige Aufgabe darin besteht Erfahrungen zu halten und zufällige Batches von Erfahrungen zu liefern.

3.2 Klassen

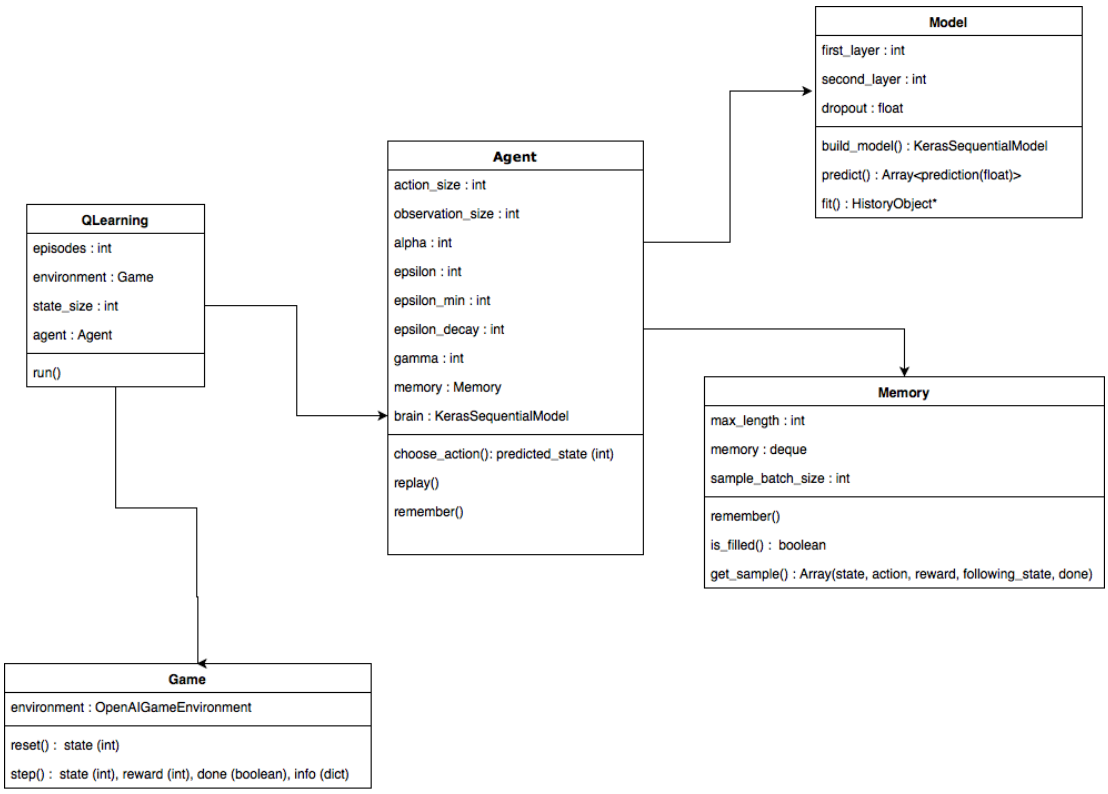

Abbildung 3.2: Klassendiagramm der wichtigsten Klassen

3.2.1 Game

Die Game-Klasse ist ein Adapter für die Spielimplementation von OpenAI. Sie hält eine Instanz der Open-AI-Spielumgebung. Die Methode reset() setzt das Spiel in einen Anfangszustand zurück. Dies geschieht nach jeder Lern-Episode, also nach jedem Sieg, jeder Niederlage im Spiel.
Die wichtigste Methode der Klasse ist step(). Diese Methode nimmt die vom Agenten gewählte Aktion als Parameter und führt diese in der Spielumgebung aus. Als Rückgabewert liefert sie den neuen Zustand (state) der Spielumgebung, eine Belohnung und einen Wahrheitswert, der angibt ob das Spiel beendet wurde (Sieg/Niederlage) oder nicht.

3.2.2 QLearning

Die QLearning Klasse instanziiert die übrigen Klassen und steuert den Lernvorgang. Sie hält jeweils eine Instanz der Spielumgebung (Game) und des Agenten (Agent), sowie die Information über die Anzahl an Episoden und die Größe des Zustandsraumes.
Die einzige Methode ist run(). Für die festgelegte Anzahl an Durchläufen (Episoden) wird per Schleife je ein Lernvorgang durchgeführt. Ein Lernvorgang beginnt mit der Zurücksetzung der Spielumgebung (reset()) und endet mit Beendigung des Spiels (done = true). Der Agent wird zwischen den Episoden nicht zurückgesetzt, er behält seine Lernfortschritte bei.
Die QLearning-Klasse bestimmt den Zeitpunkt, zu dem der Agent seine Methoden duchführt. In jedem Schritt des Spieldurchlaufs wird über die choose_action()-Methode des Agenten die nächste zu vollführende Aktion ermittelt. Die vom Agenten gewählte Aktion wird über die step()-Methode der Game-Klasse auf die Spielumgebung angewandt. Die von der Spielumgebung zurückgegebene Erfahrung wird über die remember()-Methode des Agenten an diesen zum Speichern weitergegeben. Zuletzt wird über die replay()-Methode des Agenten ein Memory-Replay angestoßen (vergl. Kapitel 2.7.3)

3.2.3 Agent

Der Agent hält Informationen über die gewählte Lernrate α (learning_rate), die Explorationsrate ϵ(epsilon) und den Discount-Faktor γ (gamma). Außerdem instanziiert der Agent die Klassen Model und Memory.
Über die Methode choose_action() wählt der Agent Aktionen aus. Dabei ergibt sich die Wahrscheinlichkeit eine zufällige Aktion zu wählen aus dem aktuellen Wert von Epsilon. Wird keine zufällige Aktion gewählt wird Exploited, d.h. die Aktion gewählt, die nach aktuellem Modell vom Neuronalen Netz die beste Bewertung erhält. Die so gewählte Aktion wird zurückgegeben. Hiernach wird die Explorationsrate (epsilon) verringert (vergl. Kapitel 2.8).
Über die Methode remember() speichert der Agent die ihm von der QLearning-Klasse übergebene Erfahrungen (Folgen seiner Aktionen) über die gleichnamige Methode (remember()) der Memory-Klasse.
Nach jeder von der Spielumgebung erlangten Erfahrung, wird durch die replay()-Methode des Agenten ein Mini-Batch bereits erlangter Erfahrungen für das Memory Replay von der Memory-Klasse über get_sample() angefordert. Für jede Erfahrung des Samples wird der Target-Wert berechnet (vergl. Kapitel2.7) und über die fit()-Methode der Model-Klasse eine Backpropagation gestartet.

3.2.4 Memory

Die Memory-Klasse hält eine FIFO-Queue in der Erfahrungen gespeichert werden, eine Information über die gewünschte Maximallänge der Queue und die Größe des Batches, die von get_sample() zurückgegeben werden soll.
Neue Erfahrungen werden über die Methode remember() eingefügt. Hat die Queue ihre Maximallänge erreicht, wird die älteste Erfahrung aus der Queue entfernt und eine neue Erfahrung eingefügt.
Über die Methode get_sample() gibt die Klasse einen Batch aus zufällig ausgewählten Erfahrungen zurück.

3.2.5 Model

Das Model hält das Keras-Model (vergl. Kapitel 3.5) und Informationen über die Größe und Anzahl der Hidden Layer.
Die Methode _build_model() baut ein Keras-Model mit den gewünschten Hyperparametern zusammen. Hier wird die Reihenfolge der Schichten, mit ihrer jeweiligen Größe, Aktivierungsfunktion, Input-Dimensionen und der Optimierungs- sowie Loss-Funktion für die Backpropagation festgelegt.
Die Methode predict() bekommt als Eingabeparameter einen Zustand, reicht diesen Input durch das KNN und gibt den vom KNN errechneten Output zurück. Der Output ist ein Array von float-Werten, die für jede Aktion eine Schätzung über ihre Qualität darstellen.
Durch die Methode fit() wird das Model auf einer Kombination aus Zustand und Target 2.7 trainiert.

3.3 Prozesse

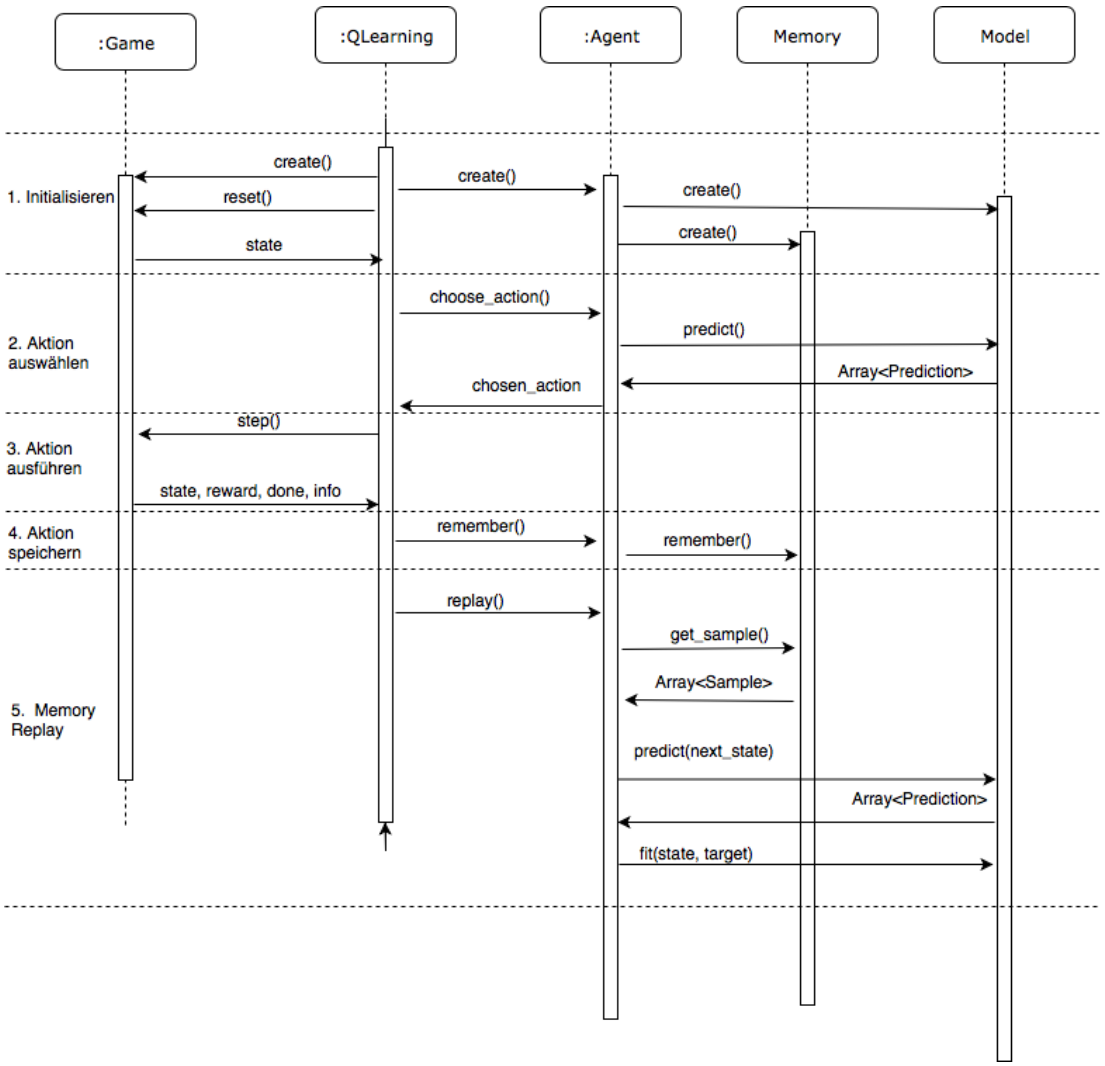

Abbildung 3.3: Sequenzdiagramm der wichtigsten Abläufe

Das Diagramm zeigt die Interaktionen zwischen den Klassen im Verlauf eines Lernvorgangs. Der Prozess lässt sich in die Fünf unten beschriebenen Phasen aufteilen.

Tabelle 3.1: Prozessdiagram der Architektur

Phasen eines Lernvorgangs

1. Initialisieren	Zu Beginn instanziiert die QLearning-Klasse die Klassen Game und Agent und setzt die Spielumgebung auf einen Anfangszustand zurück, der von Game definiert wird. Der Agent instanziiert seine Memory und das Model.
2. Aktion auswählen	Für die Dauer eines Spieles fragt die QLearning-Klasse zu Beginn jedes Lernschrittes die nächste auszuführende Aktion vom Agenten an. Dieser konsultiert dafür entweder sein Modell und gibt die aktuell als beste angesehene Aktion zurück oder gibt eine zufällige Aktion zurück. Dies ist abhängig vom aktuellen ϵ Wert.
3. Aktion ausführen	Die vom Agenten ausgewählte Aktion wird an die Spielumgebung kommuniziert, die diese Aktion auf dem Spiel ausführt. Game liefert den neuen Zustand, die erhaltene Belohnung und eine Information darüber, ob das Spiel beendet ist, zurück.
4. Aktion speichern	Die neue Erfahrung wird von der QLearning-Klasse an den Agenten weitergegeben, der diese in seiner Memory speichert.
5. Memory Replay	Nach jeder Aktion wird ein Memory-Replay von der QLearning-Klasse angestoßen. Der Agent holt sich dazu ein Batch von zuffälligen Erfahrungen aus seiner Memory. Für jede Erfahrung wird wie in Kapitel 2.7 erklärt ein Target-Wert errechnet.

3.4 Netzarchitektur

Um den in Kapitel 2.8.2 gestellten Anforderungen zu genügen wird die Anzahl und Größe der Hidden-Layer des Modells in Kapitel 4 empirisch ermittelt. Dazu werden unterschiedliche Modell-Architekturen miteinander verglichen um eine der Problemstellung entsprechende Architektur zu finden.

Die Eingabeschicht wird auf einen Knoten festgesetzt, der ein Array von 4 Werten als Input erhält. Dies entspricht der Form der Zustände des Spiels (vergl. Kapitel 2.8).
Die Ausgabeschicht wird auf zwei Knoten festgesetzt, wobei jeder Knoten einer möglichen Aktion des Aktionsraums (vergl. Kapitel 2.2) entspricht. Als Aktivierungsfunktion wird im Input-Layer und in den Hidden-Layers die in Kapitel 2.5 vorgestellte Rectifier-Funktion benutzt, in der Ausgabeschicht eine lineare Aktivierungsfunktion.

3.5 Tensorflow & Keras

Die Implementation der Modelle wurde mit Keras vorgenommen. Dabei handelt es sich um eine „high-level neural networks API“ (vergl. Chollet u. a. (2015)) zur einfachen Definition Künstlicher Neuronaler Netze und der dazugehörigen Hyperparameter in der Skriptsprache Python (vergl. Rossum (1995)), die auf der von Google entwickelten Library Tensorflow aufbaut. Tensorflow ist aktuell ein quasi Standard in der Arbeit mit Künstlichen Neuronalen Netzen und abstrahiert die Berechnung einzelner Neuronen und Verbindungen und erlaubt eine effiziente Berechnung von Backpropagation und Gradientenabstiegsverfahren.
Keras ermöglicht darauf aufbauend ein schnelles Prototyping verschiedenster Netzarchitekturen und hat die für diese Thesis benötigten Messungen ermöglicht.
Mit den gewählten Frameworks ließ sich die Architektur ohne nennenswerte Probleme implementieren.

4 Evaluation

4.1 Ergebnisse

Um zu prüfen ob sich die zuvor genannte Architektur eignet die in Kapitel 2.8.2 aufgestellten Anforderungen zu erfüllen wurden Messungen vorgenommen. Für die Messungen dieser Thesis wurden einige Hyperparameter im Voraus festgelegt:

Tabelle 4.1: Verwendete Hyperparameter

Lernrate	0.001	α
Discountrate	0.99	γ
Explorationsrate zu Beginn	1.0	ϵ_{max}
Explorationsrate minimum	0.01	ϵ_{min}
Experience Memory Größe	2000	
Batch-Größe der Memory Sample pro Replay	32	

Die Discountrate γ wurde bewusst hoch gewählt, um in der Zukunft liegende Belohnungen stark in die Bewertungen potentieller Aktionen einfließen zu lassen. Die Lernrate α ist bewusst niedrig gewählt, um schnell zumindest lokale Minima zu finden. Auf ein Feintuning wurde in dieser Thesis verzichtet, der Wert wurde auf 0.001 gesetzt um das Modell schnell zumindest lokale Minima finden zu lassen um die Anforderung der Lösung in möglichst wenig Schritten zu erfüllen.
Die ersten Messungen wurden durchgeführt um eine Einschätzung über sinnvolle Netzarchitekturen zu bekommen. Diese wurden initial ohne Memory Replay (vergl. Kapitel 2.7.3) und ohne die Erweiterung des Double Q Learning (vergl. Kapitel 2.7.2) gemacht. Die Modelle wurden mit unterschiedlichen Modellarchitekturen, mit ϵ_{decay} = 0.995 über 1000 Episoden trainiert.
Als Metriken wurden für jede Episode die gesammelte Belohnung „reward “ und der aktuelle Epsilonwert „epsilon “, sowie die durchschnittliche, gesammelte Belohnung der letzten 100 Episoden „average_reward “ aufgenommen. Bei der durchschnittlichen, gesammelten Belohnung

ist zu beachten, dass ein Modell, das nach X Episoden einen „average_reward “ von >=195 vorweisen kann, sowohl im Leaderboard OpenAI (2018c), als auch in dieser Thesis als „gelöst nach X-100 Schritten“ gilt.

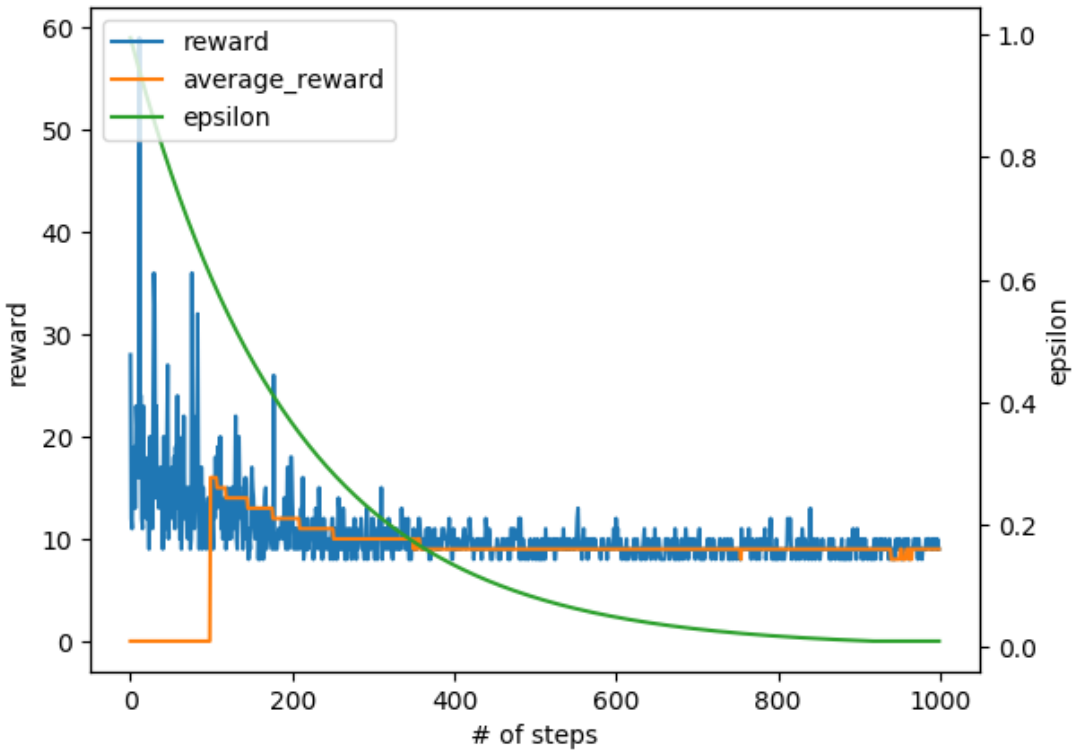

Abbildung 4.1: Beispiel einer Messung mit 1000 Episoden, ϵ_{decay} = 0.995, 1 Hidden Layer mit 12 Neuronen, restliche Parameter wie in Tabelle 4.1

Am Beispiel 4.1 sieht man, dass mit 1000 Episoden damit keine nennenswerten Ergebnisse erzielt werden. Andere Architekturen zeigten teilweise noch schlechtere Ergebnisse. Die durchschnittliche, gesammelte Belohnung „average_reward“ ist mit Werten zwischen 10 und 20 nicht konkurenzfähig. Die Modellqualität stagniert, es ist keine Verbesserung der durchschnittlichen, gesammelten Belohnung zu erwarten.
Um die in Kapitel 2.8.2 genannten Anforderungen zu erfüllen kam es nicht infrage die Anzahl der Episoden zu erhöhen. Stattdessen wurden Erweiterungen ausgesucht und implementiert, die bei gleichbleibender Zahl an Episoden die Modellqualität erhöhen.

Dabei fiel die Wahl auf das in Kapitel 2.7.2 vorgestellte Double Q Learning und das in Kapitel 2.7.3 vorgestellte Memory Replay. Unter Zuhilfenahme beider Methoden wurden erste Architekturversionen miteinander verglichen.

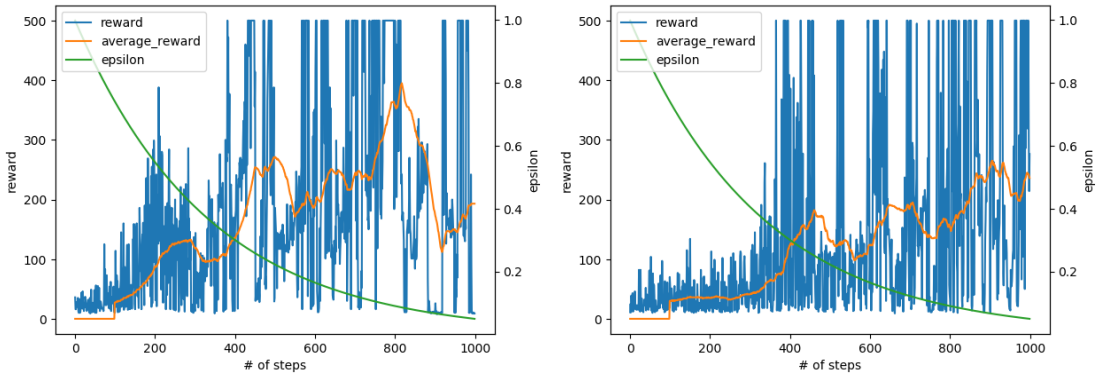

Abbildung 4.2: (links) Gelöst nach 432 Episoden: 1 Hidden Layer mit 12 Neuronen, (rechts) Gelöst nach 706 Episoden: 2 Hidden Layer mit 24 Neuronen. $\epsilon_{decay} = 0.997$ für beide Modelle, nicht genannte Parameter wie in Tabelle 4.1

An den Messungen in Abbildung 4.2 sieht man, dass die durchschnittliche, gesammelte Belohnung nun den Wert 200 erreicht und tendentiell sogar übersteigt. Abgebildet sind die besten Ergebnisse aus jeweils 3 Messungen pro Modell, mit zum einen einem Hidden-Layer und 12 Neuronen, und zum anderen 2 Hidden Layern mit jeweils 24 Neuronen. Andere getestete Architekturen mit der gleichen Anzahl an Hidden-Layern wie in den abgebildeten Messungen lieferten vergleichbare Ergebnisse. Tendentiell erreicht also eine Modellarchitektur mit einem Hidden-Layer die in 2.8.2 aufgestellten Anforderungen schneller als eine Modellarchitektur mit mehr als einem Hidden-Layer, die im Best Case immerhin fast doppelt so viele Episoden benötigten. Da die Ergebnisse allerdings noch immer nah beieinander liegen, werden im weiteren Verlauf weitere Messungen vorgenommen. Allerdings zeichnet sich schon jetzt ab, dass die Modellarchitekturen mit mehr als einem Hidden-Layer zwar länger brauchen um das Problem zu lösen, dafür allerdings ihre Qualität weniger fluktuiert. Dies lässt sich auf deren erhöhte Komplexität zurückführen, die für die Problemstellung des CartPole-Spiels tendentiell nicht notwendig ist. Sie brauchen durch ihre erhöhte Anzahl an Neuronen potentiell länger um zu einer optimalen Lösung zu konvergieren, wären aber bei komplexeren Spielen notwendig, deren Komplexität einschichtige Netzarchitekturen nicht abbilden können.
In Abbildung 4.2 sieht man, dass eine Lösung des Spiels nach ca 430 Episoden möglich ist,

obwohl die Explorationsrate zu dem Zeitpunkt mit ca 30% noch recht hoch ist. Es bot sich deshalb an die Explorationsrate anzupassen um möglicherweise eine Lösung des Spiels in weniger Episoden zu erreichen.
Die Explorationsrate beginnt in jedem Versuchsdurchlauf bei 1, was dazu führt, dass zu Beginn 100% der Aktionen zufällig gewählt werden. Nach jeder Episode wird ein Faktor $0 < \epsilon_{decay} < 1$ auf den aktuellen ϵ Wert multipliziert, und dieser somit verringert. Wie in Kapitel 2.8 erläutert werden Aktionen dadurch mit fortgeschrittener Lerndauer immer häufiger vom Modell ausgewählt. Im folgenden wurden verschiedene Epsilon-Verfallraten ϵ_{decay} miteinander verglichen. Dazu wurden jeweils 180 Episoden für das Training benutzt und nur der ϵ_{decay} Parameter geändert.

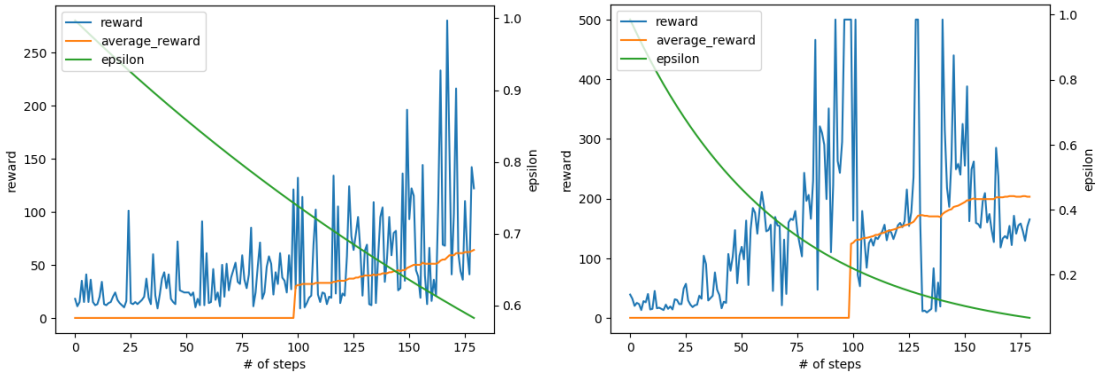

Abbildung 4.3: (links) nicht gelöst innerhalb von 180 Episoden mit $\epsilon_{decay} = 0.997$, (rechts) gelöst nach 151 Episoden mit $\epsilon_{decay} = 0.985$. Beide Modelle mit 1 Hidden Layer mit 12 Neuronen, nicht genannte Parameter wie in Tabelle 4.1

In Abbildung 4.3 werden zwei Messungen gezeigt, die den Einfluss des ϵ_{decay} gut verdeutlichen. In Abbildung 4.3 links sieht man einen langsamen Verfall der Explorationsrate. Nach 100 gespielten Episoden werden immer noch mehr als 70% der Aktionen zufällig ausgewählt. Dementsprechend liegt die Modellqualität durchschnittlich deutlich unter der in Abbildung 4.3 rechts. Dort werden nach 100 Episoden nur noch 20 % der Aktionen zufällig gewählt, die Modelqualität nach 100 Episoden ist fast doppelt so gut wie in der verglichenen Abbildung. In Abbildung 4.3 rechts gelang es das Spiel nach 51 Episoden zu lösen, in Abbildung 4.2 links wurden mehr als 400 Episoden benötigt. Die Explorationsrate und ihr Verfall üben also einen starken Einfluss auf die Ergebnisse eines Lernverfahrens aus und sollten abhängig von der Problemstellung unterschiedliche Werte annehmen.

Ein weiteres Tuning der ϵ_{decay} dürfte helfen noch bessere Ergebnisse zu erzielen, für die in Kapitel 2.8.2 gestellten Anforderungen war es in dieser Thesis jedoch nicht notwendig weiter nach einem optimalen Wert zu suchen.
Mit dem nun gefundenen Wert für die Abnahme der Explorationsrate wurde beispielhaft an einem Modell gezeigt, welchen Anteil an der Qualitätssteigerung der Netze das Memory Replay, beziehungsweise das Double Q Learning haben. Die Größe der Replay Memory wurde dabei vorerst auf 2000 gesetzt. Auf ein Feintuning dieses Parameters wurde in dieser Thesis verzichtet. Nach Liu und Zou (2017) kann dieser Parameter aber ebenfalls größeren Einfluss auf die Modellqualität ausüben.

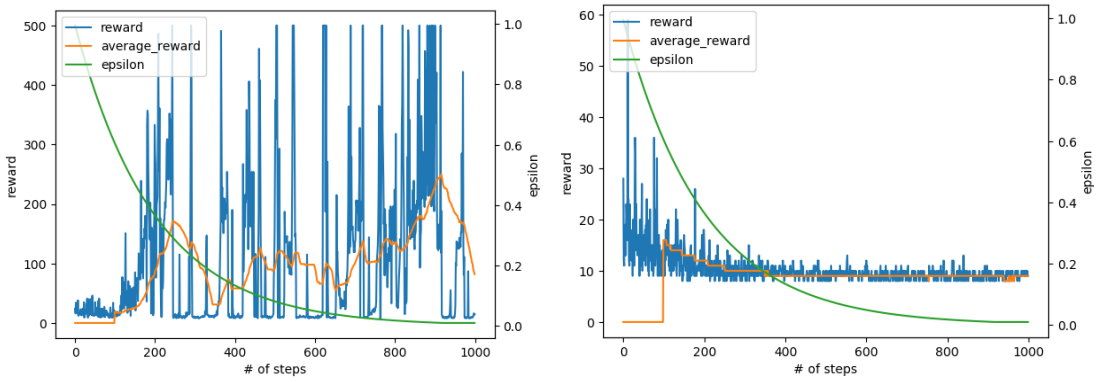

Abbildung 4.4: (links) Mit Memory Replay, (rechts) ohne Memory Replay. Beide Modelle ohne Double Q Learning, nicht genannte Parameter wie in Tabelle 4.1

In Abbildung 4.4 sieht man, dass bei sonst gleichen Parametern und gleicher Modellarchitektur die Benutzung einer Replay Memory eine erhebliche Verbesserung der Modellqualität herbeiführt. Während in Abbildung 4.4 rechts keine Verbesserung der Modellqualität innerhalb der Trainingszeit zu beobachten ist, sieht man in Abbildung 4.4 links eine erhebliche Steigerung um Faktor 10 und mehr. Letztgenannte löst das Spiel nach 782 Episoden ohne die Benutzung des Double Q Learning (vergl. Kapitel 2.7.2).
Hier muss jedoch beachtet werden, dass die Benutzung des Memory Replay eine Steigerung der Anzahl an Updates auf dem Modell zur Folge hat. Nach jeder getätigten Aktion wird ein Replay auf Erfahrungen aus der Memory mit einer Sample-Größe von 32 ausgeführt. Bei 1000 Trainings-Episoden entspricht das in etwa 32000 Gradientenabstiegsverfahren auf dem Künstlichen Neuronalen Netz. In der folgenden Abbildung sieht man ein Netz, das zum Vergleich

ohne Memory Replay, jedoch mit 3200 Episoden trainiert wurde.

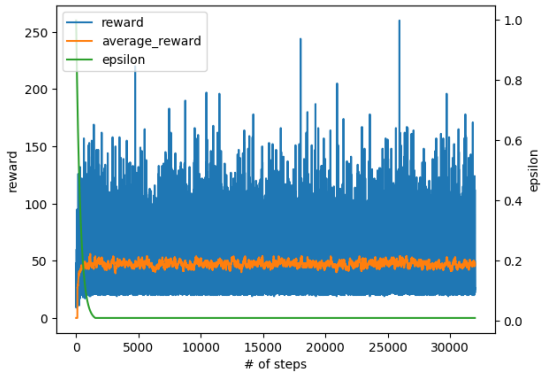

Abbildung 4.5: gleiches Modell wie in Abbildung 4.4 rechts, über 32000 Episoden trainiert. Nicht genannte Parameter wie in Tabelle 4.1

Man sieht, dass die Netzqualität trotz einer ähnlichen Zahl an Trainingsdurchläufen erheblich schlechter ist. Das Modell erreicht zwar zum Teil gute Ergebnisse in einzelnen Episoden, die durchschnittliche Modellqualität fluktuiert allerdings extrem. Dies zeigt die Vorteile des Memory Replay, bei dem potentiell wichtige Erfahrungen mehr als einmal zum Training benutzt werden, und somit eine zielgerichtete Konvergenz zu einer optimalen Lösung begünstigen.
In der Abbildung 4.4 links sieht man, dass auch bei Benutzung des Memory Replay die Qualität des Modells fluktuiert. Dies könnte daran liegen, dass nach einer Weile des Trainings ein gutes Modell vorliegt, dass viele Episoden hintereinander gute Ergebnisse erzielt. Dies vertreibt Negativbeispiele aus der Replay Memory, woraufhin das Modell nach einer Weile verlernt gute Ergebnisse zu erzielen. In Kapitel 5 wird eine mögliche Erweiterung vorgeschlagen, die das Problem potentiell beheben könnte.

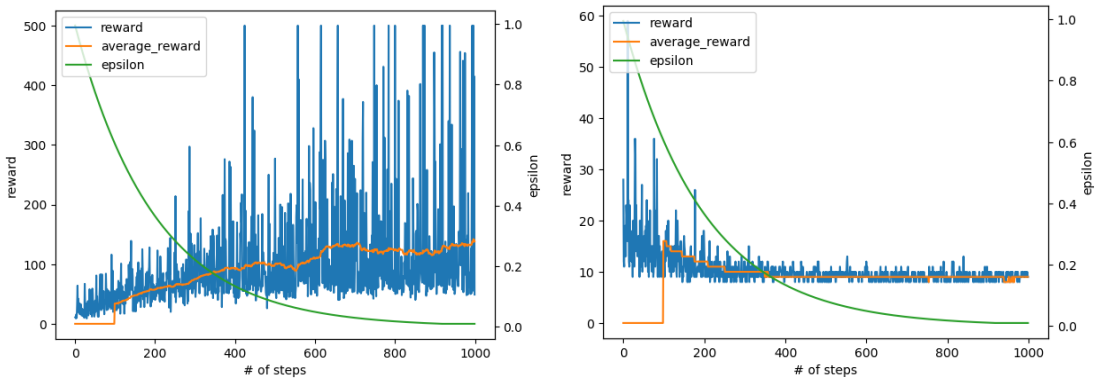

Abbildung 4.6: links mit Double Q Learning, rechts ohne Double Q Learning. Beide Modelle ohne Memory Replay, nicht genannte Parameter wie in Tabelle 4.1

Die zweite getätigte Erweiterung am Modell ist das Double Q Learning. In Abbildung 4.6 links sieht man, dass verglichen zur Abbildung rechts die Benutzung des Double Q Learning eine erhebliche Steigerung der Durchschnittsqualität zur Folge hat. Dies liegt daran, dass mit dem Double Q Learning die Zielwerte nicht nach jedem Schritt, sondern nur nach jeder Episode angepasst werden, und es somit für das Netz einfacher ist ein Minimum der Kostenfunktion zu finden, weil das Ziel nicht nach jedem Schritt ein Neues ist. In dem Beispiel dieser Thesis ist der Einfluss des Double Q Learning auf die Lerngeschwindigkeit nicht so stark wie der Einfluss des Memory Replay.
Nach Van Hasselt u. a. (2016) hat die Einführung des Double Q Learning nicht immer einen Ansteig der Lerngeschwindigkeit zur Folge, meist aber einen Anstieg der Stabilität. Dies sieht man in Abbildung 4.6 links, wo die Modellqualität im Vergleich viel stabiler wirkt als in Abbildung 4.4 links.
Zuletzt wurden drei Netzarchitekturen mit jeweils einem Hidden Layer mit jeweils 6, 12 und 24 Neuronen miteinander verglichen. Alle Messungen wurden unter Zuhilfenahme von Memory Replay und Double Q Learning, mit $\epsilon_{decay} = 0.985$ und den in Tabelle 4.1 genannten Hyperparametern durchgeführt. Für den Vergleich wurde jeweils das beste Ergebnis aus Drei Trainingsläufen herangezogen.

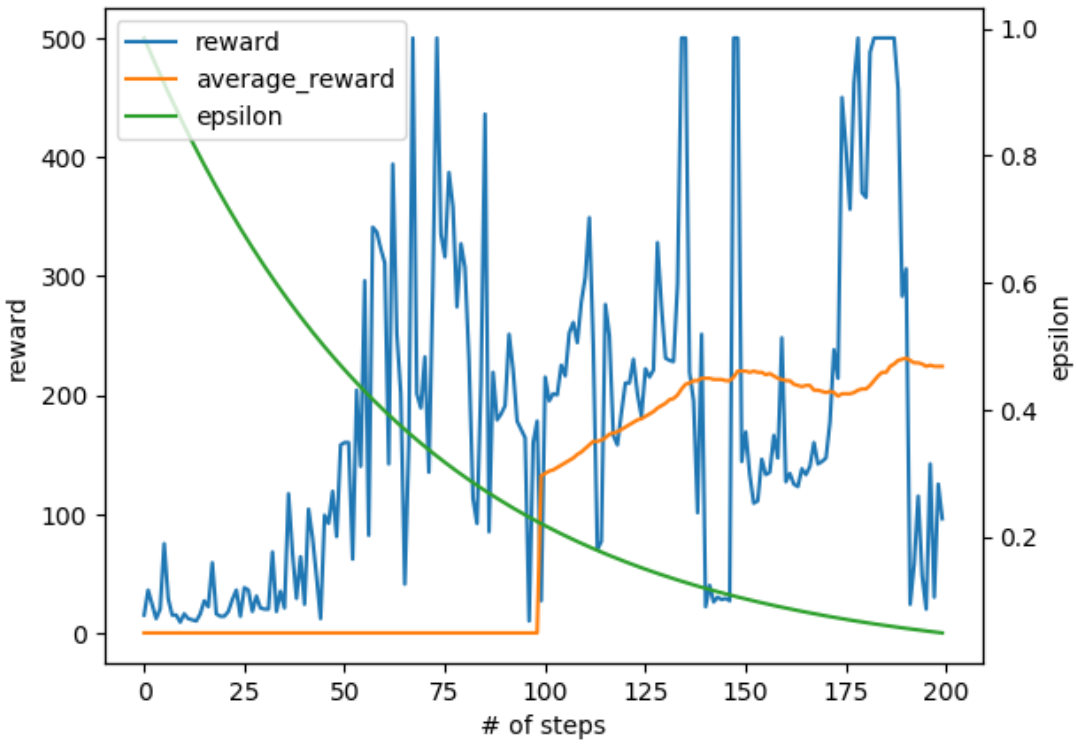

Abbildung 4.7: gelöst nach 31 Episoden mit 1 Hidden Layer mit 24 Neuronen, nicht genannte Parameter wie in Tabelle 4.1

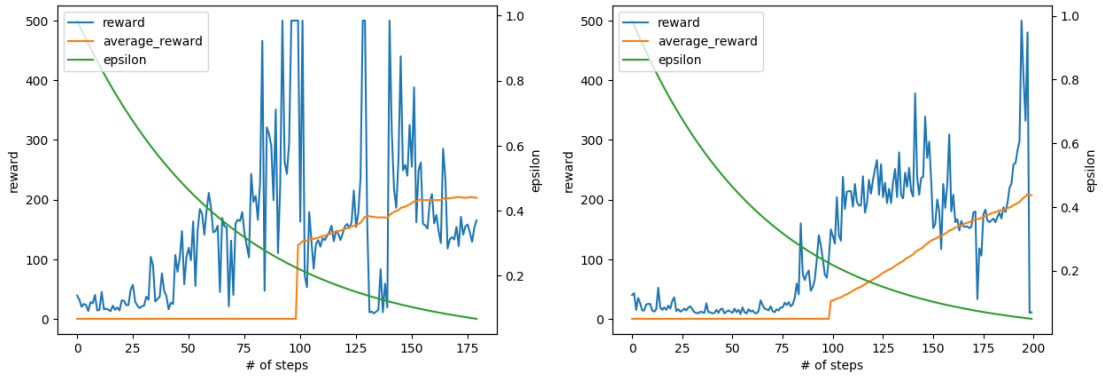

Abbildung 4.8: (links) gelöst nach 51 Episoden mit 1 Hidden Layer mit 12 Neuronen, (rechts) gelöst nach 93 Episoden mit 1 Hidden Layer mit 6 Neuronen, nicht genannte Parameter wie in Tabelle 4.1

Das in dieser Thesis beste gefundene Modell konnte das CartPole Spiel nach 31 Episoden lösen, also ein Modell trainieren, das nach 31 Episoden über die 100 folgenden Spiele eine durchschnittliche, gesammelte Belohnung >= 195 erreichen konnte. Das in dieser Thesis beste gefundene Modell besteht aus einem Hidden-Layer mit 24 Neuronen. Allerdings wurden mit jedem Modell nur 3 Messungen gemacht, eine absolute Aussage über eine optimale Architektur ließe sich nur mit mehr Messungen und einer anderen Zielsetzung treffen.

4.2 Probleme

Ein großes Problem der Künstlichen Neuronalen Netze in diesem Anwendungsfall ist ihre Instabilität. Sowohl die kurzfristigen Belohnungen, als auch die Modellqualität (dargestellt durch „average_reward“) fluktuieren mit fortschreitender Lerndauer teilweise gravierend. Obwohl der Trend der Entwicklung der Modellqualität positiv ist, muss, je nach Anwendungsgebiet, ein Modell zuverlässig gute Ergebnisse liefern um für reale Anwendungen benutzt werden zu können. So ist es zum Beispiel bei der Steuerung eines Autos im Straßenverkehr durch ein Künstliches Neuronales Netz wichtig, dass dieses vom Modell zuverlässig gesteuert wird.
Die Frage wie gut KNN geeignet sind um kritische Probleme zu lösen kann in dieser Thesis nicht behandelt werden. Angelehnt an die Ergebnisse der Messungen 4.2 wäre die erste Überlegung die Wahl einer komplexeren Netzarchitektur, die zwar langsamer, dafür aber stabiler trainiert.

4.3 Bewertung

Das im Rahmen dieser Thesis formulierte Ziel ein Modell zu implementieren, das in annähernd konkurrenzfähiger Anzahl an Episoden das CartPole-Spiel löst, konnte erreicht werden. Es gelang das Spiel in 31 Episoden zu lösen. Dabei ist zu erwähnen, dass es im Gegensatz zu den Ergebnissen des vorgestellten Leaderboards (vergl. OpenAI (2018c)) in dieser Thesis einfacher war eine durchschnittliche, gesammelte Belohnung zu erreichen, da in der hier benutzten Implementation das Spiel CartPole nicht nach 200, sondern erst nach 500 Schritten terminiert. So können einige sehr gute Ergebnisse die durchschnittliche, gesammelte Belohnung stark positiv beeinflussen. Das erreichte Ergebnis wird trotzdem als vergleichbar angesehen, da die in dieser Thesis vorgestellte Lösung gänzlich auf das Einpflegen von Domänenwissen verzichtet und kein umfassendes Feintuning aller Hyperparameter vornimmt.
Das in dieser Thesis erreichte Ergebnis ist also im Vergleich mit den Referenzwerten des Leaderboards zufriedenstellend, da mit einem Feintuning der Hyperparameter wie der Lernrate α, der Netzarchitektur, der Verfallsgeschwindigkeit ϵ_{decay}, der Größe der Experience Memory und der Sample-Batches beim Experience Replay mit sehr hoher Wahrscheinlichkeit ein stark konkurrenzfähiges Modell für die im Leaderboard benutzte Implementation des CartPole Spiels erreicht werden könnte.
Ein wichtiger Aspekt der in dieser Thesis verfolgten Definition einer Lösung des Spiels ist der Fokus auf die schnelle Lösung des Spiels, bei der das Modell unter anderem durch schnelle Abnahme der Explorationsrate und einer geringen Lernrate ein hohes Risiko birgt in lokalen Minima zu verbleiben, anstatt ein globales Minimum der Kosten zu finden. Wie in Kapitel 4.2 erwähnt ist es für die Anwendung Künstlicher Neuronaler Netze auf reale Anwendungsbereiche wichtig ein zuverlässiges Modell zu trainieren. Dabei steht die Qualität des Netzes in gewisser Weise oft umgekehrt proportional zur Trainingsgeschwindigkeit. Verlangen die Anforderungen also ein zuverlässiges Modell, sollte das vorrangige Ziel beim Tuning der Hyperparameter nicht sein möglichst schnell zu einer Lösung zu kommen, sondern die Architektur und die Lernmethoden so zu wählen, dass ein möglichst zuverlässiges Netz trainiert wird. Dabei muss mit einer erhöhten Trainingsdauer kalkuliert werden.
Da die Benutzung des Memory Replay in dieser Thesis eine erhebliche Beschleunigung des Lernverfahrens zur Folge hatte, wäre eine naheliegende Erweiterung einer Architektur für das CartPole-Spiel die Einführung des „Priorisierten Memory Replay “ (vergl. Schaul u. a. (2015)), bei der gemachte Erfahrungen nach ihren Kosten priorisiert werden. Dies könnte der in Kapitel 4.2 genannten Fluktuation der Modellqualität entgegenwirken, indem unwichtige Erfahrungen kürzer in der Experience Memory verbleiben und wichtige Erfahrungen öfter für das Gradi-

entenabstiegsverfahren benutzt werden. Eine weitere Erweiterung könnte das von Mnih u. a. (2015) genannte Clipping der Kosten sein, das eine weitere Verbesserung der Stabilität der Lernalgorithmen zur Folge hat.

5 Fazit & Ausblick

Grundsätzlich ist die Anwendung von Künstlichen Neuronalen Netzen auf das CartPole Problem nicht gut gewählt, für das Spiel gibt es bessere Lösungen. Eine lineare Lösung, beispielweise eine einfache If-Else Schleife würde wohl bessere Ergebnisse als das in dieser Thesis vorgestellte Modell liefern. Trotzdem lassen sich aus den Ergebnissen dieser Arbeit Rückschlüsse über die Vorteile der Benutzung von Künstlichen Neuronalen Netzen für unterschiedliche Probleme ziehen.
Der größte Vorteil Künstlicher Neuronaler Netze für die Lösung von Problemen ist deren Unabhängigkeit von Domänenwissen. Eine lineare Lösung, egal welchen Problems, benötigt ein Wissen über die Domäne, in der das Problem liegt. Im Rahmen dieser Thesis wurde deswegen bewusst kein Domänenwissen in das Modell eingepflegt. Dem Modell wurde der gesamte Zustandsraum übergeben und die Auswahl der Aktionen wurde komplett vom Netz übernommen.
An den Ergebnissen dieser Thesis sieht man, dass KNN auch ohne Domänenwissen in der Lage sind gute, bis teilweise sehr gute Ergebnisse in komplexeren Anwendungsbereichen zu erreichen. Dies ermöglicht generalisierte Lösungen, wie zum Beispiel die Lösung unterschiedlichster Spiele mit einem einzigen Algorithmus wie von Mnih u. a. (2015) gezeigt. Auch die Steuerung von Autos im Straßenverkehr (vergl. Bojarski u. a. (2016)) oder die Vorhersage über das Kaufverhalten von Kunden werden mittlerweile zum Teil von Machine Learning Algorithmen übernommen. Diese können unter anderem neue Lösungsstrategien für bereits bekannte Probleme finden, wie beispielsweise neue Strategien für das Schach Spiel, da beim Training eines Künstlichen Neuronalen Netzes, unter anderem durch die Exploration, Strategien ausprobiert werden, die nach menschlicher Betrachtung als sinnlos oder umständlich erachtet würden.
Da in dieser Thesis mit einer Simulation gearbeitet wurde könnte ein nächster Schritt darin bestehen die verwendeten Verfahren auf ein Problem der realen Welt zu übertragen. Dazu bieten sich besonders Probleme an, für die noch keine guten Datensätze existieren, die vorhandenen Datensätze zu klein sind oder nicht in genügender Qualität vorliegen. Weiter denkbar ist eine Auseinandersetzung mit den in Kapitel 4.3 genannten Verbesserungen des Lernverfahrens.

Die guten Ergebnisse bei der Benutzung von Deep Learning sind der Grund, dass das Feld des maschinellen Lernens aktuell stark erforscht wird. Dabei ergeben sich immer mehr spannende Anwendungsfelder, wie die Übertragung der Mimik eines Schauspielers auf eine Videoaufnahme einer beliebigen anderen Person, wie z.B. eines Präsidenten (vergl. Kim u. a. (2018)) oder die immer besser werdende Spracherkennung und Übersetzung (vergl. Wu u. a. (2016)), beides mithilfe Künstlicher Neuronaler Netze. Machine Learning hat das Potential unsere Gesellschaft nachhaltig zu verändern.
Diese Veränderung geschieht leider auch in Feldern, in denen sie potentiell eine Gefahr darstellen könnte. Wenn Machine Learning Algorithmen in Zukunft über die vorzeitige Freilassung von Gefängnisinsassen oder die Bewilligung/Nicht-Bewilligung von Krediten und Versicherungen entscheiden, dann bieten sie neben den zuvor genannten Vorteilen ebenso die in dieser Thesis behandelten Nachteile. Für eine nachhaltige Nutzung von Machine Learning Algorithmen muss sichergestellt werden, dass sie auf guten Trainingsdaten und mit Verfahren trainiert werden, die eine hohe Stabilität und Verlässlichkeit sicherstellen. Im besten Fall ist für jede Entscheidung eines Modells nachvollziehbar, wie sie zustande gekommen ist. Dies wird zum Teil schon in der Wissenschaft erforscht (Explainable AI Gunning (2017)), muss aber auch im gesellschaftlichen Kontext vermehrt kommuniziert werden (vergl. Newell und Marabelli (2015)). Machine Learning in Form von Künstlichen Neuronalen Netzen kann in Zukunft beachtliche Erfolge mit sich bringen und möglicherweise einige der in Kapitel 1 genannten Visionen in naher Zukunft erreichbar machen. Damit diese neuen Möglichkeiten einen positiven Einfluss auf die Gesellschaft ausüben, muss das Feld des maschinellen Lernens und aller damit verbundenen Implikationen weiter erforscht und beobachtet werden.

Literaturverzeichnis

[ACM Digital Library 2018] ACM Digital Library: *ACM Digital Library (2018).* https://dl.acm.org/. 2018. – Accessed: 2018-09-29

[Barto u. a. 1988] Barto, Andrew G. ; Sutton, Richard S. ; Anderson, Charles W.: Neurocomputing: Foundations of Research. Cambridge, MA, USA : MIT Press, 1988, Kap. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, S. 535–549. – URL http://dl.acm.org/citation.cfm?id=65669.104432. – ISBN 0-262-01097-6

[Bojarski u. a. 2016] Bojarski, Mariusz ; Testa, Davide D. ; Dworakowski, Daniel ; Firner, Bernhard ; Flepp, Beat ; Goyal, Prasoon ; Jackel, Lawrence D. ; Monfort, Mathew ; Muller, Urs ; Zhang, Jiakai ; Zhang, Xin ; Zhao, Jake ; Zieba, Karol: End to End Learning for Self-Driving Cars. In: *CoRR* abs/1604.07316 (2016). – URL http://arxiv.org/abs/1604.07316

[Chollet u. a. 2015] Chollet, François u. a.: *Keras.* https://keras.io. 2015

[DeepMind 2018] DeepMind: *DeepMind.* https://deepmind.com/research/. 2018. – Accessed: 29-09-2018

[Glorot u. a. 2011] Glorot, Xavier ; Bordes, Antoine ; Bengio, Yoshua: Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf, 2011, S. 315–323

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

[Gunning 2017] Gunning, David: Explainable artificial intelligence (xai). In: *Defense Advanced Research Projects Agency (DARPA), nd Web* (2017)

[Jürgen Cleve 2014] Jürgen Cleve, Uwe L.: *Data mining.* De Gruyter, 2014

[Kaelbling u. a. 1996] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement learning: A survey. In: *Journal of artificial intelligence research* 4 (1996), S. 237–285

[Kim u. a. 2018] Kim, Hyeongwoo ; Garrido, Pablo ; Tewari, Ayush ; Xu, Weipeng ; Thies, Justus ; Niessner, Matthias ; Pérez, Patrick ; Richardt, Christian ; Zollhöfer, Michael ; Theobalt, Christian: Deep Video Portraits. In: *CoRR* abs/1805.11714 (2018). – URL http://arxiv.org/abs/1805.11714

[Lin 1992] Lin, Long-Ji: Self-improving reactive agents based on reinforcement learning, planning and teaching. In: *Machine Learning* 8 (1992), May, Nr. 3, S. 293–321. – URL https://doi.org/10.1007/BF00992699. – ISSN 1573-0565

[Liu und Zou 2017] Liu, Ruishan ; Zou, James: The Effects of Memory Replay in Reinforcement Learning. In: *CoRR* abs/1710.06574 (2017). – URL http://arxiv.org/abs/1710.06574

[Martin 2009] Martin, Robert C.: *Clean code: A handbook of agile software craftsmanship*. Boston, MA : Prentice-Hall, 2009 (Robert C. Martin series). – URL http://cds.cern.ch/record/1281586

[Melo 2001] Melo, Francisco S.: Convergence of Q-learning: A simple proof. In: *Institute Of Systems and Robotics, Tech. Rep* (2001), S. 1–4

[Mnih u. a. 2013] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin A.: Playing Atari with Deep Reinforcement Learning. In: *CoRR* abs/1312.5602 (2013). – URL http://arxiv.org/abs/1312.5602

[Mnih u. a. 2015] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg ; Petersen, Stig ; Beattie, Charles ; Sadik, Amir ; Antonoglou, Ioannis ; King, Helen ; Kumaran, Dharshan ; Wierstra, Daan ; Legg, Shane ; Hassabis, Demis: Human-level control through deep reinforcement learning. In: *Nature* 518 (2015), 02, S. 529 EP –. – URL http://dx.doi.org/10.1038/nature14236

[Newell und Marabelli 2015] Newell, Sue ; Marabelli, Marco: Strategic opportunities (and challenges) of algorithmic decision-making: A call for action on the long-term societal effects of 'datification'. In: *The Journal of Strategic Information Systems* 24 (2015), Nr. 1, S. 3–14

[OpenAi(2018a) 2018] OpenAi(2018a): *CartPole-v1*. https://gym.openai.com/envs/CartPole-v1/. 2018. – Accessed: 08-12-2018

[OpenAi(2018b) 2018] OpenAi(2018b): *CartPole-v0*. https://gym.openai.com/envs/CartPole-v0/. 2018. – Accessed: 08-12-2018

[OpenAI(2018c) 2018] OpenAI(2018c): *OpenAI Leaderboard*. https://github.com/openai/gym/wiki/Leaderboard. 2018. – Accessed: 08-12-2018

[Rossum 1995] Rossum, Guido: Python Reference Manual. Amsterdam, The Netherlands, The Netherlands : CWI (Centre for Mathematics and Computer Science), 1995. – Forschungsbericht

[Russell und Norvig 2016] Russell, Stuart J. ; Norvig, Peter: *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited„ 2016

[Schaul u. a. 2015] Schaul, Tom ; Quan, John ; Antonoglou, Ioannis ; Silver, David: Prioritized Experience Replay. In: *CoRR* abs/1511.05952 (2015). – URL http://arxiv.org/abs/1511.05952

[Silver u. a. 2017] Silver, David ; Schrittwieser, Julian ; Simonyan, Karen ; Antonoglou, Ioannis ; Huang, Aja ; Guez, Arthur ; Hubert, Thomas ; Baker, Lucas ; Lai, Matthew ; Bolton, Adrian ; Chen, Yutian ; Lillicrap, Timothy ; Hui, Fan ; Sifre, Laurent ; Driessche, George van den ; Graepel, Thore ; Hassabis, Demis: Mastering the game of Go without human knowledge. In: *Nature* 550 (2017), 10, S. 354 EP –. – URL http://dx.doi.org/10.1038/nature24270

[Stathakis 2009] Stathakis, D.: How many hidden layers and nodes? In: *International Journal of Remote Sensing* 30 (2009), Nr. 8, S. 2133–2147. – URL http://www.academia.edu/711697/How_many_hidden_layers_and_nodes

[Sutton und Barto 1998] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction*. MIT Press, 1998. – URL http://www.cs.ualberta.ca/~sutton/book/the-book.html

[Sutton(o.D)] Sutton(o.D): *Implementation des CartPole Problems nach Barto, Sutton, und Anderson*. https://perma-archives.org/warc/C9ZM-652R/http://incompleteideas.net/sutton/book/code/pole.c. – Accessed: 08-12-2018

[Tsitsiklis und Roy 1997] Tsitsiklis, J. N. ; Roy, B. V.: An analysis of temporal-difference learning with function approximation. In: *IEEE Transactions on Automatic Control* 42 (1997), May, Nr. 5, S. 674–690. – ISSN 0018-9286

[Van Hasselt u. a. 2016] Van Hasselt, Hado ; Guez, Arthur ; Silver, David: Deep Reinforcement Learning with Double Q-Learning. In: *AAAI* Bd. 2 Phoenix, AZ (Veranst.), 2016, S. 5

[Watkins 1989] Watkins, C.J.C.H.: *Learning from Delayed Rewards*, Cambridge University, Dissertation, 1989. – URL `http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf`

[Wu u. a. 2016] Wu, Yonghui ; Schuster, Mike ; Chen, Zhifeng ; Le, Quoc V. ; Norouzi, Mohammad ; Macherey, Wolfgang ; Krikun, Maxim ; Cao, Yuan ; Gao, Qin ; Macherey, Klaus ; Klingner, Jeff ; Shah, Apurva ; Johnson, Melvin ; Liu, Xiaobing ; Kaiser, Lukasz ; Gouws, Stephan ; Kato, Yoshikiyo ; Kudo, Taku ; Kazawa, Hideto ; Stevens, Keith ; Kurian, George ; Patil, Nishant ; Wang, Wei ; Young, Cliff ; Smith, Jason ; Riesa, Jason ; Rudnick, Alex ; Vinyals, Oriol ; Corrado, Greg ; Hughes, Macduff ; Dean, Jeffrey: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. In: *CoRR* abs/1609.08144 (2016). – URL `http://arxiv.org/abs/1609.08144`

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. Dezember 2018 Jerom Schult