Tiefes Q-Lernen mit Demonstrationen

Felix Kerkhoff

Geboren am 7. Juli 1997 in Bergisch Gladbach

23. Juni 2020

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jochen Garcke

Zweitgutachterin: Prof. Dr. Ira Neitzel

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Danksagung

Zunächst bedanke ich mich herzlich bei Herrn Prof. Dr. Jochen Garcke für die auch in Zeiten der Pandemie-bedingten Ausgangssperre stets gute und hilfreiche Betreuung meiner Arbeit. Weiterer Dank gilt Frau Prof. Dr. Ira Neitzel für die Übernahme der Zweitkorrektur. Zudem bedanke ich mich bei Herrn Prof. Dr. Herbert Koch stellvertretend für den gesamten Prüfungsausschuss für die große Kulanz bei der Verlängerung der Abgabefristen, die die Durchführung von einigen zusätzlichen Experimenten möglich gemacht hat.

Außerdem gebührt mein Dank allen Entwicklern, die zu der Erstellung frei zugänglicher Software, wie z.B. LaTeX oder *Tensorflow*, ohne die eine Arbeit in der vorliegenden Form kaum vorstellbar wäre, beigetragen haben.

Nicht zuletzt bedanke ich mich bei meiner Familie dafür, dass sie mich während des gesamten Studiums mit allen Mitteln unterstützt haben. Besonderer Dank gilt hierbei meiner Mutter für das Korrekturlesen der Arbeit und das Angebot von Kost und Logis in der Heimat während der Ausgangssperre.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Menschen haben die Fähigkeit, ihre Umgebung mithilfe verschiedener Sinnesorgane wahrzunehmen, Erkenntnisse über die Zusammenhänge der wahrgenommenen Vorgänge in ihrer Umgebung zu gewinnen und ihr Verhalten entsprechend dieser Erkenntnisse anzupassen. Kurzum: Menschen besitzen die Fähigkeit, zu lernen.

Der Gedanke, diese Fähigkeit – und andere menschliche Eigenschaften – auf Maschinen zu übertragen, ist so alt wie die Maschinen selbst. So wurde bereits in der mehr als 2000 Jahre alten griechischen Mythologie der von Hephaistos, dem griechischen Gott der Schmiedekunst, aus Bronze hergestellte Riese Talos mit menschenähnlichen Eigenschaften beschrieben. Anfangs noch reine Fiktion, gibt es aktuell, begünstigt durch den raschen technischen Fortschritt insbesondere im Hinblick auf die Rechenleistung von modernen Computern, bereits einige Bereiche, in denen Methoden des maschinellen Lernens erfolgreich eingesetzt werden. Verschiedene Varianten der Bildklassifikation, wie Gesichts- oder Handschrifterkennung und das Lösen von Regressionsproblemen, wie zum Beispiel die Vorhersage von Zeitreihen, sind hier nur wenige Beispiele unter vielen weiteren.

Ein anderer Bereich, mit dem sich die vorliegende Arbeit befassen wird, ist das Lösen von sogenannten *sequentiellen Entscheidungsproblemen*. Diese unterscheiden sich von den zuvor genannten vor allem dadurch, dass nicht nur ein Problem, sondern eine ganze Folge von Entscheidungsproblemen gelöst werden muss. Besonders die Tatsache, dass frühe Entscheidungen spätere beeinflussen können, macht dies zu einer sehr schwierigen Aufgabe, weil dadurch eine langfristige Planung bei der Lösung solcher Probleme nötig wird. Auch in diesem Bereich haben sich verschiedene Verfahren des maschinellen Lernens als gute Lösungsmöglichkeiten erwiesen.

Da diese Verfahren in weiten Teilen an den Lernmethoden von Menschen orientiert sind, folgt nun zunächst ein kurzer Einblick in die für diese Arbeit relevanten psychologischen Lerntheorien und anschließend über ihre mathematischen Gegenstücke.

Psychologische Lerntheorien

Beim Erlernen von Handlungsabfolgen oder Tätigkeiten können Menschen auf unterschiedliche Arten und Weisen vorgehen. Ein erster Ansatz, der wohl die natürlichste Art des Lernens beschreibt und auch bei Tieren häufig zu beobachten ist, besteht darin, zunächst verschiedene Handlungen auszuprobieren und anschließend die Auswirkungen der ausgeführten Handlung zu beobachten. Hatte eine getätigte Handlung

dabei einen positiven Effekt, so wird der Mensch diese Handlung in der Zukunft wahrscheinlich häufiger ausführen. Diese Art des Lernens wird in der psychologischen Lerntheorie als *operante Konditionierung* bezeichnet. Einen bedeutenden Beitrag zu dieser Lerntheorie hat der Psychologe Burrhus Frederic Skinner (siehe [Ski65]) mit der Entwicklung der nach ihm benannten *Skinner Boxen* geleistet. Dies sind kleine, von äußeren Reizen so gut wie möglich abgeschottete Käfige, in denen Tiere verschiedene Handlungen – z.B. das Betätigen eines Hebels – ausführen können. Je nach Experiment haben diese Handlungen entweder eine positive – z.B. die Ausgabe von Futter – oder eine negative – z.B. ein leichter Stromschlag – Folge für das Tier. In seinen Untersuchungen konnte Skinner dabei zeigen, dass Tiere Handlungen, die für sie eine positive Reaktion hervorgerufen haben, in der Folge häufiger ausgeführt haben als solche mit negativen Auswirkungen. Die Tiere haben dabei also anhand der Erfahrungen, die sie durch die Interaktion mit ihrer Umgebung gemacht haben, gelernt.

In der zweiten für diese Arbeit relevanten psychologischen Lerntheorie wird der Fokus beim Lernen nicht nur auf die reine Ausführung von Aktionen und die Beobachtung der Konsequenzen, sondern vor allem auch auf die sozialen und kognitiven Vorgänge beim Lernenden gelegt. In diesem Zusammenhang hat sich die von Albert Bandura (siehe [Ban+86]) beeinflusste Theorie des *Lernens am Modell* etabliert. Hiernach ist es dem Lernenden nicht nur möglich durch die eigene Ausführung von Handlungen zu lernen, sondern auch durch die Beobachtung von sog. Modellen bzw. Vorbildern. Gerade bei Menschen ist diese Form des Lernens häufig zu beobachten. Hierbei kommen ihnen besonders die Fähigkeiten zur Empathie und zur Abstraktion zugute, dass Erfahrungen, die ihre Mitmenschen machen, auch für sie selbst relevant sein könnten.

Maschinelle Lerntheorien

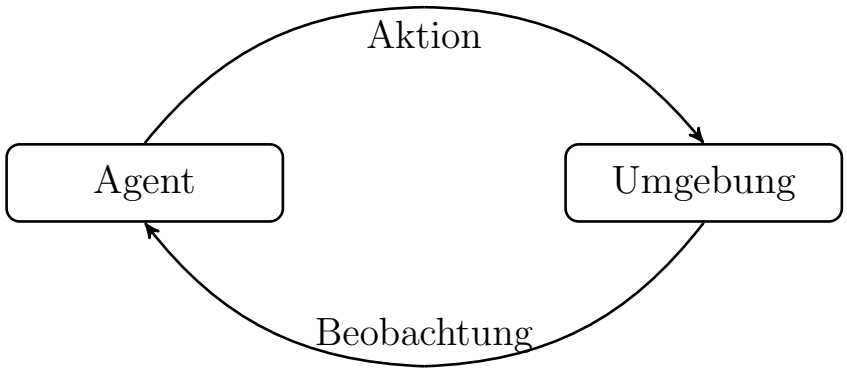


Abbildung 1.1: Allgemeine Struktur von Problemen des bestärkenden Lernens

Analog zu den gerade beschriebenen psychologischen Lerntheorien werden Theorien für das maschinelle Erlernen von sequentiellen Entscheidungsproblemen entworfen. Das Gegenstück der operanten Konditionierung in der Theorie des maschinellen Lernens ist hierbei das sog. *bestärkende Lernen*. Der Hauptbestandteil des bestärkenden Lernens ist ein Computerprogramm, das innerhalb einer bestimmten Umgebung versucht, eine Lösung für ein gegebenes sequentielles Entscheidungsproblem zu finden. Das Computerprogramm wird in diesem Zusammenhang auch *Agent* genannt.

Anders als in der klassischen Programmierung wird dem Agenten im bestärkenden Lernen jedoch nicht vorgegeben, welche Handlungen wann ausgeführt werden müssen. Stattdessen sind die einzigen Informationen, die dem Agenten für das Finden jener Lösung zur Verfügung stehen, die Aktionen, die er selbst ausführt, sowie die sich daraus ergebenden Konsequenzen. Diese Konsequenzen können sich zum einen in der Änderung des *Zustands* der Umgebung äußern und zum anderen in einer von der Umgebung ausgegebenen Belohnung. Das Ziel des Agenten besteht nun darin durch die wiederholte Ausführung von Aktionen und die Beobachtung der Konsequenzen dieser Aktionen Erkenntnisse über die Zusammenhänge seiner Umgebung des zu lösenden Problems zu gewinnen (vgl. Abb 1.1). Mithilfe dieser Erkenntnisse lernt der Agent einzuschätzen, welche Aktionen in einer bestimmten Situation zukünftig zu Belohnungen führen, um auf dieser Grundlage die Summe der erhaltenen Belohnungen zu maximieren.

Das Gegenstück des Lernens am Modell im maschinellen Lernen ist das sog. *Lernen von Demonstrationen* oder auch *Imitationslernen*. Hierbei wird angenommen, dass der Agent für ein gegebenes Problem über einen Datensatz mit Demonstrationen eines Experten in diesem Problem verfügt. Ziel des Agenten ist es, auf Grundlage dieser Demonstrationen ein Verhalten zu entwickeln, welches dem des Agenten möglichst nahe kommt. Dabei sollte er jedoch nicht einfach die Aktionen des Experten kopieren, sondern abstrahieren, weshalb der Experte in einer gegebenen Situation eine bestimmte Entscheidung getroffen hat. Dies ist nötig, damit der Agent auch in neuen Situationen, die vom gegebenen Datensatz nicht abgedeckt werden, fundierte Entscheidungen treffen kann.

Aufbau und wissenschaftlicher Beitrag der Arbeit

Einer der Hauptbestandteile dieser Arbeit wird es sein, die beiden oben beschriebenen maschinellen Lernmethoden, die in der Regel getrennt betrachtet werden, zu einem gemeinsamen Verfahren, dem in [Hes+18] vorgestellten *tiefen Q-Lernen mit Demonstrationen* (engl. *Deep Q-Learning from Demonstrations*), zusammenzuführen.

Der Aufbau und wissenschaftliche Beitrag der vorliegenden Arbeit umfasst dabei im Wesentlichen die folgenden Punkte:

- die speziell an das tiefe Q-Lernen mit Demonstrationen angepasste Einführung der mathematischen Grundlagen des bestärkenden Lernens (Kapitel 2)
- die detaillierte Darstellung aller in [Hes+18] verwendeten Varianten des tiefen Q-Lernens in einem gemeinsamen Kontext und einer vereinheitlichten – mit den mathematischen Grundlagen in Einklang gebrachten – Notation in deutscher Sprache (Kapitel 3)
- die eigenständige Implementierung all dieser Verfahren (Kapitel 4.2)
- die Konzeption und Durchführung von numerischen Experimenten auf Grundlage dieser Implementierungen mit dem Ziel der Veranschaulichung der einzelnen Bestandteile des tiefen Q-Lernens mit Demonstrationen (Kapitel 4)
- die Einordnung der dabei entstandenen Ergebnisse (Kapitel 5)

Kapitel 2

Mathematische Grundlagen

Die Idee eines Agenten, der einem sequentiellen Entscheidungsproblem ausgesetzt wird und für dieses durch die Interaktion mit seiner Umgebung und mithilfe verschiedener Lernverfahren eine Lösungsstrategie sucht, wurde bereits in der Einleitung thematisiert. Ein erster Schritt auf dem Weg zur Ermittlung einer solchen Lösungsstrategie mit mathematischen Mitteln ist es, jene eher allgemeine Idee mathematisch zu modellieren. Dies wird daher auch der erste Schritt im nun folgenden zweiten Kapitel sein, das in die theoretischen Grundlagen für die Entwicklung von Algorithmen des bestärkenden Lernens einführen soll.

2.1 Problemstellung

Eine Grundlage der mathematischen Modellierung von sequentiellen Entscheidungsproblemen, wie sie in der Einleitung beschrieben sind, bildet der *Zustandsraum* (in der englischen Fachsprache *state space*) $\mathcal{S}$, der die Menge aller möglichen Umgebungszustände des Entscheidungsproblems repräsentiert.

Zu Beginn eines Entscheidungsprozesses befindet sich ein Agent in einem Startzustand $s_0 \in \mathcal{S}$. Von dort ausgehend kann er eine Aktion a_0 aus dem *Aktionsraum* (engl. *action space*) $\mathcal{A}$ aller möglichen Aktionen auswählen und den Übergang der Umgebung in einen neuen Zustand $s_1 \in \mathcal{S}$ sowie eine *Belohnung* (engl. *reward*) $r_1 \in \mathbb{R}$ beobachten. Dieser Vorgang wird nun in einer diskreten Abfolge von Zeitschritten wiederholt, sodass sich ein Entscheidungsprozess s_0, a_0, r_1, s_1, a_1, r_2, s_2, ... ergibt. Existiert eine Zahl $T \in \mathbb{N}$, sodass $s_T \in \mathcal{S}$ ein terminaler Zustand (d.h. $s_t = s_T$ und $r_t = 0$ für alle $t > T$) ist, spricht man auch von einem *episodischen* Entscheidungsprozess.

Da neben den Aktionen des Agenten für gewöhnlich noch weitere Dynamiken in der Umgebung, die nicht notwendigerweise explizit bekannt oder möglicherweise schwer zu modellieren sind, die Zustandsübergänge und Belohnungen beeinflussen, werden diese im mathematischen Modell stochastisch dargestellt. Dadurch werden Zustände s, Aktionen a und Belohnungen r in Zufallsvariablen S, A bzw. R sowie der Entscheidungsprozess s_0, a_0, r_1, s_1, a_1, r_2, s_2, ... in einen stochastischen Prozess S_0, A_0, R_1, S_1, A_1, R_2, S_2, ... überführt.

Geht man hierbei davon aus, dass die Übergangswahrscheinlichkeiten nicht von der gesamten Historie des Prozesses, sondern nur vom aktuellen Zustand und der

gewählten Aktion abhängen – d.h. die *Markov-Eigenschaft*

$$\begin{aligned} & \mathcal{P}[S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} \mid S_t = s_t, A_t = a_t, ..., S_0 = s_0, A_0 = a_0] \\ = \ & \mathcal{P}[S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1} \mid S_t = s_t, A_t = a_t] \end{aligned} \tag{2.1}$$

ist erfüllt – so nennt man den Prozess *Markov-Entscheidungsprozess*. Zusätzlich wird für den Rest der Arbeit vereinfachend angenommen, dass die Belohnung im nächsten Schritt durch den aktuellen Zustand und die aktuelle Aktion determiniert sind. Formal bedeutet dies:

$$\forall \ (s_t, a_t) \in \mathcal{S} \times \mathcal{A} \ \exists \ r_{t+1} \in \mathbb{R} : \mathcal{P}[R_{t+1} = r_{t+1} \mid S_t = s_t, A_t = a_t] = 1 \tag{2.2}$$

In diesem Fall lässt sich die Belohnung als Funktion $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ darstellen und Eigenschaft (2.1) folgendermaßen vereinfachen:

$$\begin{aligned} & \mathcal{P}[S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t, ..., S_0 = s_0, A_0 = a_0] \\ = \ & \mathcal{P}[S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t] \\ =: \ & p(s_{t+1}|s_t, a_t) \end{aligned} \tag{2.3}$$

Diese Art von Prozessen bildet die theoretische Grundlage der meisten Algorithmen des bestärkenden Lernens und wird auch in der vorliegenden Arbeit als solche dienen, weshalb nun zusammenfassend eine formale Definition der oben beschriebenen Markov-Entscheidungsprozesse folgt.

Definition 2.1 Markov-Entscheidungsprozess

Ein *Markov-Entscheidungsprozess* ist ein Tupel $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$, wobei

- $\mathcal{S}$ einen endlichen Zustandsraum,
- $\mathcal{A}$ einen endlichen Aktionsraum,
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0, 1], \quad (s, a, s') \longmapsto p(s'|s, a)$ eine Übergangswahrscheinlichkeit, die Eigenschaft (2.3) erfüllt, und
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ eine Belohnungsfunktion repräsentiert.

Bemerkung 2.2. Da die vorliegende Arbeit vornehmlich das Entwickeln, Implementieren und Testen von Algorithmen des bestärkenden Lernens (siehe Kapitel 3 und 4) behandeln wird, sind die obige Definition und auch die in diesem Kapitel folgenden mathematischen Grundlagen auf das – für die Einführung dieser Algorithmen – Wesentliche reduziert. Notation und allgemeine Vorgehensweise sind hierbei größtenteils an [SB18, Kapitel 3] orientiert. Für eine maß- und wahrscheinlichkeitstheoretisch lückenlose Einführung der mathematischen Grundlagen sei auf die gängige Fachliteratur, wie zum Beispiel [BS07], verwiesen.

Wird nun ein Agent einem Markov-Entscheidungsprozess ausgesetzt, besteht seine Aufgabe in der Regel darin, möglichst viele und hohe Belohnungen zu erhalten. Zu diesem Zweck ist es notwendig, eine gute *Strategie* (engl. *policy*), d.h. allgemein einen – insbesondere auch langfristig ausgelegten – Plan für die Auswahl von Aktionen mit dem Ziel der Belohnungsmaximierung, zu finden. Mathematisch lässt sich der Begriff der Strategie folgendermaßen definieren:

Definition 2.3 Strategie
Eine *Strategie* für einen Markov-Entscheidungsprozess $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ist definiert als eine Abbildung

$$\pi : \mathcal{S} \longrightarrow \mathcal{A}.$$

Bemerkung 2.4. Abweichend von [SB18], wo die Strategie als Abbildung von Zuständen auf Wahrscheinlichkeiten, bestimmte Aktionen auszuwählen, definiert ist, wird hier neben der Belohnungsfunktion auch die Strategie vereinfachend als deterministisch betrachtet.

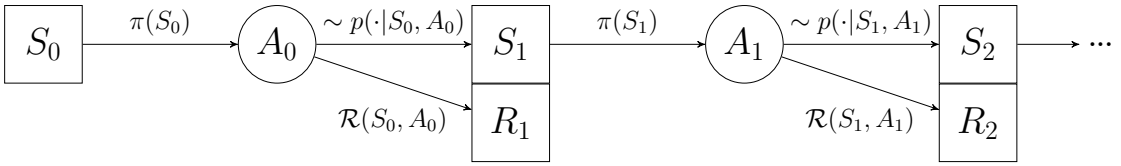


Abbildung 2.1: Struktur eines Markov-Entscheidungsprozesses

Mit den beiden, in diesem Unterkapitel eingeführten Definitionen ist nun der strukturelle Rahmen, der in Abbildung 2.1 zusammenfassend dargestellt ist, für weitere Untersuchungen zum Finden von Lösungsstrategien für Markov-Entscheidungsprozesse gegeben.

2.2 Wertfunktionen

Um mit mathematischen Algorithmen eine gute Strategie für einen Markov-Entscheidungsprozess ermitteln zu können, ist es zunächst wichtig, eine valide Grundlage für die Bewertung von Strategien beziehungsweise von ausgewählten Aktionen zu bestimmen. Der Gedanke, zu diesem Zweck ausschließlich die unmittelbar erhaltenen Belohnungen zu verwenden, mag naheliegend erscheinen, ist tatsächlich jedoch sowohl theoretisch als auch praktisch wenig zielführend. Dies liegt vor allem daran, dass unmittelbare Belohnungen meistens keine Aussage über die langfristigen Auswirkungen von Handlungen zulassen. In praktischen Anwendungen wird die Belohnungsfunktion beispielsweise häufig so entworfen, dass der Agent nur dann eine Belohnung bzw. Bestrafung erhält, wenn er eine bestimmte Aufgabe erfolgreich bzw. erfolglos abgeschlossen hat, während alle vorherigen Aktionen nicht bewertet werden.

Deshalb ist es notwendig eine Bewertungsgrundlage zu entwickeln, die Strategien und Aktionen auch und vor allem hinsichtlich ihrer langfristigen Erfolgsaussichten beurteilt. Dies wird daher Hauptbestandteil dieses Unterkapitels sein.

Der allgemeine Ansatz für die Entwicklung einer solchen, am langfristigen Erfolg orientierten Bewertungsgrundlage ist, eine Einschätzung darüber zu erlangen, welche Belohnungen in der Zukunft noch zu erwarten sind, wenn sich ein Agent in einem bestimmten Zustand befindet und von diesem Zeitpunkt an Aktionen gemäß der zu beurteilenden Strategie auswählt.

Im mathematischen Modell wird zu diesem Zweck zunächst, ausgehend von einem Startzeitpunkt $t \in \mathbb{N}$, die Summe $\sum_{k=0}^{\infty} R_{t+k+1}$ aller sich durch die Ausführung einer

Strategie π ergebenden, zukünftigen Belohnungen betrachtet. Um der größeren Unsicherheit bezüglich Belohnungen, die weiter in der Zukunft liegen, Rechnung zu tragen, wird zusätzlich ein Abnahmefaktor $\gamma \in (0,1)$ eingeführt, der zudem die Konvergenz der neu entstehenden Summe $\sum_{k=0}^{\infty} \gamma^k \ R_{t+k+1}$ sicherstellt. Letzterer Ausdruck wird im Folgenden als *diskontierter Ertrag* (engl. *discounted return*) bezeichnet und nun zusammenfassend definiert.

Definition 2.5 Diskontierter Ertrag

Sei $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess und $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ eine Strategie für M. Dann ist der *diskontierte Ertrag* G_t^π von π zum Zeitpunkt $t \in \mathbb{N}$ mit Abnahmefaktor $\gamma \in (0,1)$ definiert als:

$$G_t^\pi \coloneqq \sum_{k=0}^{\infty} \gamma^k \ R_{t+k+1},$$

wobei $R_{t+k+1} \coloneqq \mathcal{R}(s_{t+k}, a_{t+k})$, $S_{t+k+1} \sim p(\cdot | s_{t+k}, a_{t+k})$ und $a_{t+k} \coloneqq \pi(s_{t+k})$

Für den diskontierten Ertrag lässt sich zudem folgende hilfreiche Eigenschaft, die eine Beziehung vom Wert eines bestimmten Zeitpunkts zum nächsten Zeitpunkt schafft, herleiten.

Lemma 2.6

Der diskontierte Ertrag G_t^π mit Faktor $\gamma \in (0,1)$ einer Strategie π zu einem Markov-Entscheidungsprozess $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ zum Zeitpunkt $t \in \mathbb{N}$ hat folgende Eigenschaft:

$$G_t^\pi = R_{t+1} + \gamma \ G_{t+1}^\pi$$

Beweis.

$$\begin{aligned} G_t^\pi &\coloneqq \sum_{k=0}^{\infty} \gamma^k \ R_{t+k+1} &&= R_{t+1} + \sum_{k=1}^{\infty} \gamma^k \ R_{t+k+1} \\ &= R_{t+1} + \gamma \ \sum_{k=1}^{\infty} \gamma^{k-1} \ R_{t+k+1} &&= R_{t+1} + \gamma \ \sum_{k=0}^{\infty} \gamma^k \ R_{t+1+k+1} \\ &= R_{t+1} + \gamma \ G_{t+1}^\pi \end{aligned}$$

□

Möchte man nun Strategien auf Basis der gerade definierten Größe bewerten, so gibt es hierfür im Wesentlichen zwei weit verbreitete und eng miteinander verwandte Methoden. Beide nutzen die Tatsache aus, dass es sich beim diskontierten Ertrag formal um eine Zufallsvariable handelt, indem sie den bedingten Erwartungswert von G_t^π untersuchen.

Eine der beiden Techniken legt dabei den Fokus auf die Bewertung von erreichten Zuständen. In diesem Sinn wird die Strategie eines Agenten, ausgehend vom Zustand, in dem er sich befindet, nach der Höhe des zu erwartenden diskontierten Ertrages bewertet. Das heißt, es wird der bedingte Erwartungswert $\mathbb{E}[\ G_t^\pi \mid S_t = s_t]$ betrachtet.

Dieser Wert wird *Zustandswert* (engl. *state value*) genannt und mit $v_\pi(s_t)$ abgekürzt (siehe Definition 2.7 (a)).

Die andere Technik bewertet zusätzlich zu den Zuständen insbesondere die im jeweiligen Zustand getätigte Aktion. Analog zur Betrachtung des Zustandswertes wird auch hier der bedingte Erwartungswert $\mathbb{E}[\ G_t^\pi \mid S_t = s_t, A_t = a_t]$ verwendet und als *Aktionswert* (engl. *action value*) $q_\pi(s_t, a_t)$ bezeichnet (siehe Definition 2.7 (b)).

Definition 2.7 Wertfunktionen

Sei $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess und $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ eine Strategie für M.

(a) Dann ist der *Zustandswert* von π für einen Zustand $s_t \in \mathcal{S}$ definiert als:

$$\begin{aligned} v_\pi : \quad \mathcal{S} \quad &\longrightarrow \mathbb{R} \\ s_t \quad &\longmapsto v_\pi(s_t) \coloneqq \mathbb{E}[\ G_t^\pi \mid S_t = s_t] \end{aligned}$$

(b) Der *Aktionswert* von π für einen Zustand $s_t \in \mathcal{S}$ und eine Aktion $a_t \in \mathcal{A}$ ist gegeben durch:

$$\begin{aligned} q_\pi : \mathcal{S} \times \mathcal{A} &\longrightarrow \mathbb{R} \\ (s_t, a_t) &\longmapsto q_\pi(s_t, a_t) \coloneqq \mathbb{E}[\ G_t^\pi \mid S_t = s_t, A_t = a_t] \end{aligned}$$

Bereits an dieser Definition erkennt man einen starken Zusammenhang zwischen Zustands- und Aktionswert, der im nun folgenden Satz genauer untersucht wird.

Satz 2.8 Zusammenhang zwischen Wertfunktionen

Seien $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess mit Abnahmefaktor $\gamma \in (0, 1)$, $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ eine Strategie für M sowie $v_\pi : \mathcal{S} \longrightarrow \mathbb{R}$ bzw. $q_\pi : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ Zustands- bzw. Aktionswert bzgl. π. Dann gelten folgende Zusammenhänge.

(i) Für einen Zustand $s_t \in \mathcal{S}$ gilt:

$$v_\pi(s_t) \quad = \quad q_\pi(s_t, \pi(s_t))$$

(ii) Für ein Paar $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ gilt:

$$q_\pi(s_t, a_t) \quad = \quad \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1} | s_t, a_t)\ v_\pi(s_{t+1})$$

Beweis. (i) Folgt direkt aus den Definitionen.

(ii) Sei $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$. Dann gilt:

$$\begin{aligned}
q_\pi(s_t, a_t) &\coloneqq \mathbb{E}[\ G_t^\pi \mid S_t = s_t, A_t = a_t] \\
&\overset{\text{Lem. 2.6}}{=} \mathbb{E}[\ R_{t+1} + \gamma\ G_{t+1}^\pi \mid S_t = s_t, A_t = a_t] \\
&= \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)\ \mathbb{E}[G_{t+1}^\pi | S_t = s_t, A_t = a_t, S_{t+1} = s_{t+1}] \\
&\overset{\text{Eig. 2.3}}{=} \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)\ \mathbb{E}[\ G_{t+1}^\pi \mid S_{t+1} = s_{t+1}] \\
&= \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t, a_t)\ v_\pi(s_{t+1})
\end{aligned}$$

□

Dieser Zusammenhang wird sich im nächsten Unterkapitel als wichtiger Baustein für die Herleitung eines Kriteriums zur Optimalität von Zustands- bzw. Aktionswerten und damit auch von Strategien erweisen.

2.3 Optimalitätsprinzipien

Der nächste Schritt auf dem Weg zur Entwicklung von optimalen Strategien für Markov-Entscheidungsprozesse wird in diesem Unterkapitel die Einführung von optimalen Zustands- bzw. Aktionswerten sowie darauf aufbauend die Ausarbeitung von Kriterien zur Optimalität von Strategien sein. Hierzu ist es zunächst wichtig, den Begriff der *optimalen Strategie* zu formalisieren, was mithilfe der im vorherigen Unterkapitel eingeführten Wertfunktionen nun möglich ist.

Definition 2.9 Optimale Strategie
Seien $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess, $\pi_* : \mathcal{S} \longrightarrow \mathcal{A}$ eine Strategie für M sowie $v_{\pi_*} : \mathcal{S} \longrightarrow \mathbb{R}$ bzw. $q_{\pi_*} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ Zustands- bzw. Aktionswert bzgl. M.
Dann heiße π_* *optimal*

(a) bezüglich des Zustandswertes, falls gilt:

$$\forall s \in \mathcal{S}\ \ \forall \pi : \mathcal{S} \longrightarrow \mathcal{A} : \quad v_{\pi_*}(s) \geq v_\pi(s)$$

(b) bezüglich des Aktionswertes, falls gilt:

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A}\ \ \forall \pi : \mathcal{S} \longrightarrow \mathcal{A} : \quad q_{\pi_*}(s, a) \geq q_\pi(s, a)$$

Im Gegensatz zum vorherigen Unterkapitel, wo Zustands- und Aktionswert nach dem diskontierten Ertrag bezüglich einer beliebigen – aber fixierten – Strategie bemessen wurden, betrachtet man bei den optimalen Wertfunktionen nun den maximal möglichen, diskontierten Ertrag – ausgehend vom zu bewertenden Zustand bzw. Zustand-Aktions-Paar. Mathematisch lässt sich dies folgendermaßen darstellen:

Definition 2.10 Optimale Wertfunktionen

Sei $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess. Dann ist der *optimale Zustandswert* v_* definiert durch:

$$\begin{aligned} v_* : \quad \mathcal{S} &\longrightarrow \mathbb{R} \\ s &\longmapsto v_*(s) \coloneqq \max_{\pi:\mathcal{S}\to\mathcal{A}} v_\pi(s) \end{aligned}$$

Entsprechend lässt sich der *optimale Aktionswert* q_* definieren:

$$\begin{aligned} q_* : \mathcal{S} \times \mathcal{A} &\longrightarrow \mathbb{R} \\ (s,a) &\longmapsto q_*(s,a) \coloneqq \max_{\pi:\mathcal{S}\to\mathcal{A}} q_\pi(s,a) \end{aligned}$$

Bemerkung 2.11. Ist ein optimaler Aktionswert $q_* : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ bekannt, so ist leicht nachzuprüfen, dass

$$\begin{aligned} \pi_* : \mathcal{S} &\longrightarrow \mathcal{A} \\ s &\longmapsto \pi_*(s) \coloneqq \arg\max_{a\in\mathcal{A}} q_*(s,a) \end{aligned} \tag{2.4}$$

eine optimale Strategie darstellt. Es ist also möglich, das Problem, eine optimale Lösungsstrategie für einen Markov-Entscheidungsprozess zu finden, auf das Problem der Ermittlung eines optimalen Aktionswertes zurückzuführen.

Für das Auffinden eines solchen optimalen Aktionswertes lohnt es sich, erneut den Zusammenhang zwischen Zustands- und Aktionswert zu untersuchen. Ähnlich wie in Satz 2.8 lassen sich nämlich auch Relationen für optimale Wertfunktionen herstellen.

Satz 2.12 Zusammenhang zwischen optimalen Wertfunktionen

Seien $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess mit Abnahmefaktor $\gamma \in (0,1)$ sowie $v_* : \mathcal{S} \longrightarrow \mathbb{R}$ bzw. $q_* : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ optimaler Zustands- bzw. Aktionswert bzgl. M. Dann gelten folgende Zusammenhänge.

(i) Für einen Zustand $s \in \mathcal{S}$ gilt:

$$v_*(s) \quad = \quad \max_{a\in\mathcal{A}} q_*(s,a)$$

(ii) Für ein Paar $(s,a) \in \mathcal{S} \times \mathcal{A}$ gilt:

$$q_*(s,a) \quad = \quad \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ v_*(s')$$

Beweis. (i) Folgt direkt aus den Definitionen.

(ii) Sei $(s,a) \in \mathcal{S} \times \mathcal{A}$. Dann gilt:

$$\begin{aligned} q_*(s,a) &\coloneqq \max_{\pi:\mathcal{S}\to\mathcal{A}} q_\pi(s,a) \\ &\overset{\text{Satz 2.8 (ii)}}{=} \max_{\pi:\mathcal{S}\to\mathcal{A}} \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ v_\pi(s') \\ &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ \max_{\pi:\mathcal{S}\to\mathcal{A}} v_\pi(s') \\ &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ v_*(s') \end{aligned}$$

□

Die Kombination von Teil (i) und (ii) aus Satz 2.12 führt zu folgender wichtiger Eigenschaft:

Korollar 2.13 Bellman-Gleichung
Seien $M \coloneqq (\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ ein Markov-Entscheidungsprozess mit Abnahmefaktor $\gamma \in (0,1)$ sowie $q_* : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ ein optimaler Aktionswert bzgl. M. Dann gilt folgende rekursive Gleichung für q_* und ein beliebiges Zustands-Aktions-Paar $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$q_*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ \max_{a'\in\mathcal{A}} q_*(s',a') \tag{2.5}$$

Beweis. Sei $(s,a) \in \mathcal{S} \times \mathcal{A}$. Dann gilt:

$$\begin{aligned} q_*(s,a) &\overset{\text{Satz 2.12 (ii)}}{=} \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ v_*(s') \\ &\overset{\text{Satz 2.12 (i)}}{=} \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ \max_{a'\in\mathcal{A}} q_*(s',a') \end{aligned}$$

□

Bemerkung 2.14. Analog kann man auch eine rekursive Gleichung für den Zustandswert zeigen:

$$v_*(s) = \max_{a\in\mathcal{A}}\ \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)\ v_*(s') \tag{2.6}$$

Der oben bewiesenen Gleichung liegt das folgende *Optimalitätsprinzip* von Richard Bellman zugrunde, weshalb sie auch *Bellman-Gleichung* genannt wird:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
>
> Eine optimale Strategie hat die Eigenschaft, dass – gleichgültig, welcher Anfangszustand vorliegt und welche Entscheidung in diesem getroffen wird – die übrigen Entscheidungen eine optimale Strategie bezüglich des aus der ersten Entscheidung resultierenden Zustands bilden.
>
> [Bel57, Übersetzung durch den Autor]

Dieses Optimalitätsprinzip bildet die Grundlage für viele Optimierungsalgorithmen – auch außerhalb des Bereichs des maschinellen Lernens – und wird auch im weiteren Verlauf dieser Arbeit die Algorithmen zur Lösung von Markov-Entscheidungsprozessen motivieren.

2.4 Wertiteration

Mithilfe der im vorigen Unterkapitel hergeleiteten Bellman-Gleichung lässt sich nun ein erstes Lösungsverfahren für die Markov-Entscheidungsprozesse einführen. Der Grundgedanke dieses Verfahrens ist es, für jedes Zustands-Aktions-Paar $(s, a) \in \mathcal{S} \times \mathcal{A}$ einen Approximator $Q^*(s, a)$ an den optimalen Zustandswert $q_*(s, a)$ zu finden, indem man die Bellman-Gleichung wiederholt auf die beliebig – aber meist durch null – initialisierten Approximatoren anwendet. Sobald sich dabei die linke Seite der Gleichung

$$Q^*(s,a) \quad = \quad \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \ \max_{a' \in \mathcal{A}} Q^*(s',a') \tag{2.7}$$

für jedes Paar $(s, a) \in \mathcal{S} \times \mathcal{A}$ höchstens um eine vorher festgelegte Differenz $\delta > 0$ von der rechten Seite unterscheidet, kann das Verfahren beendet werden. Zusammengefasst mündet dies im folgenden Algorithmus, der aufgrund seines iterativen Charakters *Wertiteration* genannt wird.

Algorithmus 1: Wertiteration (vergleiche [SB18, Kapitel 4])

Eingabe: Ein Markov-Entscheidungsprozess $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ mit Abnahmefaktor $\gamma \in (0, 1)$ sowie eine Zahl $\delta > 0$.

Ausgabe: Eine Approximation $Q^*(s, a)$ an den optimalen Aktionswert $q_*(s, a)$ für alle $(s, a) \in \mathcal{S} \times \mathcal{A}$.

initialisiere

$Q^{(0)}(s,a) \leftarrow 0$ für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$

$t \leftarrow 0$

wiederhole

$Q^{(t+1)}(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \ \max_{a' \in \mathcal{A}} Q^{(t)}(s',a')$ für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$

$\Delta \leftarrow \max_{(s,a) \in \mathcal{S} \times \mathcal{A}} |Q^{(t+1)}(s,a) - Q^{(t)}(s,a)|$

$t \leftarrow t + 1$

bis $\Delta < \delta$

Bemerkung 2.15. In der Literatur (vergleiche [PS78], [SB18, Kapitel 4], [Wat89, Kapitel 4]) wird die Wertiteration meist auf analoge Weise gemäß der Bellman-Gleichung 2.6 für den Zustandswert betrachtet. Dies ist auch sinnvoll in Anbetracht

der Tatsache, dass bei der Wertiteration für den Zustandswert in jedem Iterationsschritt nur $|\mathcal{S}|$ – im Gegensatz zu $|\mathcal{S}| \cdot |\mathcal{A}|$ für den Aktionswert – Approximatoren neu berechnet werden müssen. Da die Wertiteration hier jedoch nur als eine theoretische Grundlage für die in den nächsten Kapiteln folgenden Algorithmen, bei denen eher die Aktionswerte relevant sind, dienen und nicht tatsächlich implementiert werden soll, wird nachfolgend auch die Wertiteration im Bezug auf den Aktionswert behandelt.

Es lässt sich zeigen, dass die Wertiteration tatsächlich gegen den optimalen Aktionswert konvergiert. Für einen Beweis hierfür sei zum Beispiel auf [PS78] oder [Bel57] verwiesen. Hauptbestandteil der Beweise ist die Erkenntnis, dass es sich beim durch die Bellman-Gleichung motivierten Operator

$$\mathcal{T}(Q^*(s,a)) \quad \coloneqq \quad \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \ \max_{a' \in \mathcal{A}} Q^*(s',a') \tag{2.8}$$

um eine Kontraktion handelt und deshalb die Fixpunktiteration

$$Q^{(t+1)}(s,a) \quad \coloneqq \quad \mathcal{T}(Q^{(t)}(s,a)), \tag{2.9}$$

welche gerade der Iterationsvorschrift aus Algorithmus 1 entspricht, nach dem Fixpunktsatz von Banach für $t \to \infty$ – unter zusätzlichen Voraussetzungen – gegen einen eindeutigen Fixpunkt $Q^*(s,a)$ konvergiert.

Die Wertiteration und verschiedene Abwandlungen davon (z.B. die Strategie-Iteration (vergleiche [SB18, Kapitel 4])) wurden ab Mitte der 1950-er Jahre entwickelt und konnten durch die Implementierung mittels dynamischer Programmierung (siehe [Bel57]) auch erste kleine, durch Markov-Entscheidungsprozesse modellierte Probleme lösen.

Trotz ihrer Konvergenzeigenschaften und ihrer frühen Erfolge, ist die Wertiteration für die Lösung der meisten praktisch relevanten Markov-Entscheidungsprozesse unbrauchbar. Dies hat vor allem zwei Gründe:

Erstens muss für jedes mögliche Zustands-Aktions-Paar ein eigener Approximator berechnet werden, was zum einen aufgrund von möglicherweise starken Ähnlichkeiten zwischen verschiedenen Zuständen und Aktionen ineffizient und zum anderen im Fall von sehr großen Zustands- bzw. Aktionsräumen praktisch unmöglich werden kann. Zweitens hat ein Agent in einem Markov-Entscheidungsprozess in der Regel keine explizite Kenntnis über die in der Wertiteration verwendeten Übergangswahrscheinlichkeiten und kann außerdem nicht, wie bei der Wertiteration erforderlich, systematisch auf jedes einzelne Paar aus dem Zustands-Aktions-Raum zugreifen, sondern muss diesen durch Interaktion mit seiner Umgebung erkunden. In der Fachsprache werden Algorithmen, die, wie die Wertiteration, Wissen über die Hintergründe der Umgebung, d.h. zum Beispiel die genaue Beschaffenheit der Übergangswahrscheinlichkeiten oder der Belohnungsfunktion, voraussetzen, *Modell-basiert* (engl. *model-based*) genannt.

Demgegenüber stehen die *Modell-freien* (engl. *model-free*) Algorithmen, die jenes explizite Wissen nicht erfordern. Ein solcher Algorithmus wird im nächsten Unterkapitel untersucht.

2.5 Q-Lernen

Anders als bei der Wertiteration, die – wie im vorherigen Unterkapitel bereits angeführt – direkten Zugriff auf jedes einzelne Zustands-Aktions-Paar sowie auf die Belohnungsfunktion und die Übergangswahrscheinlichkeiten der Umgebung benötigt und damit in vielen Fällen nicht anwendbar ist, soll eine optimale Strategie nun auf eher natürliche Weise gefunden werden. Ähnlich wie in den Ausführungen zu Beginn der vorliegenden Arbeit muss ein Agent also die Zustände und Zusammenhänge in seiner Umgebung erst durch die Interaktion mit dieser selbst erlernen, um darauf aufbauend Strategien zu entwickeln. Deshalb wird der im folgenden eingeführte Algorithmus auch *Q-Lernen* genannt.

Bezogen auf das Finden des optimalen Aktionswertes im mathematischen Modell bedeutet dies, dass die in der Iterationsvorschrift der Wertiteration verwendeten Ausdrücke $\mathcal{R}(s,a)$ und $\sum_{s'\in\mathcal{S}} p(s'|s,a) \ \max_{a'\in\mathcal{A}} Q^{(t-1)}(s',a')$ ersetzt werden müssen. Im Fall der Belohnungsfunktion ist dies recht unkompliziert möglich, da ein Agent, der in einem Zustand $s_t \in \mathcal{S}$ eine Aktion $a_t \in \mathcal{A}$ ausführt eine Belohnung $r_{t+1} \coloneqq \mathcal{R}(s_t, a_t)$ beobachten kann, auch wenn er die Belohnungsfunktion nicht explizit kennt. Für den zweiten Ausdruck lohnt es sich, den folgenden Zusammenhang zu betrachten:

$$\sum_{s_{t+1}\in\mathcal{S}} p(s_{t+1}|s_t,a_t) \max_{a_{t+1}\in\mathcal{A}} Q^{(t-1)}(s_{t+1},a_{t+1}) \quad = \quad \mathbb{E}_{S_{t+1}\sim p(\cdot|s_t,a_t)}[\max_{a_{t+1}\in\mathcal{A}} Q^{(t-1)}(S_{t+1},a_{t+1})] \tag{2.10}$$

Der zweite Ausdruck ist demnach der Erwartungswert der Größe $\max_{a_{t+1}\in\mathcal{A}} Q^{(t-1)}(S_{t+1},a_{t+1})$, wobei S_{t+1} eine gemäß $p(\cdot|s_t,a_t)$ verteilte Zufallsvariable ist. Anders als bei der Wertiteration kann dieser Erwartungswert nun nicht mehr direkt berechnet werden, da die Übergangswahrscheinlichkeiten dem Agenten nicht bekannt sind. Erneut kann der Agent aber den neuen Zustand $s_{t+1} \in \mathcal{S}$ beobachten, wenn er im Zustand s_t die Aktion a_t ausführt. Hiermit lässt sich dann der Zustandswert $V^{(t-1)}(s_{t+1}) \coloneqq \max_{a_{t+1}\in\mathcal{A}} Q^{(t-1)}(s_{t+1},a_{t+1})$ berechnen. Da s_{t+1} jedoch nur eine einzige Realisierung der Zufallsvariable S_{t+1} ist, wird der alte Approximator $Q^{(t-1)}(s_t,a_t)$ nur zu einem Anteil $\eta \in [0,1)$ neu berechnet und zum Anteil $1-\eta$ beibehalten, sodass sich insgesamt folgende neue Aktualisierungsvorschrift ergibt:

$$Q^{(t)}(s_t,a_t) \quad \coloneqq \quad (1-\eta_t)Q^{(t-1)}(s_t,a_t) \quad + \quad \eta_t(r_{t+1}+\gamma V^{(t-1)}(s_{t+1})) \tag{2.11}$$

In diesem Zusammenhang beschreibt η also, mit welchem Gewicht neue Erfahrungen die Einschätzungen des Agenten beeinflussen, weshalb η auch *Lernrate* genannt wird.

Zusammengefasst ergibt sich damit der folgende Algorithmus:

Algorithmus 2: Q-Lernen (vergleiche [WD92] und [SB18, Kapitel 4])

Eingabe: Ein Markov-Entscheidungsprozess $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ mit Abnahmefaktor $\gamma \in (0,1)$, eine Zahl $T \in \mathbb{N}$ sowie Lernraten $\eta_1, ..., \eta_T \in [0,1)$.

Ausgabe: Eine Approximation $Q^*(s,a)$ an den optimalen Aktionswert $q_*(s,a)$ für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$.

initialisiere

$Q^{(0)}(s,a) \leftarrow 0$ für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$

$s_0 \in \mathcal{S}$

für $t \leftarrow 1$ **bis** T **führe aus**

wähle eine Aktion $a_t \in \mathcal{A}$

erhalte Belohnung $r_{t+1} \in \mathbb{R}$ und neuen Zustand $s_{t+1} \in \mathcal{S}$

berechne den Zustandswert

$$V^{(t-1)}(s_{t+1}) \leftarrow \begin{cases} 0, & \text{falls } s_{t+1} \text{ ein terminaler Zustand ist} \\ \max\limits_{a \in \mathcal{A}} Q^{(t-1)}(s_{t+1}, a) & \text{andernfalls} \end{cases}$$

aktualisiere die Aktionswerte für alle $(s,a) \in \mathcal{S} \times \mathcal{A}$:

$$Q^{(t)}(s,a) \leftarrow \begin{cases} (1-\eta_t) Q^{(t-1)}(s_t, a_t) \; + \; \eta_t (r_{t+1} + \gamma V^{(t-1)}(s_{t+1})), & \text{falls } (s,a) = (s_t, a_t) \\ Q^{(t-1)}(s,a) & \text{andernfalls} \end{cases} \tag{2.12}$$

In [WD92] und [Tsi94] wird gezeigt, dass – unter zusätzlichen Annahmen an die Lernraten – auch beim Q-Lernen die Approximatoren gegen den optimalen Zustandswert konvergieren.

Auch wenn Algorithmus 2 – bei Beibehaltung der Konvergenzeigenschaft – eines der Hauptprobleme der Wertiteration lösen kann, ist auch das Q-Lernen für größere Markov-Entscheidungsprozesse impraktikabel. Dies liegt zum einen daran, dass beim Q-Lernen, genau wie bei der Wertiteration, weiterhin für jedes einzelne Paar $(s,a) \in \mathcal{S} \times \mathcal{A}$ ein eigener Approximator berechnet werden muss. Zum anderen entstehen neue Probleme dadurch, dass die Zustände, die der Agent beobachten und von denen er lernen kann, von seinen eigenen Entscheidungen abhängen.

Diese und weitere Probleme werden im nun folgenden Kapitel mit dem Ziel angegangen, das Q-Lernen für aufwendige, in der Praxis relevante Entscheidungsprozesse anwendbar zu machen.

Kapitel 3

Varianten des tiefen Q-Lernens

Die Beweise für die Konvergenz des Q-Lernens beruhen vor allem darauf, dass während der Durchführung des Verfahrens jedes Paar $(s, a) \in \mathcal{S} \times \mathcal{A}$ mit positiver Wahrscheinlichkeit auftritt, sodass bei hinreichend langer Durchführung des Lernprozesses jedes Zustands-Aktions-Paar auch hinreichend häufig auftritt, um so eine gute Approximation des Aktionswertes für dieses Paar zu erreichen. Problematisch ist hierbei allerdings, dass der Zeithorizont der meisten Entscheidungsprozesse in der Realität endlich ist. Bei sehr großen Zustands- und Aktionsräumen kann dies dazu führen, dass es sehr unwahrscheinlich wird, ein einzelnes Zustands-Aktions-Paar mehr als ein einziges Mal zu beobachten. Dadurch können für die einzelnen Zustands-Aktions-Paare beim klassischen Q-Lernen keine geeigneten Approximatoren mehr gefunden werden.

Im Fall von sehr großen Zustands- und Aktionsräumen wäre es demnach wichtig, Erfahrungen, die für ein Paar $(s, a) \in \mathcal{S} \times \mathcal{A}$ gemacht wurden, verallgemeinern und auf ähnliche Paare $(s', a') \in \mathcal{S} \times \mathcal{A}$ ebenso anwenden zu können. Anstatt also, wie bei der Wertiteration und dem Q-Lernen, für jedes einzelne Element (s, a) aus $\mathcal{S} \times \mathcal{A}$ einen separaten Wert $Q^*(s, a)$ zu finden, ist es nun das Ziel, eine einzige Funktion $Q^* : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ zu ermitteln, die für alle $(s, a) \in \mathcal{S} \times \mathcal{A}$ gleichermaßen den optimalen Aktionswert auf Grundlage der spezifischen Eigenschaften von (s, a) berechnet.

Da die exakte Ermittlung einer solchen Funktion in der Regel nicht realisierbar ist, verwendet man Methoden der Funktionsapproximation für das Finden einer Näherungslösung. Die Möglichkeit, zu diesem Zweck *künstliche neuronale Netzwerke* (siehe Kapitel 3.1) zu gebrauchen, wird bereits in [Wat89, Kapitel 10] vorgeschlagen und in [Tes95] bei einem – dem Q-Lernen ähnlichen – Verfahren des bestärkenden Lernens erfolgreich auf ein komplexes Problem (das Spielen des Brettspiels *Backgammon*) aus der realen Welt angewendet. Trotzdem standen für lange Zeit meist lineare Funktionsapproximatoren im Mittelpunkt der Forschung, da für diese neben dem geringeren Rechenaufwand im Gegensatz zu den nichtlinearen künstlichen neuronalen Netzwerken auch bessere Konvergenzeigenschaften nachgewiesen werden konnten (siehe [PSD01] und [TV97]).

Erst als zu Beginn der 2010er Jahre – begünstigt durch die mittlerweile leistungsfähigere Computer-Hardware – durch Anwendung von künstlichen neuronalen Netzwerken in vielen Bereichen des maschinellen Lernens große Erfolge erreicht wurden (siehe zum Beispiel [GS09] oder [KSH12]), erlebten künstliche neuronale

Netzwerke auch als Funktionsapproximatoren im bestärkenden Lernen einen großen Aufschwung. Durch die Veröffentlichung im einflussreichen Wissenschaftsjournal *Nature* (siehe [Mni+15]) erhielt hierbei das sog. tiefe Q-Lernen, das tiefe neuronale Netzwerke als Funktionsapproximatoren im Q-Lernen einsetzt, große Aufmerksamkeit und Resonanz, sodass in der Folge viele Erweiterungen und Varianten dieses Verfahrens entwickelt wurden. Einige davon werden in diesem Kapitel genauer betrachtet.

3.1 Tiefes Q-Lernen

Da künstliche neuronale Netzwerke ein wesentliches Element des tiefen Q-Lernens darstellen, folgt nun zunächst eine Einführung, die sich größtenteils an [Gri+18] orientiert, in ihre grundlegenden Bestandteile und Funktionsweisen, um darauf aufbauend das Verfahren und verschiedene Erweiterungen des tiefen Q-Lernens zu beleuchten.

Künstliche neuronale Netzwerke

Wie das natürliche Vorbild setzen sich auch künstliche neuronale Netzwerke aus Neuronen zusammen, die Informationen erhalten, verarbeiten und an andere Neuronen weitergeben können. Sind die Neuronen hierbei in Schichten so angeordnet, dass Informationen nur an die nächst tiefere Schicht weitergegeben werden, spricht man auch von *Feedforward*-Netzwerken. In diesem Fall lässt sich das neuronale Netzwerk, ähnlich wie in Abbildung 3.1 dargestellt, als gerichteter azyklischer Graph interpretieren und wird in Anlehnung an [Ros58] manchmal auch Perzeptron genannt. Hierbei nennt man die Schicht der ersten Informationsaufnahme *Eingabeschicht* (engl. *input layer*) und die der letzten Informationsabgabe *Ausgabeschicht* (engl. *output layer*). Alle dazwischenliegenden Schichten werden als *verdeckte Schichten* (engl. *hidden layer*) bezeichnet. Informationen werden also in der Eingabeschicht aufgenommen, durch die verdeckten Schichten weitergeleitet und verarbeitet und schließlich in der Ausgabeschicht abgegeben.

Im mathematischen Modell nimmt hierbei die Eingabeschicht einen Vektor $\boldsymbol{x} \coloneqq (x_1, \dots, x_{d_1}) \in \mathbb{R}^{d_1}$ auf, wobei $d_1 \in \mathbb{N}$ die Anzahl der Neuronen in der Eingabeschicht ist. Die in Eingabeneuron $i \in \{1, \dots, d_1\}$ aufgenommene Information x_i wird anschließend, multipliziert mit einem Gewicht $w_{i,j}^{(1)}$, an das Neuron $j \in \{1, \dots, d_2\}$ der ersten verdeckten Schicht weitergegeben. Für jedes Neuron $j \in \{1, \dots, d_2\}$ der ersten verdeckten Schicht werden dann alle diese Informationen gesammelt und zusammen mit dem Bias $b_j^{(2)} \in \mathbb{R}$ zur sogenannten Netzeingabe $net_j^{(2)} \in \mathbb{R}$ summiert:

$$net_j^{(2)} \coloneqq \sum_{i \in \{1,\dots,d_1\}} w_{i,j}^{(1)} \; x_i \; + \; b_j^{(2)} \tag{3.1}$$

Durch Anwendung einer *Aktivierungsfunktion*[1] $\phi^{(2)} : \mathbb{R} \longrightarrow \mathbb{R}$ auf $net_j^{(2)}$ wird die Ausgabe $o_j^{(2)} \in \mathbb{R}$ der ersten verdeckten Schicht berechnet:

$$o_j^{(2)} \coloneqq \phi^{(2)}(net_j^{(2)}) \tag{3.2}$$

[1] Eine der geläufigsten Aktivierungsfunktionen ist zum Beispiel die sog. ReLU-Aktivierung (für engl. *rectified linear unit*) $\phi_{ReLU}(x) \coloneqq \max\{x, 0\}$

Wird dieses Vorgehen bis zur Ausgabeschicht fortgesetzt, spricht man auch von der *Vorwärts-Propagierung*, die nun abschließend in Abbildung 3.1 schematisch dargestellt und in Definition 3.1 in der etwas kompakteren Matrix-Vektor-Schreibweise zusammengefasst wird.

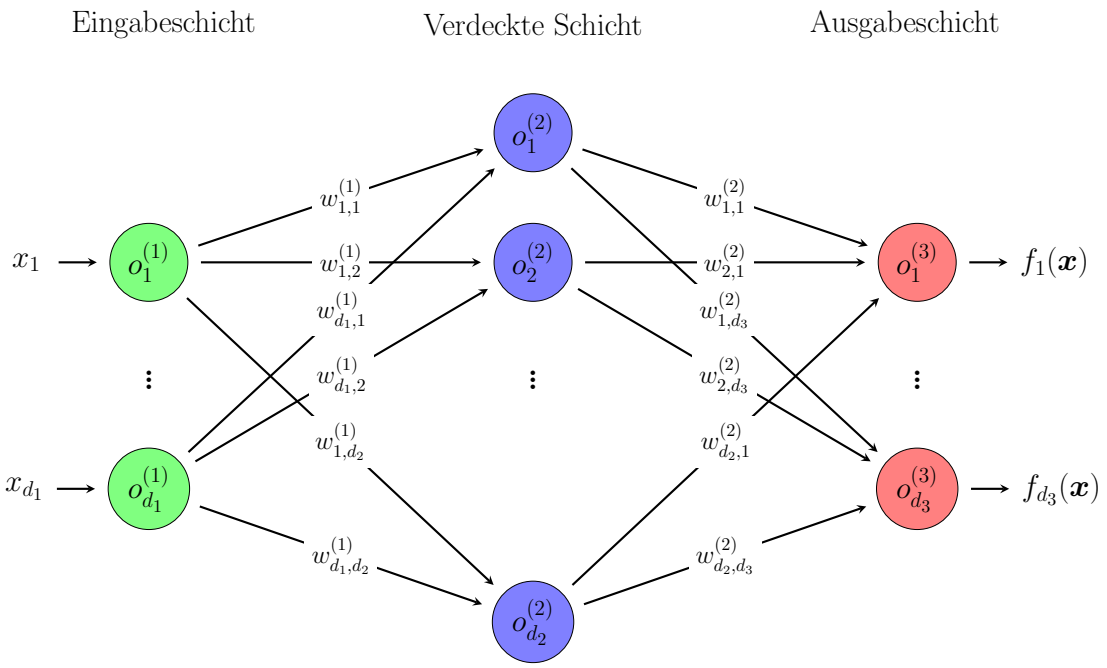


Abbildung 3.1: Struktur eines zweischichtigen Feedforward-Netzwerks mit einer verdeckten Schicht

Definition 3.1 Feedforward-Netzwerk und Vorwärts-Propagierung
Ein *Feedforward-Netzwerk* mit $L \in \mathbb{N}$ Schichten[a] und Dimensionen $d_1 \in \mathbb{N}, \ldots, d_{L+1} \in \mathbb{N}$ ist definiert durch:

- Gewichtungsmatrizen $W^{(1)} \in \mathbb{R}^{d_1 \times d_2}, \ldots, W^{(L)} \in \mathbb{R}^{d_L \times d_{L+1}}$,
- Aktivierungsfunktionen $\phi^{(2)} : \mathbb{R} \longrightarrow \mathbb{R}, \ldots, \phi^{(L+1)} : \mathbb{R} \longrightarrow \mathbb{R}$
- und Bias-Vektoren $\boldsymbol{b}^{(2)} \in \mathbb{R}^{d_2}, \ldots, \boldsymbol{b}^{(L+1)} \in \mathbb{R}^{d_{L+1}}$

Die *Vorwärts-Propagierung* $f : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{d_L}$ durch ein Feedforward-Netzwerk für einen Eingabevektor $\boldsymbol{x} \in \mathbb{R}^{d_1}$ lässt sich folgendermaßen berechnen:

$$
\begin{aligned}
\boldsymbol{o}^{(1)} &:= \boldsymbol{x} \\
\boldsymbol{net}^{(l)} &:= (W^{(l-1)})^T \ \boldsymbol{o}^{(l-1)} + \boldsymbol{b}^{(l)} \quad \text{für } l \in \{2, \ldots, L+1\} \\
\boldsymbol{o}^{(l)} &:= \phi^{(l)}(\boldsymbol{net}^{(l)}) \quad \text{komponentenweise für } l \in \{2, \ldots, L+1\} \\
f(\boldsymbol{x}) &:= \boldsymbol{o}^{(L+1)}
\end{aligned}
$$

[a] Ist $L > 2$ spricht man auch von einem *tiefen* Feedforward-Netzwerk

Feedforward-Netzwerke werden typischerweise für die Lösung von Problemen des

überwachten Lernens verwendet. Hierbei ist ein Datensatz[2]

$$\mathcal{D} \coloneqq \{(\boldsymbol{x}_i, y_i) \in \Omega \times \Gamma \mid i \in \{1, \dots, n\}\} \tag{3.3}$$

gegeben, der aus $n \in \mathbb{N}$ unabhängigen und identisch – gemäß einer Verteilung μ auf $\Omega \times \Gamma$ – verteilten Ziehungen besteht[3]. Ziel ist es dann, mithilfe der Informationen aus diesem Datensatz eine Funktion $f : \Omega \longrightarrow \Gamma$ zu finden, sodass näherungsweise gilt:

$$f(\boldsymbol{x}) \quad \approx \quad y \qquad \text{für ein beliebiges Paar } (\boldsymbol{x}, y) \sim \mu \tag{3.4}$$

Möchte man zu diesem Zweck ein Feedforward-Netzwerk benutzen, so ist es also das Ziel, die Gewichtungsmatrizen und Bias-Vektoren so anzupassen, dass Eigenschaft 3.4 für die Vorwärts-Propagierung näherungsweise erfüllt wird. Hierzu betrachtet man zunächst eine Fehlerfunktion, die die Güte dieser Näherung bemisst. Meist wird dabei die mittlere quadratische Abweichung verwendet:

$$\mathcal{J}_{\mathcal{D}}(f) \quad \coloneqq \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathcal{C}(f(\boldsymbol{x}), y) \quad \coloneqq \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} (f(\boldsymbol{x}) - y)^2 \tag{3.5}$$

Um diesen Ausdruck zu minimieren werden meist Gradienten-basierte Algorithmen (siehe Algorithmus 3) verwendet. Für diese ist es nötig, die Gradienten $\nabla_{W^{(l)}} \mathcal{J}_{\mathcal{D}}(f)$ bzw. $\nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{J}_{\mathcal{D}}(f)$ für $l \in \{1, \dots, L\}$ bestimmen zu können. Durch Anwendung der Kettenregel lässt sich folgende Methode zur Berechnung der Gradienten in einem neuronalen Netzwerk herleiten, die aufgrund der Tatsache, dass die Gradienten iterativ von der Ausgabeschicht zur Eingabeschicht hin berechnet werden, auch *Rückpropagierung* genannt wird (vgl. [RHW86]).

Satz 3.2 Rückpropagierung

Sei $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ die Vorwärts-Propagierung für ein Feedforward-Netzwerk gemäß Definition 3.1 mit $d_1 \coloneqq d$ und $d_{L+1} \coloneqq 1$. Dann lassen sich die Gradienten $\nabla_{W^{(l)}} \mathcal{C}(f(\boldsymbol{x}), y)$ bzw. $\nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{C}(f(\boldsymbol{x}), y)$ für $\mathcal{C}(f(\boldsymbol{x}), y) \coloneqq (f(\boldsymbol{x}) - y)^2$ wie folgt berechnen:

$$\begin{aligned} \delta^{(L)} \quad &\coloneqq \quad 2\ (f(\boldsymbol{x}) - y) \\ \delta^{(l)} \quad &\coloneqq \quad W^{(l+1)}\ \cdot\ (\delta^{(l+1)}\ \odot\ (\phi^{(l+2)})'(\boldsymbol{net}^{(l+2)})) \quad \text{für } l = L-1, \dots, 1 \end{aligned}$$

Für alle $l \in \{1, \dots, L\}$ gilt damit:

$$\begin{aligned} \nabla_{W^{(l)}} \mathcal{C}(f(\boldsymbol{x}), y) \quad &= \quad \boldsymbol{o}^{(l)}\ \cdot\ (\delta^{(l)}\ \odot\ (\phi^{(l+1)})'(\boldsymbol{net}^{(l+1)}))^T \\ \nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{C}(f(\boldsymbol{x}), y) \quad &= \quad \delta^{(l)}\ \odot\ (\phi^{(l+1)})'(\boldsymbol{net}^{(l+1)}) \end{aligned}$$

Dabei ist $\odot$ der Operator für das Hadamard-Produkt.

[2] In diesem Zusammenhang wird $\boldsymbol{x_i}$ oft *Feature-Vektor* oder auch *erklärende Variable* und y_i *Zielvariable* bzw. *erklärte Variable* genannt.

[3] Der Einfachheit halber wird hier $\Omega \subseteq \mathbb{R}^d$ sowie $\Gamma \subseteq \mathbb{R}$ angenommen, wobei im Allgemeinen auch Γ einem höherdimensionalen Raum entstammen kann.

Mit diesem Wissen lassen sich nun auch die Gradienten des Fehlers für den ganzen Datensatz in jeder Schicht $l \in \{1, \dots, L\}$ berechnen:

$$\nabla_{W^{(l)}} \mathcal{J}_{\mathcal{D}}(f) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \nabla_{W^{(l)}} \mathcal{C}(f(\boldsymbol{x}), y) \tag{3.6}$$

$$\nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{J}_{\mathcal{D}}(f) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{C}(f(\boldsymbol{x}), y) \tag{3.7}$$

Die Grundidee der Minimierung von $\mathcal{J}_{\mathcal{D}}(f)$ durch Gradienten-basierte Verfahren ist es, die Parameter schrittweise durch Bewegung um eine Schrittweite bzw. Lernrate $\eta > 0$ in Richtung des jeweiligen Gradienten zu aktualisieren. Da die Berechnung der Gradienten für den ganzen Datensatz in jedem Schritt im Normalfall zu aufwendig ist, berechnet man meist nur die Gradienten für eine zufällige Teilmenge $\mathcal{B} \subset \mathcal{D}$ der Größe $K \in \mathbb{N}$, die auch *Mini-Batch* genannt wird. Zusammengefasst führt dies zu folgendem Aktualisierungsschritt:

Algorithmus 3: Ein Schritt des stochastischen Mini-Batch-Gradienten-Abstiegs

Eingabe: Ein Datensatz $\mathcal{D} \coloneqq \{(\boldsymbol{x}_i, y_i) \in \Omega \times \Gamma \mid i \in \{1, \dots, n\}\}$, eine Batch-Größe $K < n$, eine Lernrate $\eta > 0$ sowie Gewichtungsmatrizen $W^{(1)}, \dots, W^{(L)}$ und Bias-Vektoren $\boldsymbol{b}^{(2)}, \dots, \boldsymbol{b}^{(L+1)}$ eines Feedforward-Netzwerkes mit Vorwärts-Propagierung f gemäß Definition 3.1.

Ausgabe: Aktualisierte Gewichtungsmatrizen $W^{(1)}, \dots, W^{(L)}$ und Bias-Vektoren $\boldsymbol{b}^{(2)}, \dots, \boldsymbol{b}^{(L+1)}$.

Ziehe eine zufällige Stichprobe $\mathcal{B} \subset \mathcal{D}$ der Größe K.

Berechne $f(\boldsymbol{x})$ durch Vorwärts-Propagierung für alle $(\boldsymbol{x}, y) \in B$.

Berechne $\nabla_{W^{(l)}} \mathcal{J}_{\mathcal{B}}(f)$ und $\nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{J}_{\mathcal{B}}(f)$ für alle $l \in \{1, \dots, L\}$ durch Rückpropagierung.

Aktualisiere die Gewichtungsmatrizen und Bias-Vektoren für alle $l \in \{1, \dots, L\}$ gemäß:

$$W^{(l)} \leftarrow W^{(l)} - \eta \, \nabla_{W^{(l)}} \mathcal{J}_{\mathcal{B}}(f)$$

$$\boldsymbol{b}^{(l+1)} \leftarrow \boldsymbol{b}^{(l+1)} - \eta \, \nabla_{\boldsymbol{b}^{(l+1)}} \mathcal{J}_{\mathcal{B}}(f)$$

Tiefes Q-Lernen

Der nächste Schritt ist nun, die oben eingeführte Theorie der tiefen neuronalen Netzwerke mit der Theorie des Q-Lernens aus Kapitel 2.5 zum *tiefen Q-Lernen*, das in [Mni+13] zuerst eingeführt wurde, zu verbinden. Hierzu betrachtet man den

Aktionswert als parametrische Funktion

$$\begin{aligned} Q_{\boldsymbol{\theta}} : \quad & \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R} \\ & (s, a) \longmapsto Q_{\boldsymbol{\theta}}(s, a), \end{aligned} \tag{3.8}$$

wobei zur besseren Übersichtlichkeit $\boldsymbol{\theta}$ die Gewichtungsmatrizen $W^{(1)}, \ldots, W^{(L)}$ und Bias-Vektoren $\boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(L+1)}$ eines tiefen Feedforward-Netzwerks gemäß Definition 3.1 zusammenfasst, d.h.

$$\boldsymbol{\theta} \quad := \quad (W^{(1)}, \ldots, W^{(L)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(L+1)}). \tag{3.9}$$

$Q_{\boldsymbol{\theta}}(s, a)$ wird in diesem Fall auch *tiefes Q-Netzwerk* (engl. *Deep Q-Network*, kurz DQN) genannt und kann entsprechend durch Vorwärts-Propagierung berechnet werden.

Bemerkung 3.3. Auch wenn der Aktionswert hier als Funktion $Q_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ dargestellt ist, werden in praktischen Anwendungen meist künstliche neuronale Netzwerke betrachtet, die für jede Aktion $a \in \mathcal{A}$ ein eigenes Ausgabeneuron besitzen. Insbesondere wird als Eingabe für das neuronale Netzwerk nur der Zustand verwendet, sodass bei einer Vorwärts-Propagierung eines Zustands $s \in \mathcal{S}$ für alle $a \in \mathcal{A}$ gleichzeitig der Zustandswert $Q_{\boldsymbol{\theta}}(s, a)$ berechnet wird. Zudem werden in der Praxis bei Anwendungen, die auf visuellen Wahrnehmungen, wie zum Beispiel den Pixeln eines Bildes, beruhen, meist sog. *faltende neuronale Netzwerke* (engl. *convolutional neural networks*) verwendet (siehe z.B. [LB+95] für eine genauere Beschreibung). Diese lassen sich als eine Erweiterung der gewöhnlichen Feedforward-Netzwerke auffassen, die mehrdimensionale Schichten sowie unter Neuronen geteilte Gewichte erlaubt.

Im Gegensatz zum überwachten Lernen, für das künstliche neuronale Netzwerke ursprünglich konzipiert sind, ist jedoch beim bestärkenden Lernen nicht im Vorhinein ein Datensatz mit erklärenden und Zielvariablen gegeben. Stattdessen ist der Ansatz beim tiefen Q-Lernen, dass der Agent sich diesen Datensatz selbst erarbeitet. Als erklärende Variablen dienen in diesem Fall die vom Agenten erreichten Zustände zusammen mit den in diesen Zuständen getätigten Aktionen. Die Zielvariablen hingegen muss der Agent selbst berechnen. Hierzu verwendet man erneut die rechte Seite der Bellman-Gleichung 2.5 für den Aktionswert, wobei, genau wie beim Q-Lernen, auf die expliziten Übergangswahrscheinlichkeiten und die Belohnungsfunktion verzichtet wird. Befindet sich nun ein Agent in einem Zustand $s \in \mathcal{S}$, trifft eine Entscheidung $a \in \mathcal{A}$ und beobachtet eine Belohnung $r \in \mathbb{R}$ und einen neuen Zustand $s' \in \mathcal{S}$, so erhält er folgendes Paar aus erklärender und Zielvariable:

$$(\boldsymbol{x}, y) \quad := \quad \big(\ (s, a) \ , \quad r \ + \ \gamma \ \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s', a') \ \big) \tag{3.10}$$

Der Fehler gemäß Gleichung 3.5 für einen Datensatz $\mathcal{D}$, bestehend aus derartigen Erfahrungen des Agenten, lässt sich also folgendermaßen berechnen:

$$\mathcal{J}_{\mathcal{D}}(Q_{\boldsymbol{\theta}}) \quad = \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} \mathcal{C}(Q_{\boldsymbol{\theta}}(\boldsymbol{x}), y) \quad = \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} (Q_{\boldsymbol{\theta}}(\boldsymbol{x}) \ - \ y)^2 \tag{3.11}$$

Analog zum allgemeinen Fall der Feedforward-Netzwerke kann auch bei den tiefen Q-Netzwerken der Gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}})$ für eine Teilmenge $\mathcal{B} \subset \mathcal{D}$ der Größe $K \in \mathbb{N}$ per

Rückpropagierung berechnet werden. Damit ergibt sich, ähnlich wie in Algorithmus 3, mit einer Lernrate $\eta > 0$ eine Aktualisierungsvorschrift für die Parameter $\boldsymbol{\theta}$ des tiefen Q-Netzwerks:

$$\boldsymbol{\theta} \quad \leftarrow \quad \boldsymbol{\theta} \; - \; \eta \; \nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}}) \tag{3.12}$$

Ersetzt man nun im klassischen Algorithmus 2 die Aktualisierungsvorschrift 2.12 durch die neue Vorschrift 3.12 mit

$$\mathcal{B} \quad \coloneqq \quad \{((s_t, a_t) \;\; , \quad r_{t+1} \;\; + \;\; \gamma \;\; \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s_{t+1}, a'))\}, \tag{3.13}$$

so erhält man eine erste einfache Form des tiefen Q-Lernens. Durch die Aktualisierung mit Mini-Batches der Art 3.13, die nur ein einziges Element – nämlich die gerade gemachte Erfahrung des Agenten – enthalten, entstehen jedoch große Instabilitäten beim Lernprozess.

Dies liegt zum einen daran, dass ein Mini-Batch in dieser Form nur aus einem einzigen Element besteht, was aufgrund des fehlenden ausgleichenden Effektes größerer Mini-Batches zu stark schwankenden Gradienten führen kann. Eine naheliegende Lösung hierfür wäre es, eine vorher festgelegte Anzahl $K \in \mathbb{N}$ der letzten Erfahrungen zu speichern und auf dieser Grundlage die Mini-Batch-Aktualisierungen durchzuführen. Doch auch dann liefert diese einfache Form des tiefen Q-Lernens oftmals keine guten Ergebnisse (siehe Kapitel 4).

Denn zum anderen gibt es aufgrund des sequentiellen Charakters von Problemen des bestärkenden Lernens oftmals große Korrelationen zwischen aufeinander folgenden Erfahrungen, was die Annahme der Unabhängigkeit der Elemente des Datensatzes bei Problemen des überwachten Lernens verletzt. Eine Möglichkeit, diese Korrelationen zu einem gewissen Grad aufzulösen, wird im folgenden Unterkapitel vorgestellt.

3.2 (Priorisierte) Erfahrungswiederholung

Um die im vorherigen Unterkapitel angeführten Instabilitäten bei der Parameteraktualisierung mit stark korrelierten aufeinander folgenden Erfahrungen des Agenten zu überwinden, orientiert man sich am für die natürlichen Lernprozesse von Lebewesen wichtigen Langzeitgedächtnis. Nach [KT12] sind beim Langzeitgedächtnis die folgenden Prozesse zu beobachten:

- *Enkodierung*, d.h. die Repräsentation und Speicherung von Wahrnehmungen im Gedächtnis
- *Konsolidierung*, d.h. die Festigung von Erinnerungen durch wiederholtes Abrufen
- *Vergessen*, d.h. der Verlust oder die Überschreibung von alten Erinnerungen

Erfahrungswiederholung

Im mathematischen Modell werden nun die Wahrnehmungen $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, $r_{t+1} \in \mathbb{R}$ und $s_{t+1} \in \mathcal{S}$ eines Agenten während eines Lernprozesses zum Zeitpunkt $t \in \mathbb{N}$

enkodiert, indem das Tupel $(s_t, a_t, r_{t+1}, s_{t+1})$ einer Menge $\mathcal{D}$ hinzugefügt wird[4]. Ein Element dieser Menge $\mathcal{D}$ bezeichnet man in diesem Zusammenhang auch als Erfahrung.

Der Prozess des Vergessens wird modelliert, indem $|\mathcal{D}|$ auf eine maximale Kapazität von $N \in \mathbb{N}$ beschränkt und jede Erfahrung, die weiter als N Schritte zurückliegt, aus der Menge $\mathcal{D}$ entfernt wird. Insgesamt besteht das Gedächtnis des Agenten zum Zeitpunkt t also aus folgender Menge an Erfahrungen

$$\mathcal{D}_t \quad \coloneqq \quad \{(s_{t_0}, a_{t_0}, r_{t_0+1}, s_{t_0+1}), \ldots, (s_t, a_t, r_{t+1}, s_{t+1})\}, \tag{3.14}$$

wobei $t_0 \coloneqq \max\{0, t - N\}$ der früheste Zeitpunkt ist, an den sich der Agent erinnern kann.

Für die Konsolidierung wird das Abrufen von Erinnerungen durch das zufällige Ziehen einer Menge von Indizes $\mathcal{I} \subset \{t_0, \ldots, t\}$ realisiert. Daraufhin werden die zu diesen Indizes $i \in \mathcal{I}$ gehörenden Erfahrungen $(s_i, a_i, r_{i+1}, s_{i+1})$ aus der Menge $\mathcal{D}$ entnommen und die entsprechenden Paare aus erklärenden und Zielvariablen $(\boldsymbol{x}_i, y_i)$ gemäß 3.10 berechnet, sodass folgender Mini-Batch entsteht:

$$\begin{aligned} \mathcal{B} \quad \coloneqq \quad \{(\boldsymbol{x}_i, y_i) \mid\ & i \in \mathcal{I}, \quad (s_i, a_i, r_{i+1}, s_{i+1}) \in \mathcal{D}_t, \\ & \boldsymbol{x}_i \ \coloneqq \ (s_i, a_i), \\ & y_i \ \coloneqq \ r_{i+1} \ + \ \gamma \ \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s_{i+1}, a')\} \end{aligned} \tag{3.15}$$

Mit diesem Mini-Batch kann nun eine Parameteraktualisierung in Anlehnung an 3.12 vollzogen werden, um dadurch die Erinnerungen zu verfestigen.

Der gerade beschriebene Prozess wird im folgenden auch *Erfahrungswiederholung* (engl. *experience replay*) genannt. In den Experimenten in Kapitel 4 wird sich zeigen, dass dieser Prozess durch die zeitliche Trennung vom Erleben einer Erfahrung und dem Lernen aus dieser Erfahrung – und die damit erreichte geringere Korrelation der Elemente des Mini-Batches – essenziell für die Funktionsfähigkeit und Anwendbarkeit des tiefen Q-Lernens bei komplexen Problemen sein kann.

Priorisierte Erfahrungswiederholung

Eine Möglichkeit, dieses Verfahren noch zu verfeinern, besteht darin, die Auswahl der Erfahrungen für den Aktualisierungsschritt nicht mehr rein zufällig zu vollziehen. Stattdessen versucht man Erfahrungen, die einen großen Lernfortschritt versprechen, mit höherer Wahrscheinlichkeit auszuwählen. In diesem Fall nennt man das Verfahren, das unter anderem in [Lin92] eingeführt und in [Sch+15] auf das tiefe Q-Lernen angewendet wird, auch *priorisierte Erfahrungswiederholung* (engl. *prioritized experience replay*). Hierbei werden die Elemente der Menge $\mathcal{D}_t$ mit Prioritäten $p_i \in \mathbb{R}_{>0}$ für $i \in \{t_0, \ldots, t\}$ versehen. Meist verwendet man als Priorität für eine Erfahrung $(s_i, a_i, r_{i+1}, s_{i+1}) \in \mathcal{D}_t$ die Kosten $\mathcal{C}(\boldsymbol{x}_i, y_i)$, wobei $(\boldsymbol{x}_i, y_i)$ erneut gemäß 3.10 bestimmt wird. Um jeder Erfahrung eine positive Priorität zu garantieren, wird noch ein Wert $\epsilon > 0$ addiert, sodass sich insgesamt folgende Priorität ergibt:

$$p_i \quad \coloneqq \quad \mathcal{C}(Q_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i) + \epsilon \tag{3.16}$$

[4]Beachte den Unterschied zum vorherigen Unterkapitel, in dem $\mathcal{D}$ als eine Menge von Paaren von erklärender und Zielvariable der Form 3.10 betrachtet wurde.

Die Prioritäten sind also bis auf ϵ proportional zum Fehler des Paares $(\boldsymbol{x}_i, y_i)$, weshalb man in diesem Zusammenhang von *proportionaler Priorisieung* spricht. Eine eng damit verwandte Variante ist die sog. *Rang-basierte Priorisierung*, bei der Erfahrungen gemäß ihres Ranges in einer bezüglich des Fehlers sortierten Liste aller Erfahrungen priorisiert werden, d.h.

$$p_i \quad \coloneqq \quad \frac{1}{rang(i)}, \tag{3.17}$$

wobei

$$rang(i) \quad \coloneqq \quad |\{\ j \in \{t_0, \ldots, t\}\ \mid\ \mathcal{C}(Q_{\boldsymbol{\theta}}(\boldsymbol{x}_j), y_j)\ \ \geq\ \ \mathcal{C}(Q_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)\}|. \tag{3.18}$$

In beiden Fällen wird ein Index i dann mit Wahrscheinlichkeit

$$\mathcal{P}(i) \quad \coloneqq \quad \frac{p_i^\alpha}{\sum_{j=t_0}^{t} p_j^\alpha} \tag{3.19}$$

ausgewählt, wobei $\alpha \in [0, 1]$ ein Koeffizient ist, der den Grad der Priorisierung steuert. Für $\alpha = 0$ entspricht $\mathcal{P}$ zum Beispiel gerade der Wahrscheinlichkeitsfunktion einer Gleichverteilung auf der Menge $\{t_0, \ldots, t\}$, während die Wahrscheinlichkeiten im Fall $\alpha = 1$ proportional zu den jeweiligen Prioritäten sind.

Ein Problem, das bei der priorisierten Erfahrungswiederholung auftreten kann, betrifft die Aktualisierungsschritte des stochastischen Mini-Batch-Gradienten-Abstiegs. Wie bereits in Kapitel 3.1 angeführt, ist hierbei der eigentliche Sinn der Mini-Batch-Aktualisierungen, den Gradienten des Fehlers bezüglich des ganzen Datensatzes $\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{D}_t}(Q_{\boldsymbol{\theta}})$ durch die Gradienten $\nabla_{\boldsymbol{\theta}} \mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}})$ bezüglich zufälliger Teilmengen $\mathcal{B} \subset \mathcal{D}_t$ anzunähern. Durch die priorisierte Ziehung der Teilmengen $\mathcal{B}$ entsteht dabei nun eine Verzerrung bei den Parameteraktualisierungen. Obwohl diese Verzerrung aufgrund des dadurch oftmals erreichten schnelleren Lernfortschritts teilweise erwünscht ist, kann sie vor allem am Ende des Lernprozesses zu schlechteren Ergebnissen führen (vgl. [Sch+15, Figure 12]). Um den Effekt der Verzerrung zu kompensieren, werden bei der Berechnung des Fehlerterms $\mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}})$ die Kosten $\mathcal{C}(Q_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ eines Paares $(\boldsymbol{x}_i, y_i) \in \mathcal{B}$ mit einem Gewicht $w_i \in [0, 1]$ versehen, das bei steigender Priorität p_i geringer wird. Insgesamt entsteht damit folgender Fehlerterm für die priorisierte Erfahrungswiederholung:

$$\mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}}) \quad = \quad \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{B}} w_i \cdot \mathcal{C}\left(Q_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i\right) \tag{3.20}$$

Die Gewichte w_i werden dabei folgendermaßen berechnet:

$$w_i \quad \coloneqq \quad \left(\frac{1}{|\mathcal{D}_t| \cdot P(i)}\right)^\beta \tag{3.21}$$

Anschließend werden die Gewichte noch mit $(\max_{j \in \mathcal{I}} w_j)^{-1}$ skaliert, um $w_i \in [0, 1]$ sicherzustellen. Ähnlich wie zuvor α, ist $\beta \in [0, 1]$ ein Koeffizient der den Grad der Verzerrungskompensierung steuert. Um sowohl den oben beschriebenen Effekt des schnelleren Lernfortschritts als auch den des besseren Endergebnisses auszunutzen, verwendet man für β meist einen relativ geringen Startwert und nähert diesen im Verlauf des Lernprozesses dem Wert 1 an.

Weitere interessante Ansätze, deren genauere Untersuchung den Rahmen dieser Arbeit jedoch überschreitet, könnten darin bestehen, nicht nur den Prozess der Konsolidierung sondern auch die Prozesse der Enkodierung und des Vergessens in priorisierter Form zu modellieren. Indem man also Erfahrungen, von denen kein großer Lernfortschritt mehr zu erwarten ist, vorzeitig aus dem Wiederholungsspeicher entfernt bzw. solche Erfahrungen erst gar nicht aufnimmt, könnte man möglicherweise den – im Normalfall recht hohen – Speicherbedarf der Erfahrungswiederholung deutlich verringern.

3.3 Ziel- und Strategienetzwerk

Instabilitäten beim tiefen Q-Lernen können nicht nur aufgrund der im vorherigen Unterkapitel behandelten Korrelationen zwischen aufeinander folgenden Erfahrungen auftreten, sondern auch aufgrund der Art und Weise, wie die Zielvariablen für die Parameteraktualisierungen berechnet werden. Betrachtet man erneut die Berechnung des Zielwerts y für eine Erfahrung gemäß 3.10, so fällt auf, dass y selbst von den Parametern $\boldsymbol{\theta}$ abhängt. Diese Abhängigkeit kann zu Oszillationen der Parameterwerte führen.

Eine erste Möglichkeit diese Abhängigkeit zu mildern wird in [Mni+15] eingeführt. Der Grundgedanke des dort beschriebenen Ansatzes ist es, separate Parameter $\widetilde{\boldsymbol{\theta}}$ für die Berechnung des Zielwertes einer Erfahrung (s, a, r, s') bei den Parameteraktualisierungen zu verwenden, d.h.

$$y \quad \coloneqq \quad r \; + \; \gamma \; \max_{a' \in \mathcal{A}} Q_{\widetilde{\boldsymbol{\theta}}}(s', a'). \tag{3.22}$$

Die Parameter $\widetilde{\boldsymbol{\theta}}$ sind hierbei zunächst eine Kopie der ursprünglichen Parameter $\boldsymbol{\theta}$ – insbesondere beschreiben die Parameter $\widetilde{\boldsymbol{\theta}}$ dieselbe Netzwerkarchitektur wie $\boldsymbol{\theta}$ – , bleiben anschließend jedoch für einen Zeitraum von $\tau \in \mathbb{N}$ Schritten unverändert und werden erst danach erneut mit den aktuellen Parametern $\boldsymbol{\theta}$ überschrieben. In diesem Zusammenhang wird $Q_{\widetilde{\boldsymbol{\theta}}}$ auch *Zielnetzwerk* (engl. *target network*) und $Q_{\boldsymbol{\theta}}$ entsprechend *Strategienetzwerk* (engl. *policy network*) genannt.

Ein anderes verbreitetes Problem bei der Berechnung der Zielwerte im bestärkenden Lernen mit Funktionsapproximation im Allgemeinen und im tiefen Q-Lernen im Speziellen ist eine systematische Überbewertung der Aktionswerte (vgl. [TS93]). Hauptgrund hierfür ist die Verwendung des Maximum-Operators bei der Berechnung des Zielwerts gemäß 3.10 bzw. 3.22. Zufällige zunächst kleine Fehler – zum Beispiel aufgrund der Initialisierung der Netzwerkparameter – bei den Aktionswerten können so dazu führen, dass Aktionen mit überschätzten Aktionswerten mit höherer Wahrscheinlichkeit für die Berechnung des Zielwertes verwendet werden, was im Verlauf des Lernprozesses zu einer Verstärkung dieser Überschätzungen führen kann.

Eine mögliche Lösung dieses Problems, deren Ansatz in einer weiteren Trennung der Auswahl der Aktionen a' in Gleichung 3.22 und der Bewertung des entsprechenden Aktionswertes liegt, wird in [VGS16] vorgestellt. Bei der Realisierung dieses Lösungsansatzes bietet sich für die Auswahl der Aktionen das Strategienetzwerk und für die Bewertung der Aktionswerte das Zielnetzwerk an. Anstatt also, wie in 3.22, das Maximum der Aktionswerte über alle möglichen Aktionen a' bezüglich des Zielnetzwerkes zu berechnen, wählt man nun die bezüglich des Strategienetzwerkes

beste Aktion aus und berechnet für diese den Aktionswert mithilfe des Zielnetzwerkes, sodass sich folgende neue Berechnung der Zielvariablen ergibt:

$$y^{DDQN} \quad := \quad r \; + \; \gamma \; Q_{\tilde{\boldsymbol{\theta}}} \left(s', \; \arg\max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s', a') \right) \tag{3.23}$$

Werden die Zielwerte durch tiefe Q-Netzwerke nach obiger Gleichung 3.23 berechnet, so spricht man nach [VGS16] auch von *tiefen Doppel-Q-Netzwerken* (engl. *Double Deep Q-Networks*, kurz DDQN). Ein Mini-Batch der Form 3.15 mit Zielwerten gemäß 3.23 wird in der Folge mit $\mathcal{B}_{DDQN}$ bezeichnet.

In den Experimenten in Kapitel 4 wird sich zeigen, dass jene Trennung von Aktionsauswahl und Aktionsbewertung zu stabileren Aktionswerten und besseren Lernergebnissen führen kann.

3.4 Duellierende Netzwerkarchitektur

Eine weitere Modifikation zielt auf die Struktur des tiefen Q-Netzwerkes ab. Wie in Bemerkung 3.3 bereits angeführt, werden tiefe Q-Netzwerke $Q_{\boldsymbol{\theta}}$ für gewöhnlich so konstruiert, dass für einen Zustand $s \in \mathcal{S}$ direkt per Vorwärts-Propagierung die Aktionswerte $Q_{\boldsymbol{\theta}}(s, a)$ für alle $a \in \mathcal{A}$ berechnet werden.

In [Wan+16] wird nun die sog. *duellierende Netzwerkarchitektur* (engl. *dueling network architecture*) vorgestellt, die Aktionswerte über den Zustandswert und den Vorteil bzw. Nachteil, den eine bestimmte Aktion gegenüber diesem Wert bringt, bestimmt. Hierzu wird die letzte Schicht des neuronalen Netzwerkes in einen sog. *Wertstrom* (engl. *value stream*) $V_{\boldsymbol{\theta}}(s)$ und einen Vorteilsstrom (engl. *advantage stream*) $A_{\boldsymbol{\theta}}(s, a)$ aufgespalten. Diese beiden Ströme werden anschließend auf eine spezielle Weise wieder zu den Aktionswerten $Q_{\boldsymbol{\theta}}(s, a)$ in der Ausgabeschicht zusammengeführt (siehe Abbildung 3.2 für eine schematische Darstellung).

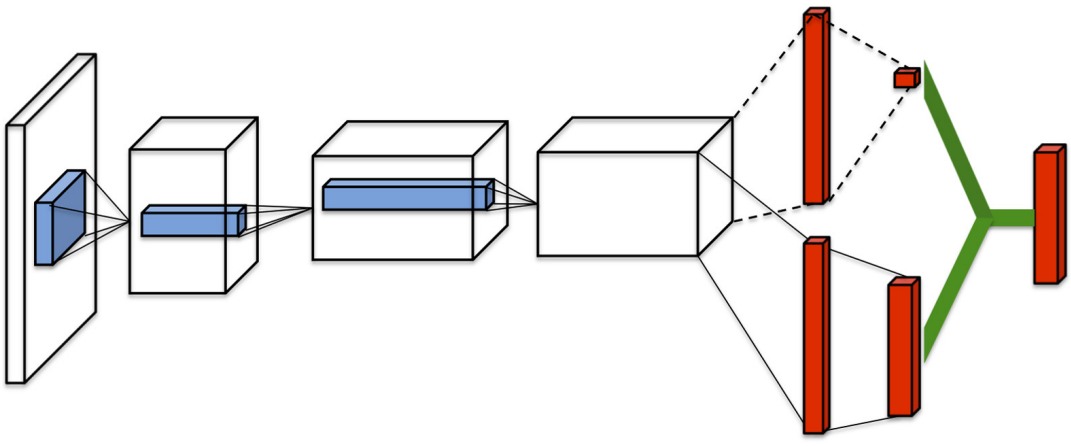


Abbildung 3.2: Duellierende Netzwerkarchitektur (entnommen aus [Wan+16])

Durch diese duellierende Netzwerkarchitektur wird mit jeder Parameteraktualisierung bezüglich einer Erfahrung (s, a, r, s') nicht nur der Aktionswert $Q_{\boldsymbol{\theta}}(s, a)$ approximiert, sondern gleichzeitig auch der Zustandswert $V_{\boldsymbol{\theta}}(s)$ und damit implizit ebenfalls die

Zustandswerte aller Aktionen $a' \in \mathcal{A}$, was zu einem effizienteren Lernprozess führen kann.

Da letztendlich $Q_{\boldsymbol{\theta}}(s,a)$ bzw. $V_{\boldsymbol{\theta}}(s)$ eine Approximation an den optimalen Aktionswert $q_*(s,a)$ bzw. an den optimalen Zustandswert $v_*(s)$ und entsprechend $A_{\boldsymbol{\theta}}(s,a)$ eine Approximation an deren Differenz $q_*(s,a) - v_*(s)$ liefern soll, erscheint es naheliegend, für die oben bereits angesprochene Zusammenführung von Wert- und Vorteilsstrom deren Summe

$$Q_{\boldsymbol{\theta}}(s,a) \quad \coloneqq \quad V_{\boldsymbol{\theta}}(s) \;+\; A_{\boldsymbol{\theta}}(s,a) \tag{3.24}$$

zu verwenden. Für einen gegebenen Aktionswert $Q_{\boldsymbol{\theta}}$ lässt sich jedoch keine eindeutige Zerlegung der Form 3.24 finden, da zum Beispiel eine Erhöhung von $V_{\boldsymbol{\theta}}$ um einen bestimmten Betrag und eine entsprechende Verringerung von $A_{\boldsymbol{\theta}}$ um denselben Betrag zu einer weiteren solchen Zerlegung führen würde. Gemäß [Wan+16] kann diese Mehrdeutigkeit der Zerlegung 3.24 die Funktionsfähigkeit des Lernverfahrens beeinträchtigen.

Um die Mehrdeutigkeit aufzulösen lohnt sich erneut die Betrachtung des bereits in Satz 2.12 hergestellten Zusammenhangs zwischen dem optimalen Aktionswert und dem optimalen Zustandswert:

$$v_*(s) \quad = \quad \max_{a \in \mathcal{A}} q_*(s,a) \tag{3.25}$$

Die Idee ist nun, diesen Zusammenhang auf die Approximatoren zu übertragen. Hierzu verwendet man folgende Zerlegung:

$$Q_{\boldsymbol{\theta}}(s,a) \quad \coloneqq \quad V_{\boldsymbol{\theta}}(s) \;+\; \left(A_{\boldsymbol{\theta}}(s,a) \;-\; \max_{a' \in \mathcal{A}} A_{\boldsymbol{\theta}}(s,a') \right) \tag{3.26}$$

Damit gilt der Zusammenhang:

$$\begin{aligned} \max_{a \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s,a) \quad &= \quad \max_{a \in \mathcal{A}} \left(V_{\boldsymbol{\theta}}(s) \;+\; \left(A_{\boldsymbol{\theta}}(s,a) \;-\; \max_{a' \in \mathcal{A}} A_{\boldsymbol{\theta}}(s,a') \right) \right) \\ &= \quad V_{\boldsymbol{\theta}}(s) \end{aligned} \tag{3.27}$$

Eine weitere sehr ähnliche Möglichkeit, die oben beschriebene Mehrdeutigkeit aufzulösen, besteht in der folgenden Zerlegung:

$$Q_{\boldsymbol{\theta}}(s,a) \quad \coloneqq \quad V_{\boldsymbol{\theta}}(s) \;+\; \left(A_{\boldsymbol{\theta}}(s,a) \;-\; \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A_{\boldsymbol{\theta}}(s,a') \right) \tag{3.28}$$

Auch wenn sie den Zusammenhang 3.27 nicht mehr erfüllt, wird in der Praxis meist trotzdem eher die zweite Möglichkeit verwendet, da diese im Vergleich zur ersten eine höhere Stabilität des Lernprozesses verspricht (vgl. [Wan+16]).

Bemerkung 3.4. Obwohl die duellierende Netzwerkarchitektur in den letzten verdeckten Schichten eine andere Struktur aufweist, verwendet sie dennoch in der Ein- und Ausgabeschicht dieselbe Struktur wie gewöhnliche tiefe Q-Netzwerke. Daher sind auch die duellierenden tiefen Q-Netzwerke mit allen weiteren, in diesem Kapitel vorgestellten Varianten kompatibel. Im Gegenteil wird in [Wan+16] sogar gezeigt, dass eine Kombination der Varianten zu noch besseren Ergebnissen führen kann.

3.5 Regularisierungs- und Mehrschrittfehler

Da das tiefe Q-Lernen auf der Minimierung des Fehlerterms $\mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}})$ beruht, ist es möglich, den Lernprozess durch Modifikationen des Fehlerterms auf gewünschte Weisen zu beeinflussen. Dies wird Hauptbestandteil dieses und zum Teil auch des nächsten Unterkapitels sein.

Regularisierungsfehler

Im vorherigen Unterkapitel wurde eine mögliche Architektur der letzten Schichten eines tiefen Q-Netzwerkes vorgestellt. Ein bisher recht wenig und zudem meist nur empirisch untersuchtes Problem betrifft die Architektur der vorangehenden verdeckten Schichten. So orientiert man sich bei der Wahl der Anzahl an verdeckten Schichten und der Neuronen pro Schicht – und damit auch der Anzahl an Parametern – in praktischen Anwendungen zumeist an Faustregeln und an Netzwerkarchitekturen, die in ähnlichen Anwendungen bereits gute Ergebnisse geliefert haben. Dabei tritt – oftmals auch gewollt – häufig der Fall auf, dass die Netzwerkarchitektur zu groß gewählt ist, was zu einer Überanpassung an die Trainingsdaten führen kann.

Ein Weg, diese Überanpassung einzuschränken ist es, möglichst viele der Parameterwerte nahe der Null zu halten. Dies lässt sich zum Beispiel durch das Hinzufügen eines weiteren Fehlerterms, der hohe Parameterwerte bestraft, realisieren. Im Fall der *L2-Regularisierung* verwendet man hierfür die Summe der quadrierten Parameterwerte, sodass sich folgender Regulariserungsfehler

$$\mathcal{J}_{L2}(Q_{\boldsymbol{\theta}}) \quad \coloneqq \quad \sum_{\theta \in \boldsymbol{\theta}} \theta^2 \tag{3.29}$$

ergibt, wobei unter Missbrauch der Notation $\theta \in \boldsymbol{\theta}$ einen Eintrag einer Gewichtungsmatrix oder eines Bias-Vektors des verwendeten tiefen Q-Netzwerkes bezeichne.

Mehrschrittfehler

Ein weiterer Fehlerterm zielt auf einen stärkeren Einbezug von weiter in der Zukunft liegenden Belohnungen des Agenten ab. Anstatt also, wie bisher, den Zielwert y_i für einen Zeitpunkt $i \in \mathbb{N}$ nur auf Grundlage der unmittelbaren Belohnung r_{i+1} und der eigenen Einschätzung des Agenten über den nächsten Zustand s_{i+1} zu berechnen, verwendet man nun die nächsten $n \in \mathbb{N}$ Belohnungen $r_{i+1}, \ldots, r_{i+n}$ und die Einschätzung über den n Schritte in der Zukunft liegenden Zustand s_{i+n} als Grundlage für die Berechnung der Zielwerte. Hierfür ist es essenziell, dass der Agent für einen gegebenen Zeitpunkt auch Zugriff auf Belohnungen und Zustände hat, die aus der Sicht jenes Zeitpunktes in der Zukunft liegen [5].

Zu diesem Zweck erweist sich erneut die zeitliche Trennung vom Durchleben einer Erfahrung und dem Lernen aus dieser bei der Erfahrungswiederholung sowie die Verwaltung der Erfahrungen in einem Wiederholungsspeicher $\mathcal{D}_t$ als sehr nützlich. Denn damit ist für eine Erfahrung $(s_i, a_i, r_{i+1}, s_{i+1}) \in \mathcal{D}_t$ (für $i \in \{t_0, \ldots, t-n\}$) auch der Zugriff auf die Erfahrungen $(s_{i+1}, a_{i+1}, r_{i+2}, s_{i+2}), \ldots, (s_{i+n-1}, a_{i+n-1}, r_{i+n}, s_{i+n})$

[5] Dieser Blick in die Zukunft bei der Berechnung der Mehrschrittfehler wird im Zusammenhang mit dem bestärkenden Lernen oftmals auch *Vorwärtssicht* (engl. *forward view*) genannt.

möglich, sodass in Kombination mit den Doppel-Q-Netzwerken aus Kapitel 3.3 folgender n-Schritt Zielwert berechnet werden kann:

$$y_i^n \quad := \quad \sum_{k=0}^{n-1} \gamma^k \; r_{i+k+1} \quad + \quad \gamma^n \; Q_{\widetilde{\boldsymbol{\theta}}} \left(s_{i+n}, \; \arg\max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s_{i+n}, a') \right) \tag{3.30}$$

Bemerkung 3.5. Ähnlich wie im einfachen Fall, lässt sich auch der n-Schritt Zielwert als Approximator der rechten Seite einer n-mal auf sich selbst angewendeten Bellman-Gleichung gemäß 2.5 mathematisch motivieren. Insbesondere ist die bisherige Berechnung der Zielwerte ein Spezialfall der n-Schritt Zielwerte für $n = 1$.

Ein Mini-Batch der Form 3.15 mit Zielwerten y_i^n gemäß 3.30 wird in der Folge mit $\mathcal{B}_n$ bezeichnet. Der n-Schritt Fehler $\mathcal{J}_{\mathcal{B}_n}(Q_{\boldsymbol{\theta}})$ lässt sich damit analog zum ein-Schritt Fehler $\mathcal{J}_{\mathcal{B}}(Q_{\boldsymbol{\theta}})$ berechnen.

Die Verwendung des Mehrschrittfehlers ist vor allem dann sinnvoll, wenn die Umgebung nur sehr selten von Null verschiedene Belohnungen ausgibt und zudem Handlungen die für das Erreichen einer Belohnung maßgeblich sind erst später zu einer Belohnung führen.

3.6 Tiefes Q-Lernen mit Demonstrationen

Wenngleich der im vorherigen Kapitel eingeführte Mehrschrittfehler die Lernfähigkeit des Agenten bei seltenen und zeitlich verzögerten Belohnungen verbessern kann, gibt es weiterhin Umgebungen, in denen es sehr unwahrscheinlich ist, dass Agenten ohne jegliches Vorwissen jemals positive Belohnungen erhalten (siehe z.B. Kapitel 4). In einem solchen Fall kann das reine Lernen durch Bestärkung sehr schwierig bis nahezu unmöglich werden.

Wie bereits in der Einleitung angeführt können Menschen nicht nur aus den Erfahrungen, die sie selbst durch die Interaktion mit ihrer Umgebung gemacht haben, lernen, sondern auch durch Erfahrungen, die sie bei anderen beobachten oder die andere an sie weitergeben. Dieses Lernen von Demonstrationen erlaubt es dem Menschen, die Bewältigung komplexer Aufgaben deutlich schneller zu erlernen, als es ihm allein möglich wäre. Daher ist es ein vielversprechender Ansatz, diese Fähigkeit auch auf Agenten des bestärkenden Lernens zu übertragen. Voraussetzung hierfür ist ein Datensatz von Demonstrationen eines Experten – unerheblich, ob Mensch oder Maschine –, der dasselbe Problem, wie das dem Agenten gestellte, bereits gelöst hat. Aufgrund der immer weiter steigenden Datenerfassung ist diese Voraussetzung jedoch in vielen Fällen erfüllt.

Ist nun also ein Datensatz mit Demonstrationen eines Experten gegeben, stellt sich die Frage, wie ein Agent mithilfe dieser Demonstrationen lernen kann. Neben der Möglichkeit, hierfür ohne weitere Anpassungen ein Verfahren des überwachten Lernens – in diesem Fall spricht man auch vom *Verhaltensklonen* (engl. *behavioral cloning*) – oder die durch geschickte Erweiterungen des Datensatzes ausgereifteren Verfahren des *Imitationslernens* (engl. *imitation learning*, siehe z.B. [Sch99] oder [RGB11]) zu verwenden, wird in [Hes+18] ein Algorithmus eingeführt, der es ermöglicht, das tiefe Q-Lernen mit dem Lernen von Demonstrationen zu verbinden.

Dies wird zum einen dadurch realisiert, dass Demonstrationen des Experten wie die eigenen Erfahrungen des Agenten behandelt werden, sodass die Zielwertberechnungen und Mini-Batch-Aktualisierungen wie zuvor durchgeführt werden können. Zum anderen wird erneut ein zusätzlicher Fehlerterm eingeführt, der dazu führen soll, dass der Agent besonders die Demonstrationen des Experten nachahmt. Es wird deshalb versucht sicherzustellen, dass der Aktionswert $Q_{\boldsymbol{\theta}}(s_E, a_E)$ für einen Zustand $s_E \in \mathcal{S}$, in dem der Experte eine Aktion $a_E \in \mathcal{A}$ ausgeführt hat, um eine gewisse Spanne größer ist als $Q_{\boldsymbol{\theta}}(s_E, a')$ für $a' \in \mathcal{A}$. Für die Herstellung dieser Spanne verwendet man bei der Berechnung des sog. Experten-Zielwertes eine Funktion

$$\ell(a_E, a') \quad \coloneqq \quad c_E \;\; \mathbb{1}_{\{a \in \mathcal{A} \;\mid\; a = a_E\}}(a'), \tag{3.31}$$

wobei $\mathbb{1}$ die Indikatorfunktion und $c_E \in \mathbb{R}_{>0}$ eine positive Konstante sei, die in der Folge als *Experten-Klassifikationskoeffizient* bezeichnet wird. Damit kann der Experten-Zielwert folgendermaßen berechnet werden:

$$y^E \quad \coloneqq \quad \max_{a' \in \mathcal{A}} \;\; Q_{\boldsymbol{\theta}}(s_E, a') \;\; + \;\; \ell(a_E, a') \tag{3.32}$$

Bemerkung 3.6. Aus der Beschreibung in [Hes+18] geht nicht hervor, ob auch für die Berechnung der Experten-Zielwerte die Doppel-Q-Netzwerke verwendet werden. In diesem Fall würde man die Zielwerte nach der Formel

$$y^E \quad \coloneqq \quad Q_{\tilde{\boldsymbol{\theta}}}(s_E, a^*) \;\; + \;\; \ell(a_E, a^*) \tag{3.33}$$

berechnen, wobei $a^* \coloneqq \arg\max_{a' \in \mathcal{A}} \; Q_{\boldsymbol{\theta}}(s_E, a') \;+\; \ell(a_E, a')$. Während das eigentliche Ziel der Verwendung von Doppel-Q-Netzwerken die Vermeidung von Überschätzungen der Zielwerte ist, besteht der Sinn der Experten-Zielwerte jedoch gerade darin, die Aktionswerte des Demonstrators höher einzuschätzen. Deshalb wird hier bei der Berechnung der Experten-Zielwerte auf die Verwendung der Doppel-Q-Netzwerke verzichtet, was den gewünschten Effekt der Funktion ℓ zusätzlich unterstützt.

Ein Mini-Batch gemäß 3.15, mit Zielwerten der Form 3.32 für Demonstrationen des Experten berechnet, wird in der Folge mit $\mathcal{B}_E$ bezeichnet [6]. Im Gegensatz zu den anderen Fehlern wird in [Hes+18] für die Berechnung von $\mathcal{J}_{\mathcal{B}_E}(Q_{\boldsymbol{\theta}})$ nicht der mittlere quadratische, sondern der mittlere absolute Fehler verwendet, d.h.

$$\mathcal{J}_{\mathcal{B}_E}(Q_{\boldsymbol{\theta}}) \quad \coloneqq \quad \frac{1}{|\mathcal{B}_E|} \sum_{(\boldsymbol{x}, y^E) \in \mathcal{B}_E} |Q_{\boldsymbol{\theta}}(\boldsymbol{x}) \;\; - \;\; y^E| \tag{3.34}$$

Neben einem möglicherweise besseren und schnelleren Lernerfolg durch das Nachahmen des Experten besteht ein weiterer großer Vorteil des Lernens von Demonstrationen darin, dass der Agent durch sog. *Vortraining* mit den Demonstrationsdaten bereits eine hinreichend gute Strategie lernen kann – noch bevor er einen einzigen Schritt in der Umgebung machen musste. Dies ist vor allem dann nützlich, wenn das Training in der Umgebung nicht simuliert werden kann, sondern in der realen Welt stattfindet, wo schlechte Strategien des Agenten möglicherweise großen Schaden anrichten können.

[6] Die Experten-Zielwerte werden hierbei nur für die Demonstrationen des Experten und nicht für die eigenen Erfahrungen des Agenten berechnet.

Das in [Hes+18] eingeführte und in Algorithmus 4 zusammenfassend dargestellte *tiefe Q-Lernen mit Demonstrationen* (engl. *deep Q-learning from demonstrations*, kurz DQfD) kombiniert nun alle in diesem Kapitel vorgestellten Varianten des tiefen Q-Lernens:

Algorithmus 4: Tiefes Q-Lernen mit Demonstrationen (vergleiche [Hes+18])

Eingabe:

$(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$	Markov-Entscheidungsprozess mit Faktor $\gamma \in (0,1)$
$\mathcal{D}$	mit Demonstrationen initialisierter Wiederholungsspeicher
$\boldsymbol{\theta}, \widetilde{\boldsymbol{\theta}}$	zufällig initialisierte Strategie- und Zielnetzwerk-Parameter
τ	Aktualisierungsintervall des Zielnetzwerkes
T_V, T_U	Anzahl an Vor- bzw. Umgebungstrainings-Schritten
$\eta_1, ..., \eta_T$	Lernraten, wobei $T \coloneqq T_V + T_U$
$\lambda_{DDQN}, \lambda_n, \lambda_E, \lambda_{L2}$	Fehlergewichtungen

Ausgabe: Parameterwerte $\boldsymbol{\theta}$, sodass $Q_{\boldsymbol{\theta}}$ eine Approximation an den optimalen Aktionswert q_* bezüglich $(\mathcal{S}, \mathcal{A}, p, \mathcal{R})$ darstellt

für $t \leftarrow 1$ **bis** T **führe aus**

- ziehe eine Indexmenge $\mathcal{I}$ mit Priorisierung
- berechne die Zielwerte y_i^{DDQN} und y_i^n für alle $i \in \mathcal{I}$ sowie y_i^E, falls i eine Demonstration des Experten repräsentiert
- erhalte die entsprechenden Mini-Batches $\mathcal{B}_{DDQN}$, $\mathcal{B}_n$ und $\mathcal{B}_E$
- berechne den Fehler:
  $\mathcal{J}(Q_{\boldsymbol{\theta}}) \coloneqq \lambda_{DDQN}\mathcal{J}_{\mathcal{B}_{DDQN}}(Q_{\boldsymbol{\theta}}) + \lambda_n\mathcal{J}_{\mathcal{B}_n}(Q_{\boldsymbol{\theta}}) + \lambda_E\mathcal{J}_{\mathcal{B}_E}(Q_{\boldsymbol{\theta}}) + \lambda_{L2}\mathcal{J}_{L2}(Q_{\boldsymbol{\theta}})$
- berechne $\nabla_{\boldsymbol{\theta}}\mathcal{J}(Q_{\boldsymbol{\theta}})$ durch Rückpropagierung
- aktualisiere die Parameter: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_t \nabla_{\boldsymbol{\theta}}\mathcal{J}(Q_{\boldsymbol{\theta}})$
- **wenn** $t \bmod \tau = 0$ **dann** $\widetilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$
- **wenn** $t > T_V$ **dann**
  - beobachte den aktuellen Zustand $s \in \mathcal{S}$ und wähle eine Aktion $a \in \mathcal{A}$
  - erhalte Belohnung $r \in \mathbb{R}$ und neuen Zustand $s' \in \mathcal{S}$
  - füge die aktuelle Erfahrung (s, a, r, s') zu $\mathcal{D}$ hinzu und lösche ggf. die älteste eigene Erfahrung
  - aktualisiere den aktuellen Zustand: $s \leftarrow s'$

Bemerkung 3.7. Der Auswahl der Aktionen im bestärkenden Lernen im Allgemeinen und in Algorithmus 4 im Speziellen liegt das sog. *Exploration-Exploitation-Dilemma* zugrunde. Dieses beschreibt den Kompromiss, den der Agent bei der Auswahl der nächsten Aktion eingehen muss, um entweder durch Exploration neues

Wissen über seine Umgebung anzusammeln oder durch Exploitation sein vorhandenes Wissen für die Wahl der besten Aktion auszunutzen. In [Hes+18] wird jenes Dilemma mit einer sog. *Epsilon-Annäherungsstrategie* gehandhabt. Dies bedeutet, dass die Aktion $a_t \in \mathcal{A}$ mit einer Wahrscheinlichkeit ϵ rein zufällig und mit Wahrscheinlichkeit $1 - \epsilon$ gemäß der aktuell besten Strategie, d.h. $a_t \coloneqq \arg\max_{a \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s_t, a)$, ausgewählt wird. Die Wahrscheinlichkeit ϵ wird hierbei von einem anfänglich hohen Wert im Verlauf des Lernprozesses bis zu einem vorher festgelegten niedrigeren Wert verringert. In Kapitel 4 wird sich zeigen, dass bereits diese eher einfache Handhabung des Exploration-Exploitation-Dilemmas bei einigen Problemen zu guten Ergebnissen führen kann. Gerade bei sehr anspruchsvollen Problemen kann es dagegen sein, dass die Epsilon-Annäherungsstrategie keine gute Lösung mehr liefert. Daher besteht einer der Schwerpunkte der aktuellen Forschung im bestärkenden Lernen darin, ausgereiftere Explorationsstrategien, auf die im Rahmen dieser Arbeit jedoch nicht genauer eingegangen werden kann, zu entwickeln.

Nachdem hiermit die theoretischen Grundlagen des tiefen Q-Lernens mit Demonstrationen eingeführt sind, wird sich das nun folgende vierte Kapitel damit befassen, die Effekte der einzelnen Bestandteile von Algorithmus 4 auf den Lernprozess experimentell zu untersuchen und zu veranschaulichen.

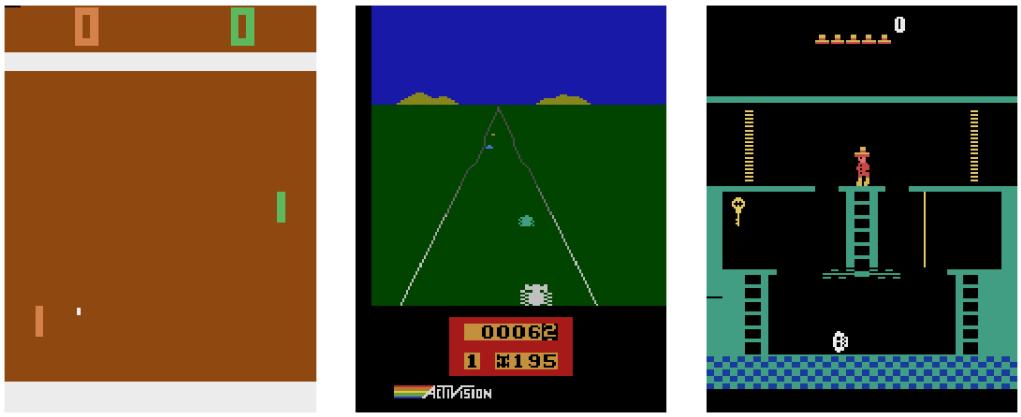
Kapitel 4

Numerische Experimente

Um die Wirkungsweisen der in Kapitel 3 vorgestellten Varianten des tiefen Q-Lernens anhand von numerischen Experimenten untersuchen zu können, ist zunächst ein sequentielles Problem in einer dafür geeigneten Testumgebung nötig. Als Testumgebung für den Vergleich von Algorithmen des bestärkenden Lernens auf Grundlage von visuellen Wahrnehmungen haben sich Videospiele etabliert. Bei diesen sind der Zustandsraum und die Zustandsübergänge durch die beim Spiel erzeugten Pixel, der Aktionsraum durch die möglichen Kontrolltasten des Spiels sowie die Belohnungen durch die Veränderungen des Spielstands auf natürliche Weise gegeben. Dadurch, dass sie eine simulierte Umgebung sind, bieten Videospiele zudem den Vorteil, dass der Lernprozess nicht in der realen Welt stattfinden muss, wo die Ausführung von Aktionen des Agenten sowie der entsprechende Zustandsübergang zum einen deutlich länger dauern und zum anderen erheblichen Schaden anrichten kann. Trotz der Tatsache, dass Videospiele nur Simulationen sind, können sie dabei aufgrund der großen Bandbreite an verschiedenen, der Realität immer näher kommenden Spielen dennoch Aufschlüsse über die Funktionsfähigkeit von Algorithmen des bestärkenden Lernens bei Problemen auch aus der realen Welt liefern.

4.1 Lernumgebung und Datensatz

In dieser Arbeit wird – genau wie in [Mni+15], [Sch+15], [VGS16], [Wan+16] und [Hes+18] – die in [Bel+13] vorgestellte *Arcade Learning Environment* (ALE) als Testumgebung dienen. Diese bietet über 50 verschiedene, in den 1970er und 1980er Jahren ursprünglich für die *Atari 2600* Spielekonsole entworfene Spiele, die zwar nicht besonders realitätsnah sind, aber dennoch wichtige Eigenschaften eines Agenten des bestärkenden Lernens erfordern, wie zum Beispiel die Fähigkeit, sich an Zustandsänderungen der Umgebung anzupassen und langfristig zu planen. Anders als in den oben genannten Quellen ist das Testen der Algorithmen für jedes dieser Spiele im Rahmen dieser Arbeit nicht möglich. Stattdessen werden im Folgenden drei Spiele, die sich hinsichtlich ihres Schwierigkeitsgrades, der Anzahl an auswählbaren Aktionen und der Vielfalt der möglichen Zustände unterscheiden, insbesondere im Hinblick auf die in Kapitel 3 vorgestellten Varianten des tiefen Q-Lernens genauer untersucht.



(a) *Pong* **(b)** *Enduro* **(c)** *Montezumas's Revenge*

Abbildung 4.1: Spielbildschirme der getesteten Spiele

Pong

Das erste Spiel *Pong* (siehe Abb. 4.1a) soll im weitesten Sinn an ein Tischtennisspiel erinnern. So hat der Agent das Ziel, durch die Bewegung seines Schlägers in Form eines senkrechten Strichs am rechten Rand des Bildschirms – d.h. es gibt drei mögliche Aktionen[1] – einen Ball in Form eines Punktes am Schläger seines Gegners am linken Bildschirmrand vorbeizuspielen. Gelingt dies einem der beiden Spieler, so erhält entweder der Agent oder sein Gegner einen Punkt. Entsprechend erhält der Agent eine Belohnung von $+1$ bzw. -1 falls er bzw. sein Gegner einen Punkt macht. Eine Episode wird beendet, sobald der erste Spieler 21 Punkte erreicht hat, sodass das mögliche Endergebnis für den Agenten zwischen -21 und 21 liegt.

Enduro

Das nächste Spiel *Enduro* (siehe Abb. 4.1b) simuliert ein Langstrecken-Autorennen. Ziel des Agenten ist es, innerhalb eines simulierten Tages eine vorgegebene Anzahl an Autos zu überholen und sich somit für den nächsten Tag zu qualifizieren. Erreicht er die Vorgabe nicht, so scheidet der Agent aus und die Episode wird beendet. Hierbei kann der Agent zwischen Beschleunigen, Bremsen, Bewegen nach links bzw. rechts sowie verschiedenen Kombinationen dieser Aktionen auswählen, sodass sich insgesamt eine Anzahl von neun möglichen Aktionen ergibt. Der Agent erhält die Belohnung $+1$, falls er ein anderes Auto überholt, und -1, falls er selbst überholt wird und gleichzeitig noch weniger als die vorgegebene Anzahl zu überholen hat. Das Gesamtergebnis wird entsprechend durch die Summe aller in einer Episode erhaltenen Belohnungen berechnet. Als Referenzwert erreicht ein guter menschlicher Spieler gemäß [Wan+16] ein durchschnittliches Gesamtergebnis von ca. 860.

Während des Rennens muss sich der Agent auf verschiedene Licht- und Wetterbedingungen, wie zum Beispiel das Fahren bei Nacht, Nebel oder Schnee, sowie von Tag zu Tag schneller werdende Gegner einstellen. In Kombination mit der größeren Anzahl an möglichen Aktionen macht dies *Enduro* bereits zu einem deutlich anspruchsvolleren Spiel für Agenten des bestärkenden Lernens als *Pong*.

[1] Tatsächlich stehen in der ALE sechs Aktionen zur Auswahl, von denen jedoch jeweils zwei denselben Effekt haben.

Montezuma's Revenge

Als eines der anspruchsvollsten Spiele der ALE gilt jedoch *Montezuma's Revenge* (siehe Abb. 4.1c), bei dem sich der Agent durch viele verschiedene Räume bewegt, die auf mehreren Ebenen pyramidenartig angeordnet und miteinander verbunden sind. Hierbei muss er Edelsteine und andere Gegenstände einsammeln, wofür er Belohnungen erhält, und Hindernissen, wie Schlangen, Lavagruben oder sich durch den Raum bewegenden Totenköpfen, ausweichen. Fällt er einem der Hindernisse zum Opfer oder stürzt er von einer Leiter oder Plattform, so verliert der Agent eines seiner anfänglich fünf Leben. Eine Episode wird beendet, sobald der Agent sein letztes Leben verloren hat. Der Agent kann hierbei in acht verschiedene Richtungen laufen oder springen, sowie auf der Stelle springen oder stehen bleiben, was zu insgesamt 18 möglichen Aktionen führt.

Während das in [Wan+16] angegebene Durchschnittsergebnis für menschliche Spieler ca. 4753 Punkte beträgt, erzielen Agenten des tiefen Q-Lernens auch mit den in Kapitel 3.2 bis 3.5 vorgestellten Erweiterungen keinen einzigen Punkt. Die große Schwierigkeit von *Montezuma's Revenge* für Agenten des bestärkenden Lernens besteht darin, dass zum einen eine überaus langfristige Planung notwendig ist[2] und zum anderen nur sehr selten von null verschiedene Belohnungen ausgegeben werden. Zusammen mit der im Vergleich zu den restlichen Spielen großen Auswahl an Aktionen und den vielfältigen Möglichkeiten ein Leben zu verlieren, führt dies dazu, dass die meisten Algorithmen des bestärkenden Lernens keine geeignete Strategie für *Montezuma's Revenge* finden.

Datensatz

Wie bereits in Kapitel 3.6 angeführt, können bei solchen Problemen, für die ein Agent von allein keine Lösung findet, Demonstrationen verwendet werden, um dennoch einen Lernfortschritt zu erreichen. In [Zha+19] wird ein Datensatz von Demonstrationen eines Menschen für verschiedene Spiele der ALE vorgestellt und zugänglich gemacht, aus dem auch die in der Folge verwendeten Demonstrationen für das Spiel *Montezuma's Revenge* stammen. Der Datensatz enthält neben dem Spielbildschirm für einen gegebenen Zeitpunkt einer Episode sowie der zu diesem Zeitpunkt getätigten Aktion und erhaltenen Belohnung auch das per Eye-Tracking aufgenommene Blickfeld des Demonstrators, welches im Rahmen dieser Arbeit jedoch nicht verwendet wird. Um das Blickfeld aufzunehmen und dem Demonstrator eine optimale Entscheidungsfindung zu ermöglichen, wurden die Spiele bei der Erstellung des Datensatzes für jede Entscheidung pausiert. Dies führt zum einen zu besseren und zum anderen zu weniger realistischen Demonstrationen, als es beim Spiel in Realgeschwindigkeit möglich wäre.

Das Auslesen und Verarbeiten dieses Datensatzes ist ein Teil der im nächsten Unterkapitel beschriebenen Implementierung.

[2] Zum Beispiel muss der Spieler in einem Raum einen Schlüssel einsammeln, um in einem anderen Raum zu einer Fackel zu gelangen, mit der er wiederum Edelsteine in einem weiteren Raum finden kann.

4.2 Implementierung

Um die in Kapitel 3 nur im Pseudocode vorgestellten Verfahren auch testen zu können, ist es nötig, diese in Programmcode zu überführen. Da die Erstellung dieser Programmcodes einen wichtigen Teil der vorliegenden Arbeit darstellt, folgt nun ein kurzer Überblick über einige Besonderheiten der größtenteils in der Programmiersprache *Python* verfassten und auf der beiliegenden *Compact Disc* befindlichen Implementierungen.

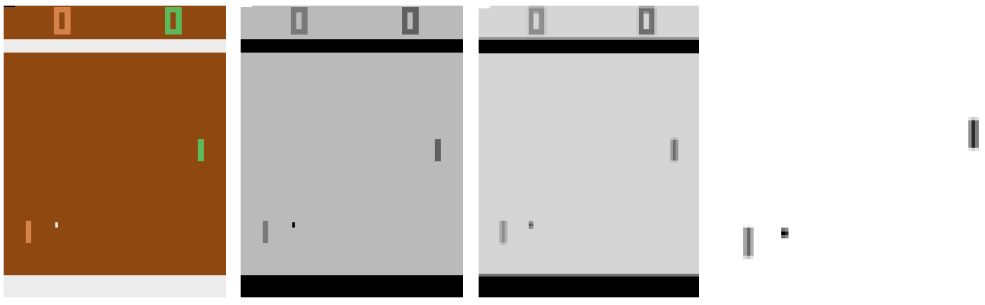
Die Implementierung unterteilt sich dabei im Wesentlichen in die vier folgenden Hauptbestandteile.

Vorverarbeitung

Ein erster wichtiger Schritt bei der Implementierung ist die Vorverarbeitung der von der Umgebung produzierten und der im Demonstrationsdatensatz gegebenen Daten. Das Vorgehen hierbei orientiert sich größtenteils an den Beschreibungen in [Mni+15]. Die Einbindung der ALE in die *Python*-Programme baut auf dem in [Bro+16] vorgestellten Softwarepaket *OpenAi Gym* auf.

Die Umgebung generiert RGB-Bilder in Form von (210, 160, 3)-dimensionalen Matrizen und entsprechende Belohnungen mit einer Frequenz von 60 Hz. Da sich aufeinander folgende Bilder bei dieser Frequenz sehr stark ähneln, entstehen hierdurch große Redundanzen in den Daten. Daher wird nur jedes n_{skip}-te Bild betrachtet und ausgewählte Aktionen werden für die folgenden n_{skip} Schritte wiederholt. In manchen Spielen, in denen einige Objekte nicht in jedem, sondern zum Beispiel nur in jedem zweiten, Bild zu sehen sind kann dies zu Problemen führen, weshalb man jeweils das Maximum der Pixelwerte der zwei letzten Bilder verwendet.

In Kapitel 2 wurde die Markoveigenschaft als wichtige Voraussetzung für die Funktionsfähigkeit von Algorithmen des bestärkenden Lernens eingeführt. Würde in diesem Zusammenhang ein einzelnes Bild den Zustand der Umgebung repräsentieren, wäre die Markoveigenschaft nicht erfüllt, da zum Beispiel bei sich bewegenden Objekten die Bewegungsrichtung und Geschwindigkeit nicht durch ein einziges Bild erfasst wird. Um dieses Problem zu umgehen wird eine Anzahl von n_{stack} Bildern zu einem Zustand zusammengefasst.



(a) Eingabe **(b)** Graustufen **(c)** Skalierung **(d)** Ausschnitt

Abbildung 4.2: Stadien der Bildbearbeitung im Spiel *Pong*

Zusätzlich werden die Bilder in folgender Weise bearbeitet: Zunächst werden die RGB-Werte in 256 mögliche Graustufen konvertiert, sodass sich zum einen die

Dimension von (210, 160, 3) auf (210, 160) und zum anderen der Speicherbedarf durch die nun mögliche Repräsentation der Pixelwerte in 8-Bit unsigned Integer-Werten reduziert. Eine weitere Reduktion von Dimension und Speicherbedarf wird durch das Herunterskalieren der Bilder auf Dimension (110, 84) erreicht[3]. Abschließend werden die Bilder noch auf den für das Spiel relevanten Bereich auf die Dimension (84, 84) zugeschnitten. Dies verhindert zusätzlich, dass Aktionswerte anhand des meist am Bildschirmrand dargestellten Spielstandes gelernt werden. Eine Veranschaulichung der gerade beschriebenen Schritte der Bildbearbeitung ist in Abb 4.2 für das Spiel *Pong* dargestellt.

Neben den Bildern werden auch die Belohnungen bearbeitet. Dies ist nötig, da sich die von der Umgebung ausgegebenen Belohnungen sowohl von Spiel zu Spiel als auch innerhalb eines Spiels teilweise stark unterscheiden können. Sind Belohnungen innerhalb eines Spiels betragsmäßig in etwa auf demselben Niveau, so reicht bereits die Vorzeichenfunktion $\hat{r} \coloneqq sign(r)$ für eine geeignete Repräsentation der Belohnungen aus[4]. Für Spiele, bei denen die Belohnungen innerhalb eines Spiels unterschiedlich sind[5], wäre es wichtig, dass auch der Agent diese Unterschiede erkennen kann. Hierfür wird die in [Hes+18] vorgeschlagene Belohnungsbearbeitung $\hat{r} \coloneqq sign(r) \cdot \log(1 + |r|)$ verwendet. Dies stellt einerseits sicher, dass die Belohnungen in einem angemessenen Rahmen liegen, und ermöglicht andererseits trotzdem eine Unterscheidung der Höhe der Belohnungen.

Neben den oben beschriebenen – selbst implementierten – Schritten zur Vorverarbeitung wurden weitere Funktionen, die v.a. das Zurücksetzen und Starten der Umgebung nach dem Ende bzw. vor dem Anfang von Episoden regeln, aus den *Baseline*-Implementierungen von *OpenAI* (siehe [Dha+17]) übernommen.

Wiederholungsspeicher

Einen weiteren wichtigen Teil stellt die Implementierung des Speichers für die (priorisierte) Erfahrungswiederholung dar. Der Speicher sollte dabei im Wesentlichen Erfahrungen aufnehmen, überschreiben und zur priorisierten Erfahrungswiederholung zur Verfügung stellen können. Da teilweise sehr große Wiederholungsspeicher mit einer Kapazität von über einer Million Erfahrungen verwendet werden, ist es hierbei besonders wichtig, das Speichern und Ausgeben von Erfahrungen möglichst effizient zu gestalten, weshalb dieser Teil auch in der effizienteren C-Erweiterung *Cython* der Programmiersprache *Python* verfasst ist.

Obwohl die Zustände, wie oben beschrieben, aus mehreren Bildern bestehen, werden sie nicht als Ganzes gespeichert, sondern nur die einzelnen Bilder. Dies erfordert deutlich weniger Speicherplatz, da zwei aufeinander folgende Zustände $s_{i-1} \coloneqq (f_{i-n_{\text{stack}}}, \dots, f_{i-1})$ und $s_i \coloneqq (f_{i-n_{\text{stack}}+1}, \dots, f_i)$, wobei $f_{t-n_{\text{stack}}}, \dots, f_i$ die

[3] Bei Betrachtung der Dimension des skalierten Bildes könnte die Frage aufkommen, warum die Dimensionen nicht auf genau die Hälfte (105, 80) skaliert werden, was durch Auslassen jedes zweiten Pixels recht einfach möglich wäre. Tatsächlich ist die auf den ersten Blick merkwürdige erscheinende Skalierung jedoch mit Absicht so gewählt, dass die relevanten Objekte im Bild durch die entstehende Unschärfe größer erscheinen.

[4] Da die Belohnungen in den Spielen *Pong* und *Enduro* ohnehin bereits -1, 1 oder 0 betragen, hat die Bearbeitung der Belohnungen in diesen Fällen keinen Einfluss.

[5] Zum Beispiel erhält der Agent in *Montezuma's Revenge* 100 Punkte für das Aufsammeln eines Schlüssels, während er 1000 für das Aufsammeln eines Edelsteins erhält.

einzelnen Bilder sind, $n_{\text{stack}} - 1$ Bilder gemeinsam haben. Die Bilder werden dann erst beim Zugriff für die Erstellung eines Mini-Batches zu Zuständen zusammengefasst. Die zu den Zuständen s_i gehörigen Aktionen a_i, Belohnungen r_{i+1} und Prioritäten p_i werden in separaten Arrays gespeichert.

Die effiziente Ziehung der Indizes für die Erstellung jener Mini-Batches mit Priorisierung[6] ist eine weitere Herausforderung bei der Implementierung des Wiederholungsspeichers. Eine Möglichkeit für die Ziehung von Indizes mit Wahrscheinlichkeit proportional zu ihrer Priorität ist es, zunächst eine zwischen Null und der Summe aller Prioritäten gleichverteilte Zahl $Z \sim \mathcal{U}([0, \sum_j p_j])$ zu ziehen und anschließend den Index i zu finden, sodass gilt:

$$\sum_{j=0}^{i-1} p_j \;<\; Z \;\leq\; \sum_{j=0}^{i} p_j \tag{4.1}$$

Für das effiziente Auffinden eines solchen Indexes wird die Datenstruktur des *binären Summenbaumes* verwendet. Dies ist ein azyklischer gerichteter Graph mit Knotengewichten, bei dem – abgesehen von den *Blättern*, d.h den Knoten des Graphen ohne ausgehende Kanten – jeder Knoten genau zwei Nachfolger besitzt, sodass das Gewicht eines Knotens gleich der Summe der Gewichte seiner Nachfolger ist.

Werden also die Prioritäten als Knotengewichte der Blätter eines solchen binären Summenbaums aufgefasst, so lässt sich mit der in Abb. 4.3 dargestellten Funktion in $\mathcal{O}(\log(n_{cap}))$, wobei n_{cap} die Kapazität des Wiederholungsspeichers sei, Schritten ein Index gemäß 4.1 finden.

Funktion `Abruf(`*i, z*`)`**:**

- **wenn** *i ist ein Blatt* **dann**
  - **Rückgabe** *i*
- **wenn** *Gewicht(links(i))* $\geq z$ **dann**
  - **Rückgabe** `Abruf(`*links(i), z*`)`
- **sonst**
  - **Rückgabe** `Abruf(`*rechts(i), z - Gewicht(links(i))*`)`

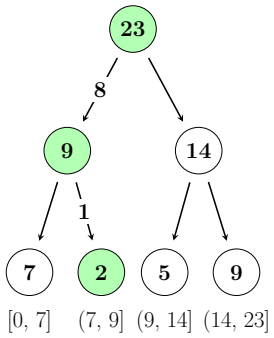


Abbildung 4.3: Funktion und Beispiel (für $Z = 8$) zum Abruf eines Indexes gemäß 4.1 mithilfe eines binären Summenbaums, wobei *links* bzw. *rechts* den linken bzw. rechten Nachfolger sowie *Gewicht* das Gewicht eines Knotens bezeichne.

Nach der Ziehung einer Menge $\mathcal{I}$ von Indizes in der oben beschriebenen Weise können dann Erfahrungen der Form $(s_i, a_i, r_{i+1}, s_{i+1})$ für alle $i \in \mathcal{I}$ an den Agenten weitergegeben werden[7]. Auf Grundlage dieser Erfahrungen werden anschließend die gewünschten Zielwerte y_i^{DDQN}, y_i^n sowie y_i^E berechnet. Bei dieser Berechnung spielen die tiefen Q-Netzwerke, die den nächsten Teil der Implementierung darstellen, eine wichtige Rolle.

[6] Hier wird nur die in Kapitel 3.2 eingeführte proportionale Priorisierung umgesetzt.

[7] Je nach verwendeter Variante werden zusätzlich die Gewichte w_i gemäß 3.21, die n-Schritt-Erträge und Zustände s_{i+n} sowie die Information, ob s_{i+1} bzw. s_{i+n} terminal sind, übergeben.

Tiefe Q-Netzwerke

Die Implementierung der tiefen Q-Netzwerke baut auf den *Python*-Bibliotheken *Tensorflow* ([Mar+15]) und *Keras* ([Cho15]) auf. Diese ermöglichen einerseits eine übersichtliche Erstellung komplexer Netzwerkarchitekturen sowie andererseits eine effiziente Ausführung der sehr rechenintensiven Vorwärts- und Rückpropagierung auf Grafikprozessoren. Als Aufbau des Netzwerkes wurde dabei die in [Wan+16] beschriebene Architektur verwendet.

Im Vergleich zur Theorie in Kapitel 3 wurden für die Implementierung zwei Anpassungen des Netzwerktrainings vorgenommen. Die erste Anpassung betrifft die verwendete Kostenfunktion. So wird anstelle des quadratischen Fehlers eine Form des sog. *Huber*-Fehlers (vgl. [Hub92]) benutzt. Für eine Vorhersage y und einen entsprechenden Zielwert y_{Ziel} ist dieser Fehler folgendermaßen definiert:

$$\mathcal{C}_{\text{Huber}}(y_{\text{Ziel}}, y) \quad := \quad \begin{cases} \frac{1}{2} \ (y_{\text{Ziel}} - y)^2, & \text{falls } |y_{\text{Ziel}} - y| \leq 1 \\ |y_{\text{Ziel}} - y| - \frac{1}{2} & \text{andernfalls} \end{cases} \tag{4.2}$$

Der Huber-Fehler hat den Vorteil, dass er gegenüber großen Abweichungen zwischen Zielwert und Vorhersage robuster ist als der quadratische Fehler, was zu einem stabileren Lernprozess führen kann. Als zweite Anpassung wird der Aktualisierungsschritt 3.12 durch die sog. *adaptive Momentum Schätzung* (engl. *adaptive moment estimation*, kurz ADAM) ersetzt (vgl. [KB15]). Hierbei fließen neben den aktuellen Gradienten auch die Gradienten der vorherigen Schritte sowie die quadrierten Gradienten des aktuellen und der vorherigen Schritte mit in die Berechnung der aktualisierten Parameterwerte ein.

Training des Agenten

Beim Agenten werden abschließend die vorherigen Implementierungen zusammengeführt, sodass Algorithmus 4 verwirklicht werden kann. Der Agent erhält dabei die bearbeiteten Zustände der Umgebung, wählt auf dieser Grundlage unter Zuhilfenahme des Strategienetzwerkes eine Aktion aus und erhält erneut den bearbeiteten Zustand und die Belohnung. Diese Erfahrung wird anschließend zum Wiederholungsspeicher hinzugefügt.

Anders als im theoretischen Algorithmus wird nun nicht mehr nach jedem, sondern nur noch nach jedem vierten Schritt eine Mini-Batch-Aktualisierung durchgeführt. Hierzu werden, wie oben beschrieben, Erfahrungen mit Priorisierung aus dem Wiederholungsspeicher an den Agenten weitergegeben, der wiederum mithilfe der tiefen Q-Netzwerke die Zielwerte berechnet und die damit entstehenden Mini-Batches an das Strategienetzwerk zur Aktualisierung der Netzwerkparameter übergibt.

Hiermit ist auch die Beschreibung der Implementierung abgeschlossen (weitere Details sind in der Dokumentation der Implementierungen zu finden), womit nun schließlich die Durchführung von numerischen Experimenten möglich ist.

4.3 Ergebnisse

Die nun folgenden Experimente haben das Ziel zum einen den Effekt einzelner Parameterwerte und zum anderen den Effekt der verschiedenen in Kapitel 3 vorgestellten

Varianten des tiefen Q-Lernens auf den Lernerfolg des Agenten zu untersuchen. Bei der Bemessung des Lernerfolges spielen vor allem das letztendlich erreichte Gesamtergebnis sowie die Anzahl der Schritte, die für das Erreichen dieses Ergebnisses nötig waren, eine Rolle.

Die Experimente werden für die in Kapitel 4.1 vorgestellten Videospiele durchgeführt, wobei für jedes Spiel verschiedene Aspekte genauer untersucht werden.

Experimente im Spiel Pong

Der Fokus bei den Experimenten im Spiel *Pong* wird auf der isolierten Untersuchung des Effekts einzelner Parameter liegen. Hierbei bleiben, abgesehen vom jeweils betrachteten Parameter, alle anderen Einstellungen des Algorithmus gleich (siehe Anhang A für eine detaillierte Darstellung der verwendeten Parameterwerte).

Kapazität des Wiederholungsspeichers Da die Erfahrungswiederholung in [Mni+13] als eine der Hauptkomponenten für die Funktionsfähigkeit des tiefen Q-Lernens bei anspruchsvollen Problemen angeführt ist, wird der erste untersuchte Parameter die maximale Kapazität n_{cap} des Wiederholungsspeichers sein. Als mögliche Werte für die maximale Kapazität werden in diesem Experiment $n_{\text{cap}} = 2^5 = 32$, was gerade der Größe eines Mini-Batches entspricht, $n_{\text{cap}} = 2^{10} = 1024$, was in etwa der Länge einer Episode entspricht, sowie $n_{\text{cap}} = 2^{15} = 32768$ betrachtet.

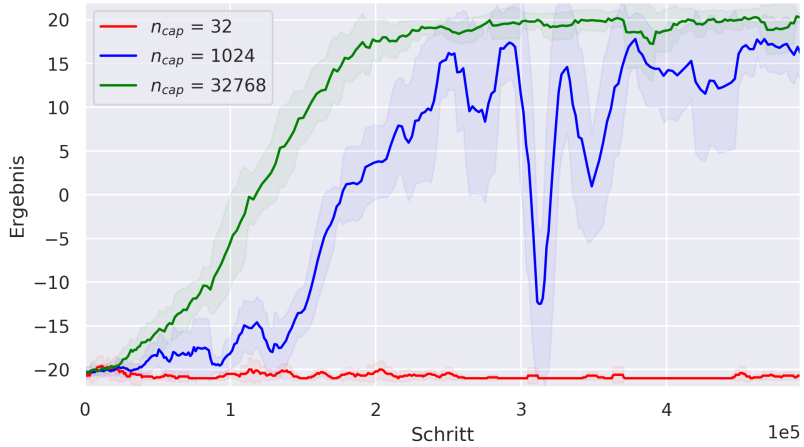


Abbildung 4.4: Lernkurven für verschiedene Größen des Wiederholungsspeichers

Bemerkung 4.1. In Abb. 4.4 sind die Lernkurven für die Durchführung des Experiments mit den genannten Speichergrößen zu sehen. Auf der x-Achse ist die Anzahl der Schritte des Agenten im Verlauf des Lernprozesses aufgetragen[8], auf der y-Achse die erreichten Endergebnisse der während des Lernprozesses gespielten Episoden. Die dargestellten Kurven basieren jeweils auf dem Durchschnitt der Lernkurven für zwei verschiedene zufällige Netzwerkinitialisierungen. Anschließend wurden die Lernkurven zur besseren Lesbarkeit durch die Verwendung des gleitenden Durchschnitts mit

[8]Mit der Anzahl der Schritte ist hierbei die Zahl der vom Agenten betrachteten Zustände und aktiv getroffenen Entscheidungen gemeint – im Gegensatz zur Gesamtanzahl der von der Umgebung generierten Bildern, die aufgrund der im vorherigen Unterkapitel beschriebenen Auslassung von Bildern n_{skip}-mal so groß ist.

einem Intervall von zehn Episoden geglättet. Die Standardabweichung innerhalb dieser Intervalle wird durch die schattierten Flächen dargestellt.

Zunächst ist aus Abb. 4.4 zu entnehmen, dass für die kleinste Speichergröße $n_{\text{cap}} = 32$ kein Lernfortschritt erkennbar ist. Da die Speichergröße in diesem Fall gerade so groß ist wie die Größe der Mini-Batches für die Parameteraktualisierungen, wird das Netzwerk ausschließlich mit den gerade gemachten Erfahrungen des Agenten trainiert. Die dabei auftretenden und bereits in Kapitel 3 beschriebenen Probleme der Korrelation zwischen aufeinander folgenden Erfahrungen und der Überanpassung der Parameter an gerade gemachte Erfahrungen sind also nicht nur in der Theorie sondern auch in der Praxis relevant. Obwohl für die nächstgrößere getestete Speicherkapazität $n_{\text{cap}} = 1024$ bereits ein Lernfortschritt zu erkennen ist und der Agent am Ende des Prozesses auch ein annähernd optimales Ergebnis liefert, ist Abb. 4.4 auch zu entnehmen, dass der Lernprozess auch für diese Kapazität noch sehr instabil ist, was sich in den vielen teils sehr starken Einbrüchen der Ergebnisse äußert. Diesen Einbrüchen liegt das sog. Phänomen des *Katastrophalen Vergessens* zugrunde, das bei zu klein gewählten Wiederholungsspeichern auftreten kann. Wie an der Kurve für $n_{\text{cap}} = 32768$, die einen schnellen Lernfortschritt und anschließend durchgehend nahe am Optimum verbleibende Ergebnisse zeigt, zu erkennen ist, kann dieses Problem durch Wiederholungsspeicher mit einer größeren Kapazität vermieden werden. Für alle weiteren Experimente wird daher die in der Literatur vorgeschlagene, nochmals deutlich größere Kapazität von etwa einer Millionen Erfahrungen (in dieser Arbeit $2^{20} = 1048576$) verwendet.

Anzahl der Schritte beim Mehrschrittfehler Im nächsten Experiment wird der Effekt verschiedener Anzahlen $n \in \{5, 10, 50\}$ an Schritten für die Berechnung der n-Schritt Zielwerte gemäß 3.30 untersucht. Die Ergebnisse hierfür sind in Abb. 4.5 dargestellt, wobei die Kurven genau wie in Bemerkung 4.1 beschrieben erstellt wurden, mit dem einzigen Unterschied, dass hier jeweils der Durchschnitt von drei zufälligen Netzwerkinitialisierungen berechnet wurde.

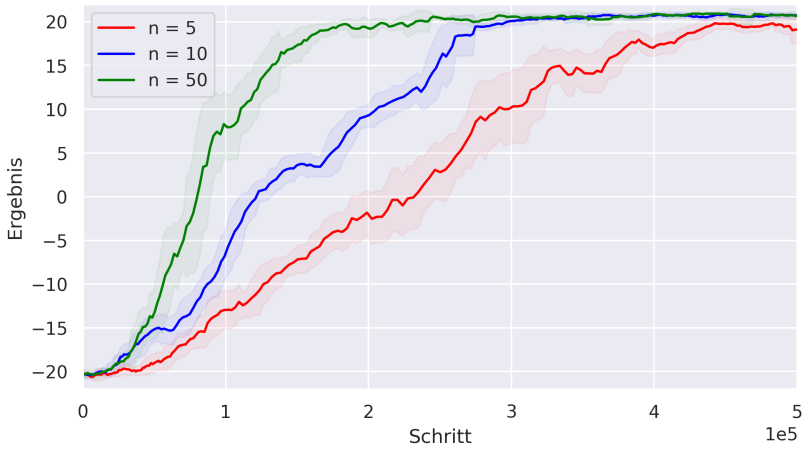


Abbildung 4.5: Lernkurven für verschiedene n-Schritt Fehler

Auch wenn der Agent für alle betrachteten Werte von n eine annähernd optimale Lösung für das Spiel *Pong* findet, so sind in Abb. 4.5 dennoch Unterschiede vor allem

im Hinblick auf die Anzahl der für das Finden der Lösung benötigten Schritte zu beobachten. So hat der Agent für $n = 50$ etwa $2 \cdot 10^5$ Schritte gebraucht um eine gute Lösung zu finden, für $n = 10$ etwa $3 \cdot 10^5$ und für $n = 5$ bereits über $4 \cdot 10^5$ Schritte. Im Spiel *Pong* kann der Mehrschrittfehler den Lernprozess demnach beschleunigen, was – wie in Kapitel 3.5 erwähnt – daran liegt, dass durch den n-Schritt Fehler eine erhaltene Belohnung in die Berechnung Zielwerte für n verschiedene Zustände mit eingeht. Dies bedeutet jedoch nicht, dass eine beliebige Erhöhung von n zwangsläufig zu einem besseren Ergebnis führt, da für zu große Werte von n mehrere Belohnungen zusammengefasst werden, was – im Fall von negativen und positiven Belohnungen – zu Aufhebungseffekten führen kann.

Regularisierung Neben dem Mehrschrittfehler wurde in Kapitel 3.5 auch der Regularisierungsfehler eingeführt. Das Gewicht λ_{L2}, mit welchem dieser Fehler in den Gesamtfehler eingeht, ist Bestandteil der nächsten Untersuchung. Ausgehend vom in [Hes+18] vorgeschlagenen Wert $\lambda_{L2} = 10^{-5}$ werden zusätzlich die Werte $\lambda_{L2} = 10^{-4}$, $\lambda_{L2} = 10^{-6}$ sowie $\lambda_{L2} = 0$ betrachtet, wobei beim letzten Wert die Ergebnisse des Experiments zum Mehrschrittfehler für $n = 10$ wiederverwendet werden.

Da es einer der Hauptgründe für die Einführung des Regularisierungsfehlers war, möglichst viele der Netzwerkparameter nahe bei Null zu halten, wird zunächst anhand von Abb. 4.6 visuell überprüft, ob dieses Ziel erreicht wurde.

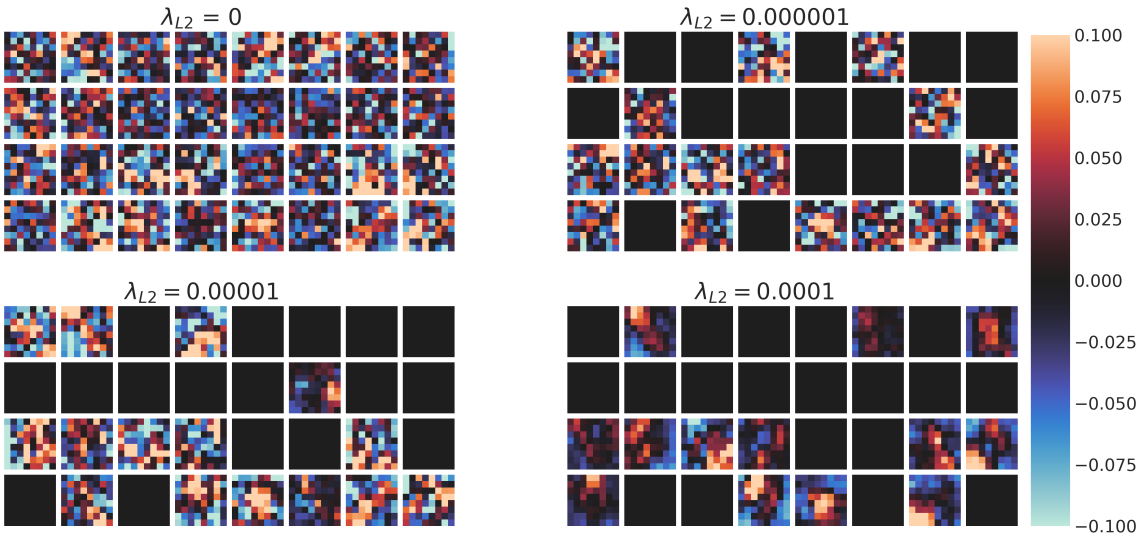


Abbildung 4.6: Filtermatrizen der ersten Faltungsschicht für verschiedene Regularisierungsgewichtungen

In Abb. 4.6 sind die Gewichtungsmatrizen (im Zusammenhang mit faltenden neuronalen Netzwerken auch *Filtermatrizen* genannt) der ersten Schicht der trainierten Netzwerke zu sehen. Je kleiner der Betrag eines Eintrags der jeweils 32 Matrizen mit Dimension 8×8 ist, desto dunkler ist das entsprechende Pixel in der Abbildung.

Der Abbildung ist zum einen zu entnehmen, dass für $\lambda_{L2} > 0$ jeweils mindestens die Hälfte der Filtermatrizen der ersten Schicht annähernd der Nullmatrix entsprechen, während dies bei $\lambda_{L2} = 0$ für keine einzige Filtermatrix der Fall ist. Zum anderen sind in den Matrizen für steigende Werte von λ_{L2} klarere Strukturen sowie weniger Rauschen zu erkennen.

Das Ziel der Eliminierung von Netzwerkparametern durch den Regularisierungsfehler ist damit also erreicht. Diese Eliminierung der Netzwerkparameter ist jedoch nur dann sinnvoll, wenn die Leistung des Agenten nicht darunter leidet, was in Abb. 4.7 untersucht wird.

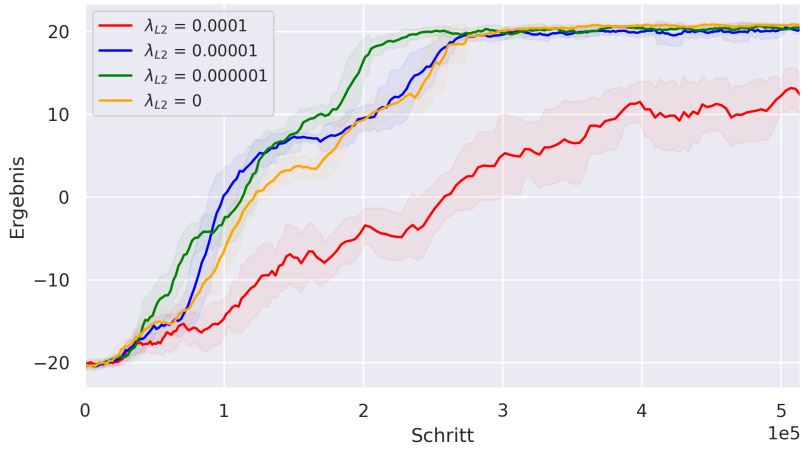


Abbildung 4.7: Lernkurven für verschiedene Regularisierungsgewichtungen

Die in Abb. 4.7 dargestellten und erneut gemäß Bemerkung 4.1 mit drei zufälligen Netzwerkinitialisierungen erstellten Lernkurven zeigen, dass mit $\lambda_{L2} = 10^{-6}$ sowie $\lambda_{L2} = 10^{-5}$ ähnliche Ergebnisse erzielt werden wie mit $\lambda_{L2} = 0$. Mit $\lambda_{L2} = 10^{-4}$ findet der Agent jedoch nur noch eine suboptimale Lösung, was darauf schließen lässt, dass in diesem Fall zu viele Netzwerkparameter eliminiert wurden.

Aktualisierungsfrequenz des Zielnetzwerkes Die letzte für das Spiel *Pong* untersuchte Größe ist die Anzahl $\tau \in \mathbb{N}$ an Schritten, nach denen die Parameter $\widetilde{\boldsymbol{\theta}}$ des Zielnetzwerkes mit den Parametern $\boldsymbol{\theta}$ des Strategienetzwerkes überschrieben werden. Betrachtet werden die Werte $\tau = 1$ – effektiv wird hierbei also nur das Strategienetzwerk verwendet – sowie der in der Literatur meist verwendete Wert $\tau = 10000$ und $\tau = 20000$.

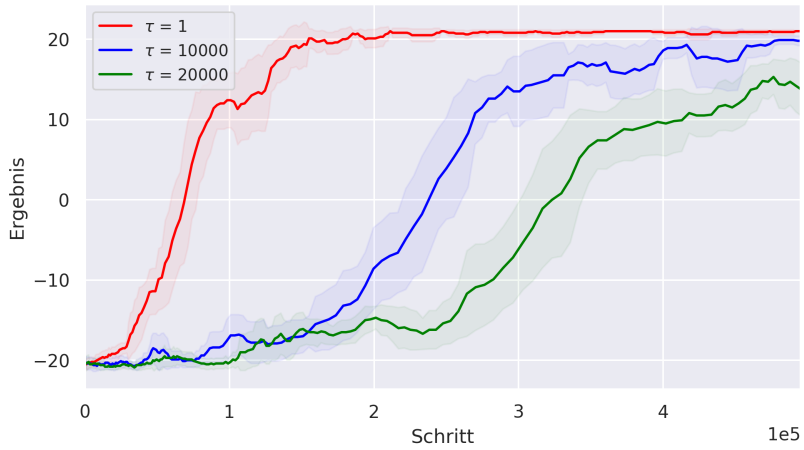


Abbildung 4.8: Lernkurven für verschiedene Aktualisierungsfrequenzen

Abb. 4.8 zeigt nun einerseits einen deutlich früheren Anstieg der Ergebnisse für

$\tau = 1$ und andererseits auch bessere Ergebnisse zum Ende des Trainings als bei den Durchführungen des Experiments mit $\tau = 10000$ und $\tau = 20000$. Die in Kapitel 3 vorgestellten Erweiterungen – in diesem Fall die Verwendung von Ziel- und Strategienetzwerk zur Berechnung der Zielwerte – können also auch einen negativen Einfluss auf den Lernprozess haben.

Da *Pong* als eines der für Agenten des bestärkenden Lernen am einfachsten zu erlernenden Spiele der ALE gilt, für das wie in [Mni+13] gezeigt und in den Experimenten bestätigt auch ohne Erweiterungen bereits annähernd optimale Ergebnisse erreicht werden können, wird in den folgenden Experimenten der Effekt der einzelnen Erweiterungen am deutlich anspruchsvolleren Spiel *Enduro* genauer untersucht.

Experimente im Spiel Enduro

Im Verlauf der Jahre haben sich in der Literatur, die sich mit dem tiefen Q-Lernen angewendet auf *Atari 2600*-Spiele beschäftigt, einerseits immer mehr Erweiterungen mit dem Ziel der Verbesserung der Algorithmen, andererseits aber auch immer besser passende Parameterwahlen entwickelt. So werden in der Literatur oft Erweiterungen mit speziell darauf angepassten Parameterwerten eingeführt (vgl. zum Beispiel [Sch+15, Kap. B.2.2]).

Verschiedene Lernraten für tiefes Q-Lernen mit und ohne Erweiterungen Einer der sensibelsten Parameter ist hierbei die Lernrate η, für die in der ursprünglichen Publikation [Mni+13] der Wert $\eta = 0,00025$ gebraucht wird, während aktuellere Publikationen meist ein Viertel dieser Lernrate verwenden, d.h $\eta = 0,0000625$. Als Ergänzung wird daher im folgenden Experiment am Beispiel des Spiels *Enduro* der Effekt dieser beiden Lernraten zum einen auf das tiefe Q-Lernen ohne Erweiterungen[9] untersucht sowie zum anderen auf das tiefe Q-Lernen mit den in Kapitel 3 vorgestellten Erweiterungen, d.h. mit Priorisierung, Mehrschrittfehler, duellierender Netzwerkarchitektur und Doppel-Q-Lernen.

In Tabelle 4.1 bzw. Abb. 4.9 sind nun die Endergebnisse bzw. Lernkurven für die Durchführung des Experiments mit den genannten Erweiterungen (in rot) und ohne diese (in blau) sowie mit der ursprünglichen Lernrate $\eta = 0,00025$ (dunkel) und der angepassten Lernrate $\eta = 0,0000625$ (hell) dargestellt. Als Endergebnis wird in diesem Zusammenhang der maximale Durchschnitt der Ergebnisse von 50 aufeinander folgenden Episoden während des Trainings bezeichnet. Die Berechnung der Lernkurven erfolgte ähnlich, wie in 4.1 beschrieben. Die Lernkurven sind hier jedoch nicht über zehn, sondern über 50 Episoden geglättet. Außerdem war es aufgrund der im Spiel *Enduro* deutlich größeren Anzahl an benötigten Schritten (ca. zehn Millionen im Vergleich zu 500 Tausend im Spiel *Pong*) nicht mehr möglich, die Experimente für mehrere verschiedene Netzwerkinitialisierungen durchzuführen.

Zunächst ist der zweiten Spalte von Tabelle 4.1 sowie den dunklen Kurven in Abb. 4.9 zu entnehmen, dass die Erweiterungen im Fall von $\eta = 0,00025$ zu besseren Ergebnissen führen. Ebenso ist anhand der hellen Kurven bzw. der Zeilen in Tabelle 4.1 zu erkennen, dass die Verringerung der Lernrate einen positiven Effekt auf das

[9] Auch wenn es hier zur Abgrenzung als tiefes Q-Lernen ohne Erweiterungen bezeichnet wird, so werden dennoch die in [Mni+13] eingeführten Techniken der Erfahrungswiederholung und der Trennung von Ziel- und Strategienetzwerk gemäß 3.22 verwendet.

Ergebnisse	$\eta = 0,00025$	$\eta = 0,0000625$
mit Erweiterungen	898,36	1112,42
ohne Erweiterungen	610,5	1268,44

Tabelle 4.1: Ergebnisse für das tiefe Q-Lernen mit und ohne Erweiterungen sowie mit verschiedenen Lernraten

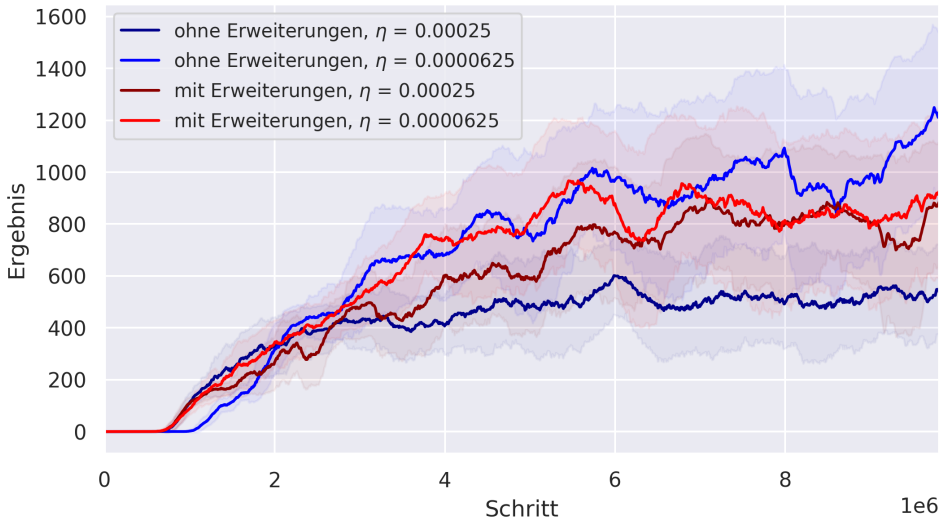


Abbildung 4.9: Lernkurven für das tiefe Q-Lernen mit und ohne Erweiterungen sowie jeweils verschiedenen Lernraten

tiefe Q-Lernen sowohl mit als auch ohne Erweiterungen hat. Während sich hierbei jedoch beim ursprünglichen Verfahren das Endergebnis ungefähr verdoppelt, fällt die Steigerung beim erweiterten Algorithmus weniger deutlich aus. Dies hat zur Folge, dass für $\eta = 0,0000625$ das tiefe Q-Lernen ohne Erweiterungen sogar bessere Ergebnisse liefert als mit Erweiterungen (helle Kurven in Abb. 4.9 bzw. dritte Spalte in Tabelle 4.1).

Weglassen von Erweiterungen Nachdem im vorherigen Experiment der Effekt der Erweiterungen als Gesamtheit untersucht wurde, zielt das nächste Experiment auf die Untersuchung des Einflusses der einzelnen Erweiterungen ab. Hierzu wird das tiefe Q-Lernen mit jeweils drei der vier Erweiterungen (Priorisierung, Mehrschrittfehler, duellierende Netzwerkarchitektur und Doppel-Q-Lernen) ausgeführt, während die verbleibende ausgelassen wird.

Die bei der Durchführung dieses Experiments entstandenen und wie zuvor berechneten Ergebnisse und Lernkurven sind in Tabelle 4.2 und Abb. 4.10 dargestellt.

Abb. 4.10 und Tabelle 4.2 ist zu entnehmen, dass sowohl das Weglassen der Priorisierung als auch des Mehrschrittfehlers keinen negativen Einfluss auf die maximal erreichten Ergebnisse hat. Während das Ausklammern des Mehrschrittfehlers (blaue Kurve) jedoch zum einen zu einem langsameren Fortschritt zu Beginn und zum anderen zu einem Leistungseinbruch am Ende des Lernprozesses führt, treten diese Effekte beim Weglassen der Priorisierung (rote Kurve) nicht auf. Wie bereits in den Experimenten im Spiel *Pong* hat die Verwendung des Mehrschrittfehlers also auch

Weggelassene Erweiterung	Ergebnis
Priorisierung	1149,9
Mehrschrittfehler	1125,68
Duellierende Netzwerkarchitektur	988,2
Doppel Q-Lernen	840,28

Tabelle 4.2: Ergebnisse für das Weglassen einzelner Erweiterungen

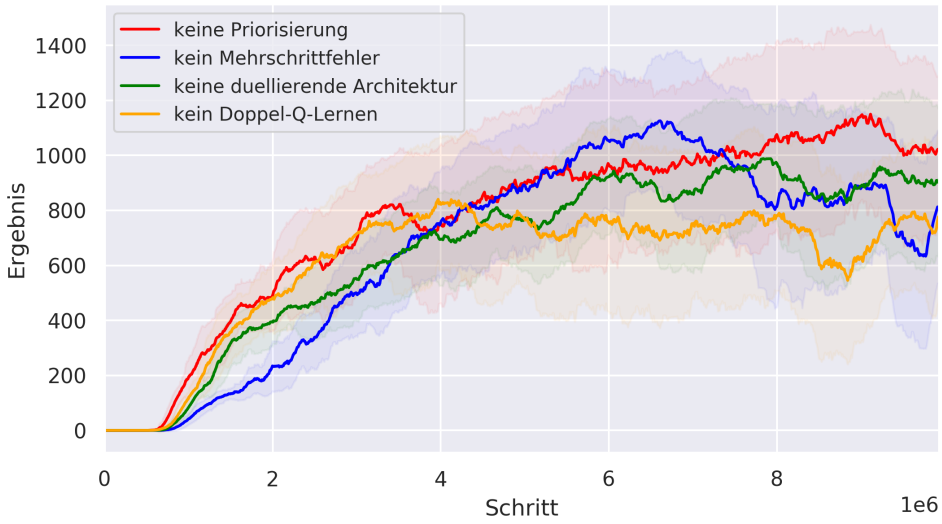


Abbildung 4.10: Durch das Weglassen verschiedener Erweiterungen entstandene Lernkurven

im Spiel *Enduro* eine beschleunigende Wirkung auf den Lernprozess. Im Gegensatz zu den beiden genannten Erweiterungen führt die Auslassung der duellierenden Netzwerkarchitektur (grüne Kurve) und vor allem des Doppel-Q-Lernens (gelbe Kurve) zu teils deutlich niedrigeren Ergebnissen.

Experimente im Spiel Montezuma's Revenge

Abschließend folgen nun Experimente im Spiel *Montezuma's Revenge*, das wie bereits erwähnt als eines der anspruchsvollsten Spiele der ALE für Agenten des bestärkenden Lernens gilt, sodass mit dem tiefen Q-Lernen mit Priorisierung, duellierender Netzwerkarchitektur und Doppel-Q-Netzwerken (kurz PDD DQN) keine geeignete Lösung gefunden wird. Deshalb wird in den folgenden Experimenten der in 4.1 vorgestellte Datensatz mit Demonstrationen für das Spiel *Montezuma's Revenge* verwendet um einzelne Komponenten des in Kapitel 3.6 vorgestellten Expertenfehlers und des darauf aufbauenden tiefen Q-Lernens mit Demonstrationen (kurz DQfD) zu untersuchen.

Fehlergewichtungen Im ersten Experiment werden hierbei verschiedene Gewichtungen des ein-Schritt-Fehlers λ_{DDQN}, des n-Schritt-Fehlers λ_n und des Experten-Fehlers λ_E betrachtet. In Abb. 4.11 sind die Lernkurven, die wie zuvor auf der Durchführung mit einer Netzwerkinitialisierung beruhen und über 50 Episoden geglättet wurden, für die Durchführung des Experiments mit unterschiedlichen Fehler-

gewichtungen dargestellt. Konkret werden hierbei die Fehler im Verhältnis $1:1:1$ – d.h. $(\lambda_{DDQN}, \lambda_n, \lambda_E) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ – sowie $1:1:3$ – d.h. $(\lambda_{DDQN}, \lambda_n, \lambda_E) = (\frac{1}{5}, \frac{1}{5}, \frac{3}{5})$ und $3:3:1$ – d.h. $(\lambda_{DDQN}, \lambda_n, \lambda_E) = (\frac{3}{7}, \frac{3}{7}, \frac{1}{7})$ – betrachtet.

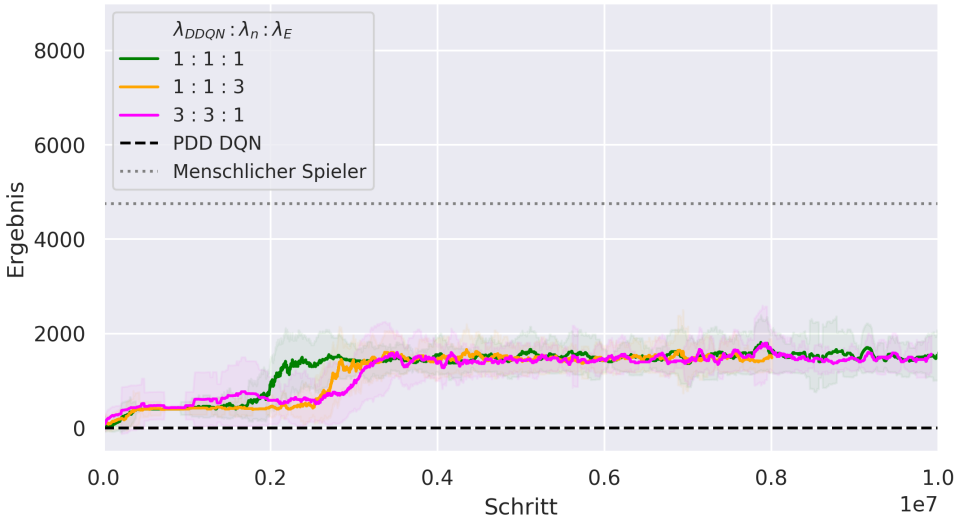


Abbildung 4.11: Lernkurven für verschiedene Gewichtungen der Fehler λ_{DDQN}, λ_n und λ_E

Aus Abb. 4.11 lässt sich ablesen, dass zum Ende des Trainings für alle betrachteten Gewichtungen in etwa dasselbe Niveau erreicht wird, während dieses Niveau jedoch für die Gewichtung $1:1:1$ am schnellsten erreicht wird. Insgesamt liegen die erreichten Ergebnisse zwar über denen des Verfahrens ohne Verwendung von Demonstrationen, aber immer noch deutlich unter denen eines menschlichen Spielers.

Experten-Klassifikationskoeffizient Ein weiterer Parameter im Zusammenhang mit dem tiefen Q-Lernen mit Demonstrationen ist der Experten-Klassifikationskoeffizient c_E, der gemäß Gleichung 3.32 den Betrag bestimmt, mit dem die Aktionswerte der vom Demonstrator ausgewählten Aktion höher eingeschätzt werden. Je höher dieser Koeffizient also gewählt ist, desto eher wird der Agent die Strategie des Demonstrators imitieren.

Als mögliche Werte werden im folgenden Experiment der in [Hes+18] vorgeschlagene Wert $c_E = 0.8$ sowie $c_E = 0.2$ betrachtet. Die mit diesen Parameterwahlen generierten und wie im vorherigen Experiment berechneten Lernkurven sind in Abb. 4.12 zu sehen, wobei für $c_E = 0.8$ die Daten aus dem vorherigen Experiment für das Verhältnis $1:1:1$ wiederverwendet werden.

Abb. 4.12 ist zu entnehmen, dass sich der Lernprozess zu Beginn für beide Werte $c_E = 0.8$ und $c_E = 0.2$ ähnlich gestaltet. So wird in beiden Fällen nach ca. zwei Millionen Schritten mit Ergebnissen von ungefähr 1600 Punkten dasselbe Niveau wie im vorherigen Experiment zu den Fehlergewichtungen erreicht. Im Gegensatz zum Verfahren mit dem Wert $c_E = 0.8$, bei dem auch nach 10 Millionen Schritten keine weitere Steigerung der Ergebnisse mehr erfolgt[10], steigen die Ergebnisse für das

[10]Da in [Hes+18] mit dem Verfahren für $c_E = 0.8$ nach 50 Millionen Trainingsschritten ein Ergebnis von ca. 4500 erreicht wird, besteht die Vermutung, dass auch hier für den Wert $c_E = 0.8$ nach weiteren Trainingsschritten eine Steigerung hätte stattfinden können.

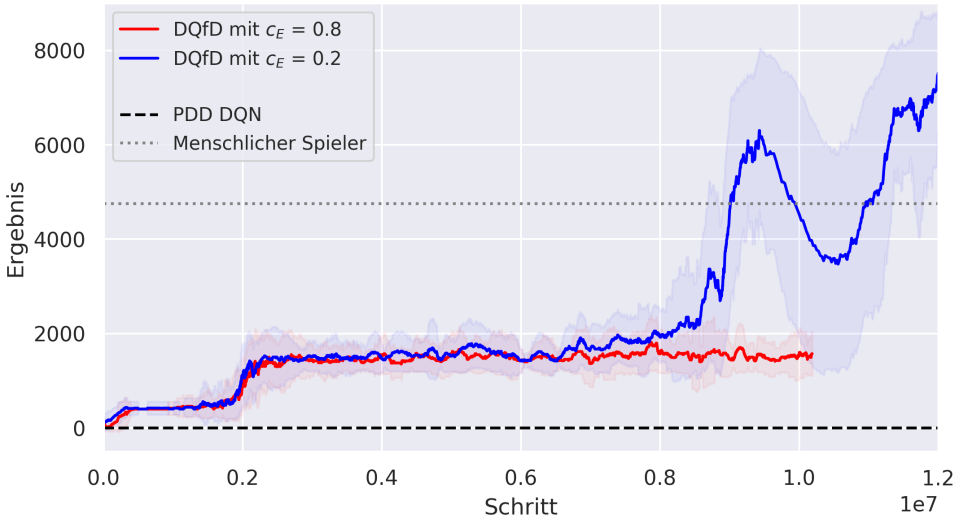


Abbildung 4.12: Lernkurven für verschiedene Experten-Klassifikationskoeffizienten

Verfahren mit $c_E = 0.2$ ab ca. 8 Millionen Schritten auf über 7000 Punkte nach 12 Millionen Schritten an und überschreiten damit sogar die in [Wan+16] angegebenen durchschnittlich etwa 4753 Punkte eines menschlichen Spielers.

Ein möglicher Grund für die besseren Ergebnisse mit dem geringeren Experten-Klassifikationskoeffizienten liegt in der Beschaffenheit des Spiels *Montezuma's Revenge* und des entsprechenden Datensatzes. So gibt es in dem Spiel, wie bereits erwähnt, zum einen mehrere verschiedene Räume. Zum anderen gibt es aber auch mehrere verschiedene Level, wobei in einem Raum in verschiedenen Levels verschiedene Aktionen notwendig sein können. Während der Agent im gesamten Lernprozess das zweite Level nicht erreicht, finden viele Demonstrationen in höheren Levels statt. Ein zu hoch gewählter Koeffizient c_E könnte dabei möglicherweise zu einer Überanpassung des Agenten an Aktionen führen, die in höheren Levels nötig wären, während sich der Agent dagegen noch im ersten Level befindet.

Nachdem hiermit das letzte Experiment vorgestellt ist, folgt im nächsten Kapitel abschließend eine Diskussion und Einordnung der Ergebnisse sowie ein Ausblick auf aktuelle und zukünftige Entwicklungen im bestärkenden Lernen, die in dieser Arbeit nicht behandelt wurden.

Kapitel 5

Diskussion

Hinsichtlich der Einordnung der Experimente ist zunächst zu beachten, dass die in dieser Arbeit erzielten Ergebnisse weniger verlässlich sind als die in der Literatur (vgl. [Mni+15], [Sch+15], [VGS16], [Wan+16], [Hes+18]) veröffentlichten Resultate. Dies hat im Wesentlichen die drei folgenden Gründe: Erstens wurden die Experimente in der Literatur für eine Menge von etwa 50 Spielen der ALE durchgeführt, während in dieser Arbeit nur eine Auswahl von drei Spielen, deren Anforderungen an den Agenten aufgrund der großen Unterschiede der ausgewählten Spiele dennoch sehr vielfältig sind, betrachtet wurde. Zweitens wurden die Experimente in der Literatur für jedes dieser Spiele mit mehreren zufälligen Netzwerkinitialisierungen durchgeführt, was hier aufgrund des großen Rechenaufwandes der beiden anderen Spiele nur im Spiel *Pong* möglich war. Drittens wurden die Agenten hier nur für zehn Millionen – bzw. 500 Tausend im Spiel *Pong* – Schritte trainiert, während sie in der Literatur in jedem Spiel für 50 Millionen Schritte trainiert wurden. Unterschiede in den Ergebnissen können zudem auf in den Details möglicherweise verschiedene Implementierungen und Parameterwahlen zurückgehen. So wurde in dieser Arbeit zum Beispiel die Epsilon-Annäherungsstrategie an die geringere Anzahl an Trainingsschritten angepasst.

Trotzdem konnten für alle drei getesteten Spiele in jeweils mindestens einem Experiment Ergebnisse erzielt werden, die gegenüber den in [Wan+16] angegeben Ergebnissen menschlicher Spieler gleichwertig oder gar besser sind. Zudem sind die erzielten Ergebnisse vergleichbar mit denen der oben zitierten Literatur. Als Ergänzung zu den in der Literatur oft nur für eine spezielle Parametereinstellung durchgeführten Experimenten, konnten durch zusätzliche Experimente außerdem die Wirkungsweisen einzelner Parameter und Varianten des tiefen Q-Lernens veranschaulicht werden. Insbesondere durch die Verwendung von Demonstrationen wurden im Spiel *Montezuma's Revenge*, in dem Agenten des bestärkenden Lernens für gewöhnlich versagen, Ergebnisse erzielt, die mit denen menschlicher Spieler vergleichbar sind.

Diskutabel ist hierbei sicherlich, ob die Verwendung von Demonstrationen dem Paradigma des bestärkenden Lernens widerspricht, dass dem Agenten, abgesehen von seinen eigenen durch die Interaktion mit seiner Umgebung gemachten Erfahrungen, keine zusätzlichen Informationen zur Verfügung stehen dürfen. Andererseits ließe sich argumentieren, dass auch Demonstratoren, die gerade dasselbe Problem wie der Agent zu lösen versuchen, Teil der Umgebung sein können.

Unabhängig davon, ob man die Verwendung von Demonstrationen im bestärkenden Lernen als zulässig erachtet oder nicht, ist die Suche nach weiteren Verbesserungen

von Verfahren des bestärkenden Lernens dennoch sinnvoll. Aktuelle Ansätze betreffen hierbei zum Beispiel die Handhabung des in Bemerkung 3.7 bereits angesprochenen Dilemmas von Exploration und Exploitation. Wenngleich auch in der aktuellen Zeit schon in vielen Fällen Algorithmen des bestärkenden Lernens erfolgreich in Problemen der realen Welt eingesetzt werden, ist es dennoch eine der großen aktuellen Herausforderungen, das bestärkende Lernen für reale Probleme noch praktikabler zu machen. Hierzu müssten entweder die Verfahren an das Training in der realen Welt angepasst werden oder es müsste eine ausreichend realitätsgetreue Simulation des realen Problems entwickelt werden, sodass ein in der Simulation trainierter Agent auch in der realen Welt anwendbar ist. Für Ersteres wäre es insbesondere wichtig, die Anzahl der benötigten Trainingsschritte zu reduzieren. Eine Möglichkeit dieses Problem anzugehen liegt in den Verfahren des sog. *Transferlernens*, indem in ähnlichen Problemen bereits generiertes Wissen des Agenten auf neue Probleme übertragen wird. Eine andere Möglichkeit besteht jedoch auch in der in Kapitel 3.6 dieser Arbeit vorgestellten Technik des Vortrainings auf Grundlage von Demonstrationen.

Anhang A

Standardparameter bei den durchgeführten Experimenten

Im Folgenden sind die verwendeten Standardparameterwerte für die in Kapitel 4 durchgeführten Experimente aufgetragen. Die Standardparameter blieben hierbei innerhalb der Experimente für die einzelnen Spiele – abgesehen vom im jeweiligen Experiment betrachteten Parameter – unverändert. Bis auf kleine Anpassungen zur Beschleunigung des Lernprozesses sind die Parameterwerte dabei – sofern angegeben – an den in der Literatur verwendeten Werten orientiert.

In Abbildung A.1 sind zunächst die Parameter dargestellt, die für alle durchgeführten Experimente gleichbleibend sind.

Parameter	**Wert**	**Beschreibung**
n_{skip}	4	Anzahl der Schritte, für die gewählte Aktionen wiederholt und Bilder ausgelassen werden
n_{stack}	4	Anzahl der zu einem Zustand zusammengefassten Bilder
n_{Batch}	32	Anzahl der für die Mini-Batch-Aktualisierungen verwendeten Erfahrungen
n_{prefill}	50000	Anzahl zufälliger Übergänge, mit denen der Wiederholungsspeicher initialisiert wird
n_{learn}	4	Anzahl der Schritte nach denen jeweils eine Mini-Batch-Aktualisierung durchgeführt wird
γ	0,99	Abnahmefaktor des modellierten Markov-Entscheidungsprozesses

Tabelle A.1: Über alle Experimente gleichbleibende Parameter

A.1 Standardparameter für das Spiel Pong

Da Agenten des bestärkenden Lernens im Spiel *Pong* nach vergleichsweise wenigen Schritten bereits gute Lösungen finden und die Leistung des Agenten durch eine

zu hohe Explorationsrate am Ende des Trainings beeinträchtigt wird, wurde hier die in der Literatur vorgeschlagene lineare ϵ-Annäherungsstrategie von $\epsilon = 1$ zu Beginn $\epsilon = 0,1$ nach einer Million Schritten folgendermaßen angepasst. Zunächst wird ϵ linear über 50 Tausend Schritte von 1 auf 0,1 angenähert. Anschließend wird ϵ nochmals jeweils über 100 Tausend Schritte linear auf 0,01 und 0,001 gesenkt. Die weiteren Parameter im Spiel *Pong* wurden wie folgt gewählt[1]:

Parameter	**Wert**	**Beschreibung**
n_{cap}	2^{20}	maximale Kapazität des Wiederholungsspeichers
α	/	Priorisierungskoeffizient
β_0	/	Anfangswert für den Priorisierungsausgleichskoeffizient, der im Verlauf des Trainings an 1 angenähert wird
ϵ_{Prio}	0,001	Wert, der bei der Berechnung der proportionalen Prioritäten hinzugefügt wird
η	0,00025	Lernrate
τ	10000	Anzahl der Schritte nach denen die Zielnetzwerkparameter überschrieben werden
n	10	Anzahl der für die Berechnung des Mehrschrittfehlers betrachteten Schritte
λ_{DDQN}	0,5	Gewichtung des ein-Schritt Fehlers
λ_n	0,5	Gewichtung des n-Schritt Fehlers
λ_E	0,0	Gewichtung des Experten-Fehlers
λ_{L2}	0,0	Gewichtung des Regularisierungsfehlers

Tabelle A.2: Parameter im Spiel *Pong*

[1]Für die Experimente im Spiel *Pong* wurde eine ältere Implementierung verwendet, bei der weder die Koeffizienten α und β noch die Ausgleichsgewichte gemäß 3.21 enthalten waren.

A.2 Standardparameter für das Spiel Enduro

In den Spiel *Enduro* wird nun wieder die langsamere ϵ-Annäherung über eine Million Schritte von 1,0 auf 0,1 und anschließend über jeweils fünf Millionen Schritte von 0,1 auf 0,01 und 0,001 verwendet. Zusätzlich wird als Standardlernrate der angepasste Wert $\eta = 0,0000625$ verwendet:

Parameter	**Wert**	**Beschreibung**
n_{cap}	2^{20}	maximale Kapazität des Wiederholungsspeichers
α	0,6	Priorisierungskoeffizient
β_0	0,4	Anfangswert für den Priorisierungsausgleichskoeffizient, der im Verlauf des Trainings an 1 angenähert wird
ϵ_{Prio}	0,001	Wert, der bei der Berechnung der proportionalen Prioritäten hinzugefügt wird
η	0,0000625	Lernrate
τ	10000	Anzahl der Schritte nach denen die Zielnetzwerkparameter überschrieben werden
n	10	Anzahl der für die Berechnung des Mehrschrittfehlers betrachteten Schritte
λ_{DDQN}	0,5	Gewichtung des ein-Schritt Fehlers
λ_n	0,5	Gewichtung des n-Schritt Fehlers
λ_E	0,0	Gewichtung des Experten-Fehlers
λ_{L2}	0,0	Gewichtung des Regularisierungsfehlers

Tabelle A.3: Parameter im Spiel *Enduro*

A.3 Standardparameter für das Spiel Montezuma's Revenge

Im Spiel *Montezuma's Revenge* wird erneut die ϵ-Annäherung angepasst. Da der Agent hier durch das Vortraining auf den Demonstrationsdaten bereits Vorwissen erlangt hat, ist dabei zu Beginn des Trainings keine so große Explorationsrate mehr nötig, weshalb ϵ zunächst über 250 Tausend Schritte von 0,25 auf 0,1 und anschließend wieder über jeweils 5 Millionen Schritte von 0,1 auf 0,01 und 0,001 linear angenähert wird. Folgende weitere Standardparameter wurden für die Experimente im Spiel *Montezuma's Revenge* verwendet:

Parameter	**Wert**	**Beschreibung**
n_{cap}	2^{20}	maximale Kapazität des Wiederholungsspeichers
α	0,6	Priorisierungskoeffizient
β_0	0,4	Anfangswert für den Priorisierungsausgleichskoeffizient, der im Verlauf des Trainings an 1 angenähert wird
ϵ_{Prio}	0,001	Wert, der bei der Berechnung der proportionalen Prioritäten eigener Erfahrungen hinzugefügt wird
$\epsilon_{\text{Prio, Expert}}$	1,0	Wert, der bei der Berechnung der proportionalen Prioritäten von Erfahrungen des Experten hinzugefügt wird
η	0,0000625	Lernrate
τ	10000	Anzahl der Schritte nach denen die Zielnetzwerkparameter überschrieben werden
n	10	Anzahl der für die Berechnung des Mehrschrittfehlers betrachteten Schritte
c_E	0,8	Experten-Klassifikationskoeffizient
λ_{DDQN}	$\frac{1}{3}$	Gewichtung des ein-Schritt Fehlers
λ_n	$\frac{1}{3}$	Gewichtung des n-Schritt Fehlers
λ_E	$\frac{1}{3}$	Gewichtung des Experten-Fehlers
λ_{L2}	10^{-5}	Gewichtung des Regularisierungsfehlers

Tabelle A.4: Parameter im Spiel *Montezuma's Revenge*

Literatur

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ros58] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), S. 386.

[Ski65] Burrhus Frederic Skinner. *Science and human behavior*. 92904. Simon und Schuster, 1965.

[PS78] Martin L Puterman und Moon Chirl Shin. „Modified policy iteration algorithms for discounted Markov decision problems“. In: *Management Science* 24.11 (1978), S. 1127–1137.

[Ban+86] Albert Bandura u. a. „Social foundations of thought and action“. In: (1986).

[RHW86] David E Rumelhart, Geoffrey E Hinton und Ronald J Williams. „Learning representations by back-propagating errors“. In: *nature* 323.6088 (1986), S. 533–536.

[Wat89] Christopher John Cornish Hellaby Watkins. „Learning from delayed rewards“. Diss. King's College, Cambridge, Mai 1989.

[Hub92] Peter J Huber. „Robust estimation of a location parameter“. In: *Breakthroughs in statistics*. Springer, 1992, S. 492–518.

[Lin92] Long-Ji Lin. „Self-improving reactive agents based on reinforcement learning, planning and teaching“. In: *Machine learning* 8.3-4 (1992), S. 293–321.

[WD92] Christopher JCH Watkins und Peter Dayan. „Q-learning“. In: *Machine learning* 8.3-4 (1992), S. 279–292.

[TS93] Sebastian Thrun und Anton Schwartz. „Issues in using function approximation for reinforcement learning“. In: *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*. 1993.

[Tsi94] John N Tsitsiklis. „Asynchronous stochastic approximation and Q-learning“. In: *Machine learning* 16.3 (1994), S. 185–202.

[LB+95] Yann LeCun, Yoshua Bengio u. a. „Convolutional networks for images, speech, and time series“. In: *The handbook of brain theory and neural networks* 3361.10 (1995).

[Tes95] Gerald Tesauro. „Temporal difference learning and TD-Gammon“. In: *Communications of the ACM* 38.3 (1995), S. 58–68.

[TV97] John N Tsitsiklis und Benjamin Van Roy. „Analysis of temporal-diffference learning with function approximation“. In: *Advances in neural information processing systems*. 1997, S. 1075–1081.

[Sch99] Stefan Schaal. „Is imitation learning the route to humanoid robots?“ In: *Trends in cognitive sciences* 3.6 (1999), S. 233–242.

[PSD01] Doina Precup, Richard S Sutton und Sanjoy Dasgupta. „Off-policy temporal-difference learning with function approximation“. In: *ICML*. 2001, S. 417–424.

[BS07] Dimitri P Bertsekas und Steven E Shreve. *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 2007.

[GS09] Alex Graves und Jürgen Schmidhuber. „Offline handwriting recognition with multidimensional recurrent neural networks“. In: *Advances in neural information processing systems*. 2009, S. 545–552.

[RGB11] Stéphane Ross, Geoffrey Gordon und Drew Bagnell. „A reduction of imitation learning and structured prediction to no-regret online learning“. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, S. 627–635.

[KT12] Hans-Otto Karnath und Peter Thier. *Kognitive Neurowissenschaften*. Springer, 2012.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, S. 1097–1105.

[Bel+13] Marc G Bellemare, Yavar Naddaf, Joel Veness und Michael Bowling. „The arcade learning environment: An evaluation platform for general agents“. In: *Journal of Artificial Intelligence Research* 47 (2013), S. 253–279.

[Mni+13] Volodymyr Mnih u. a. „Playing atari with deep reinforcement learning“. In: *arXiv preprint arXiv:1312.5602* (2013).

[Cho15] François Chollet. *Keras*. `https://github.com/fchollet/keras`. 2015.

[KB15] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Hrsg. von Yoshua Bengio und Yann LeCun. 2015.

[Mar+15] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[Mni+15] Volodymyr Mnih u. a. „Human-level control through deep reinforcement learning“. In: *Nature* 518.7540 (2015), S. 529–533.

[Sch+15] Tom Schaul, John Quan, Ioannis Antonoglou und David Silver. „Prioritized experience replay“. In: *arXiv preprint arXiv:1511.05952* (2015).

[Bro+16] Greg Brockman u. a. „Openai gym“. In: *arXiv preprint arXiv:1606.01540* (2016).

[VGS16] Hado Van Hasselt, Arthur Guez und David Silver. „Deep reinforcement learning with double q-learning“. In: *Thirtieth AAAI conference on artificial intelligence*. 2016.

[Wan+16] Ziyu Wang u. a. „Dueling Network Architectures for Deep Reinforcement Learning“. In: *International Conference on Machine Learning*. 2016, S. 1995–2003.

[Dha+17] Prafulla Dhariwal u. a. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[Gri+18] Michael Griebel, Jochen Garcke, Bastian Bohn und Jannik Schürg. *Algorithms in Machine Learning and Their Application*. Institut für numerische Simulation, Universität Bonn. Vorlesungsskript. 2018.

[Hes+18] Todd Hester u. a. „Deep Q-learning From Demonstrations“. In: AAAI Conference on Artificial Intelligence. 2018.

[SB18] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. 2. Aufl. MIT press, 2018.

[Zha+19] Ruohan Zhang u. a. „Atari-head: Atari human eye-tracking and demonstration dataset“. In: *arXiv preprint arXiv:1903.06754* (2019).