Reinforcement Learning zur Energiemarktanalyse

Masterarbeit

vorgelegt von **Wadim Koslow**

am 20. Februar 2020

am Mathematischen Institut der
Universität zu Köln in Kooperation
mit dem Deutschen Zentrum für Luft- und Raumfahrt

Erstgutachter: Prof. Dr. Axel Klawonn

Zweitgutachter: Dr. Philipp Knechtges

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Die Nutzung von Energie, insbesondere in Form von Elektrizität ist essentiell um unsere Gesellschaft aufrecht zu erhalten und dessen Wohlstand zu gewährleisten. Aus einem Bericht der „World Bank Group“ [1] geht hervor, dass im Jahr 2017 86.8% der Weltbevölkerung an eine Stromversorgung angebunden waren. Dies macht den Energiemarkt zu einem der wichtigsten Wirtschaftszweige unserer Gesellschaft. Ein entscheidender Unterschied zu anderen Märkten und eine der größten Herausforderungen des Energiemarktes ist, dass es bisher noch keine zuverlässige Möglichkeit gibt Energie zu speichern. Das bedeutet, dass die produzierte Energie zu jedem Zeitpunkt der nachgefragten Energie entsprechen muss, um die Stromversorgungssicherheit zu gewährleisten. Aufgrund dieser und vieler weiteren technischen Bedingungen, die beim Handel von Elektrizität erfüllt sein müssen, ist der Energiemarkt auch einer der komplexesten Märkte. Bis zum Ende des letzten Jahrtausend lag die gesamte Verantwortung über den deutschen Energiemarkt noch in wenigen Händen. Im Jahr 1998 wurde jedoch die Liberalisierung des Energiemarktes [33] eingeleitet um die Monopolstellung aufzubrechen und die Konditionen für einen fairen Wettbewerb zu schaffen. Doch obwohl die Zahl der Stromanbieter unmittelbar nach der Öffnung des Marktes stark anstieg, wurden die meisten von ihnen durch große Firmen vom Markt verdrängt, was den ungleichen Verhältnissen beim Zugang zum Stromnetz geschuldet war. Aus diesem Grund wurde 2005 die Bundesnetzagentur [18] gegründet, die den Energiemarkt regulieren sollte um einen fairen Wettbewerb für alle Teilnehmer zu garantieren. Hinzu kommen die Herausforderungen des Klimawandels, da die Energiebedingten Emissionen für 84,5% aller deutschen Treibhausgas-Emissionen verantwortlich sind, wovon die Energiewirtschaft ca. 40% ausmacht [6]. Dies führte in den letzten Jahren

zu immer mehr politischen Regulierungen wie z.B. dem „Erneuerbare-Energien-Gesetz (EEG)“ [14], um die Forschung und Erzeugung von erneuerbaren Energien attraktiver zu machen. Doch bevor solche Regulierungen eingeführt werden, sollte zunächst untersucht werden, ob diese auch zum gewünschten Effekt führen. Dies geschieht, wie z.B. in [9], durch agentenbasierte Simulationen. Jedoch basieren die Entscheidungen der Agenten dieses Modells auf einfachen Regeln, die mithilfe von Expertenwissen modelliert wurden. Eine vielversprechende Ergänzung des Modells ist es, die agierenden Agenten durch Reinforcement Learning Algorithmen zu ersetzen; eine Richtung in die diese Arbeit einen ersten Schritt macht.

Im Rahmen dieser Masterarbeit wird zunächst die Theorie hinter Reinforcement Learning in Kapitel 1 vorgestellt. Insbesondere wird das Q-Learning bzw. Deep Q-Learning Verfahren vorgestellt und auf Konvergenz untersucht. Anschließend soll in Kapitel 2 die Anwendung dieser Verfahren zur Analyse des Energiemarktes untersucht werden. Dazu wird zunächst der Energiehandel des „Day-Ahead“ Marktes mithilfe des Q-Learning Verfahrens simuliert, das in der Dissertation von Dr. Weidlich [31] bereits behandelt worden ist. Des Weiteren wird ein Modell des Energiemarktes vorgestellt, an dem sich große Energiespeicher beteiligen können. Diese werden durch Reinforcement Learning Agenten modelliert, wozu es bisher noch keine vergleichbaren wissenschaftlichen Arbeiten gibt. Dabei soll insbesondere der Effekt von profitorientierten Energiespeicherbetreibern auf den Energiepreis untersucht werden.

1 Reinforcement Learning

Reinforcement Learning ist ein Teilgebiet des Bereichs Machine Learning, dessen Ziel es ist Verfahren zu entwickeln, die in der Lage sind, eine Folge von guten Entscheidungen zu lernen. Die grundlegende Idee hinter den meisten Verfahren ist, dass ein Agent durch unterschiedliche Handlungen mit seiner Umgebung interagieren kann und dafür eine Belohnung erhält, dessen Größe davon abhängt, wie „gut“ die getätigte Handlung war. Das Ziel dieses Agenten ist es seine Belohnung über einen bestimmten Zeitraum zu maximieren. Im Gegensatz zum überwachten (engl.: supervised) Machine Learning Bereich, erfordert das Training eines Reinforcement Learning Agenten keine reellen Daten, sondern wird durch dessen eigene Erfahrung überwacht. Aus diesem Grund gehört es weder zu Supervised noch zu Unsupervised Learning und hat sich somit als drittes Forschungsgebiet des Bereichs Machine Learning etabliert. In diesem Kapitel wird die grundlegende Theorie von Reinforcement Learning erarbeitet, die hauptsächlich auf dem Buch „Reinforcement Learning: An Introduction“ [29] und der Vorlesung der Stanford Universität [5] basiert.

1.1 Markov Entscheidungsprozess

Ein Markov Entscheidungsprozess (Engl.: Markov Decision Process (MDP)) ist eine mathematisch idealisierte Form eines Reinforcement Learning Problems, für die es möglich ist, theoretische Aussagen zu formulieren. Es beschreibt den Prozess eines Agenten, der lernt die Belohnung, die er durch eine Folge von Entscheidungen von seiner Umwelt

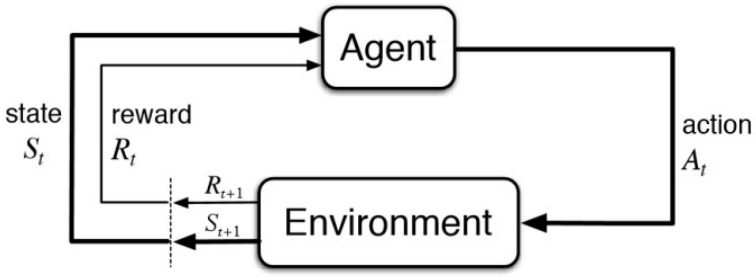


Abbildung 1.1: Interaktion zwischen Agent und Umwelt [29, Fig. 3.1]

erhält, zu maximieren. Formal definieren wir

Definition 1.1.1.

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

- $\mathcal{S}$ *eine Menge von Zuständen* s *(Engl.: States) in denen sich der Agent befinden kann,*
- $\mathcal{A}$ *eine Menge von Aktionen* a *(Engl.: Actions) die der Agent ausführen kann,*
- $p(s', r|s, a) \coloneqq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$ *die wohldefinierte Wahrscheinlichkeitsverteilung, mit der man von Zustand* s *durch Aktion* a*, in Zustand* s' *gelangt und dafür die Belohnung (Engl.: Rewards)* $r \in \mathbb{R}$ *erhält. Die Wahrscheinlichkeitsverteilung* p *wird auch als Dynamik des Modells bezeichnet.*

In Abbildung 1.1 ist die Interaktion zwischen dem Agenten und seiner Umwelt in einem MDP dargestellt. In diesem Kapitel werden nur endliche MDPs betrachtet, d.h. dass $|\mathcal{S}| = N_S < \infty$ und $|\mathcal{A}| = N_A < \infty$ gilt. Weiterhin handelt es sich hier um stationäre MDPs, d.h. dass die Dynamik des Modells sich über die Zeit nicht verändert. Außerdem impliziert die Wohldefiniertheit der Wahrscheinlichkeitsverteilung p, dass ein MDP die sogenannte Markov Eigenschaft erfüllen muss. Dies bedeutet, dass der zukünftige Zustand s_{t+1} nur von dem aktuellen Zustand s_t abhängt und nicht von den vergangenen Zuständen $h_t \coloneqq s_0, s_1, ..., s_t$:

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = p(s_{t+1}, r_{t+1}|h_t, a_t). \tag{1.1}$$

Insbesondere impliziert die Markov Eigenschaft, dass der nächste Zustand und die erhaltene Belohnung nur von dem aktuellen Zustand und ausgeführten Aktion abhängt und nicht von dem Zeitpunkt t, an dem dieser besucht wird, da für zwei Zustände $s_{t_1}, s_{t_2} \in \mathcal{S}$ mit $s_{t_1} = s_{t_2}$ und Aktionen $a_{t_1}, a_{t_2} \in \mathcal{A}$ mit $a_{t_1} = a_{t_2}$

$$\begin{aligned} p(s_{t_1+1}, r_{t_1+1}|h_{t_1}, a_{t_1}) &= p(s_{t_1+1}, r_{t_1+1}|s_{t_1}, a_{t_1}) \\ &= p(s_{t_2+1}, r_{t_2+1}|s_{t_2}, a_{t_2}) \\ &= p(s_{t_2+1}, r_{t_2+1}|h_{t_2}, a_{t_2}) \end{aligned} \tag{1.2}$$

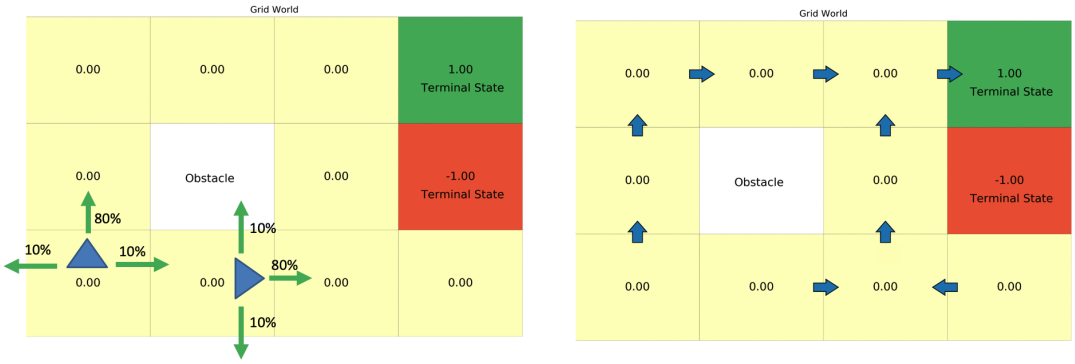
gilt. Dies ist allerdings eine Annahme, die in der Realität nicht immer erfüllt ist. Man stelle sich dazu beispielsweise einen Agenten vor, dessen Zustand der aktuelle Blutdruck eines Patienten ist und dieser entscheiden soll, ob dem Patienten Medikation verabreicht werden soll, um den Blutdruck zu regulieren. Dieses Szenario würde die Markov Eigenschaft nicht erfüllen, da eine richtige Entscheidung davon abhängt, ob der Patient z.B. vorher Sport betrieben hat, was der Agent allein durch den aktuellen Blutdruck nicht wissen kann. Der Agent kann also im gleichen Zustand und mit der gleichen Handlung, mit unterschiedlich verteilten zukünftigen Zuständen und Belohnungen konfrontiert sein. Es ist jedoch möglich ein MDP, das diese Bedingung nicht erfüllt, „markovsch“ zu machen, indem die Menge der Zustände $\mathcal{S}$ erweitert wird, zum Beispiel durch weitere medizinische Prädiktoren, sodass die Zustände eindeutiger werden.

Nun bleibt noch die Frage, wie ein Agent seine Entscheidungen trifft. Dafür benötigen wir folgende

Definition 1.1.2.

Eine Strategie (Engl.: policy) ist eine Funktion $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, die jeder möglichen Handlung $a \in \mathcal{A}$ zu einem gegeben Zustand $s \in \mathcal{S}$ eine Wahrscheinlichkeit $\pi(a|s)$ zuteilt.

Ein Agent der einer Strategie π folgt, führt im Zustand s die Aktion a mit einer Wahrscheinlichkeit von $\pi(a|s)$ aus. Es ist jedoch auch möglich eine deterministische Strategie zu definieren, die jedem Zustand genau eine Aktion zuteilt. Tatsächlich sind viele der Verfahren zum Lösen von Reinforcement Learning Problemen auf deterministische Strategien



(a) Markov Entscheidungsprozess (b) Strategie

Abbildung 1.2: Darstellung eines Markov Entscheidungsprozesses eines „Gitterwelt“ Problems. Jedes Feld ist ein Zustand in dem sich der Agent befinden kann, wobei das grüne bzw. rote Feld jeweils Endzustände sind. Die Zahlen in den Feldern beschreiben zunächst die Belohnungen die der Agent für das Erreichen dieser Felder erhält. Durch die Wahrscheinlichkeiten ist links die Dynamik des Modells dargestellt, die beschreibt wie der Agent von einem Zustand in den nächsten Zustand gelangt. Rechts ist hingegen eine Strategie beispielhaft dargestellt, die jedem Zustand eine Richtung zuweist, in die sich der Agent bewegen soll.

beschränkt. Allerdings gibt es auch einige Probleme, für die eine deterministische Strategie suboptimal ist. Wenn ein Agent z.B. bei einer Partie Schere, Stein, Papier immer einer deterministischen Strategie folgen würde, so könnte man diese leicht ausnutzen. Wenn man sich jedoch auf stationäre MDP's, die die Markov Eigenschaft erfüllen beschränkt, so ist die optimale Strategie immer deterministisch [26]. Deshalb beschränken wir uns in dieser Arbeit auf deterministische Strategien.

Beispiel 1.1.3. Ein klassisches Reinforcement Learning Problem ist die sogenannte Gitterwelt, die in Abbildung 1.2(a) dargestellt ist. Ein Agent, z.B. ein Roboter kann sich innerhalb dieser Welt von Feld zu Feld bewegen. Sein Ziel ist es von einem beliebigen Feld einen Weg zur Ladestation zu finden, die durch das grüne Feld dargestellt ist. Das rote

Feld sollte er dabei versuchen zu vermeiden, da dort eine Schrottpresse auf ihn wartet. Die Menge der Zustände $\mathcal{S}$ ist durch die elf Felder dargestellt, in denen sich der Agent befinden kann. Die Menge der Aktionen, die der Agent ausführen kann sind durch die Richtungen definiert, in die sich der Agent bewegen kann, d.h. $\mathcal{A} = \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$. Für jede Aktion die der Agent ausführt, erhält der Agent eine skalare Belohnung von 0, außer wenn er in einen der Endzustände gelangt, für die er eine Belohnung von $+1$ bzw. -1 erhält. Um das Problem nun interessanter zu machen, sind die Zustandsübergänge nicht deterministisch. Das bedeutet, wenn der Agent versucht sich ein Feld nach oben zu bewegen, kommt er nur mit einer 80% Wahrscheinlichkeit tatsächlich nach oben, und rutscht sonst mit einer Wahrscheinlichkeit von jeweils 10% nach links oder nach rechts von der gewählten Aktion aus. In Abbildung 1.2(b) ist eine Strategie dargestellt die der Agent verfolgen kann. Hierbei ist zu beachten, dass die dargestellte Strategie deterministisch ist, d.h. dass der Agent im Feld oben links immer versuchen wird nach rechts zu gehen. Ob ihm das tatsächlich gelingt, wird hingegen von der Umgebung bestimmt, also dem Zustandsübergangsmodell $p(s', r|s, a)$, das durch die Wahrscheinlichkeiten in Abbildung 1.2(a) dargestellt ist.

1.2 Strategie Evaluation

In diesem Abschnitt wollen wir untersuchen, wie man die Güte einer Strategie bewerten kann. Da die Umgebung eines Agenten in der Regel nicht deterministisch ist und die Belohnungen, die der Agent für seine ausgeführten Aktionen erhält, sich jedes Mal unterscheiden können, muss eine Strategie mehrmals simuliert werden, um eine konsistente Schätzung des Erfolgs einer Strategie zu erhalten. Sei dazu $\pi : \mathcal{S} \rightarrow \mathcal{A}$ eine deterministische Strategie, die jedem Zustand $s \in \mathcal{S}$ eine Aktion $a \in \mathcal{A}$ zuweist. Weitehin sei G_t die diskontierte zukünftige Belohnung zum Zeitpunkt t die durch

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1} \tag{1.3}$$

definiert ist, wobei R_i die Zufallsvariable der zum Zeitpunkt i erhaltenen Belohnung bezeichnet. Dabei ist T der Zeitpunkt an dem ein Endzustand erreicht wird, z.B. das Ende eines Spiels. Dieser muss nicht bei jeder Simulation des Problems zum gleichen Zeitpunkt

erreicht werden, da z.B. ein Schachspiel 10 oder auch 200 Züge dauern kann. Wenn es keinen Endzustand gibt, so wird dies als kontinuierliches Problem bezeichnet und es gilt $T = \infty$, während $T < \infty$ ein episodisches Problem darstellt. Der Diskontierungsfaktor $\gamma \in [0, 1]$ beschreibt dabei, wie wichtig die zukünftige Belohnung im Vergleich zur sofortigen Belohnung ist. Für $\gamma = 1$ ist die zukünftige Belohnung genauso wichtig wie die sofortige, was eine angebrachte Wahl für episodische Probleme ist. Für kontinuierliche Probleme sollte hingegen $\gamma < 1$ gewählt werden, da dies garantiert, dass selbst für $T = \infty$, $G_t < \infty$ gilt, solange die Folge der Belohnungen R_k beschränkt ist [29, Abschnitt 3.3]. Damit können wir nun die sogenannte Zustandswert-Funktion $V^\pi : \mathcal{S} \to \mathbb{R}$ definieren

$$V^\pi(s_t) \coloneqq \mathbb{E}_\pi[G_t | S_t = s_t], \tag{1.4}$$

wobei $\mathbb{E}_\pi$ den Erwartungswert bezüglich der Strategie π bezeichnet. Diese Funktion beschreibt, wie gut es ist in einem Zustand $s \in \mathcal{S}$ zu sein, wenn man dabei der Strategie π folgt. Eine Möglichkeit die Zustandswerte der Strategie π zu schätzen, ist es Monte-Carlo-Simulationen [29, Kapitel 5] des MDPs für jeden Zustand $s \in \mathcal{S}$ durchzuführen. Dabei wird das MDP durch die Entscheidungen der Strategie π simuliert, bis ein Endzustand T erreicht wird. Dieser Prozess wird mehrmals wiederholt, wodurch man eine Schätzung für $V^\pi(s_t)$ durch $\frac{1}{m} \sum_{i=1}^{m} G_t^{(i)}$ erhält, wobei m die Anzahl der durchgeführten Simulationen ist. Allerdings führt diese Methode insbesondere für große T zu einer hohen Varianz der Schätzung für $V^\pi(s)$ und kann auch nur auf Probleme angewandt werden, für die ein Endzustand auch tatsächlich existiert und erreicht wird. Eine weitere Möglichkeit eine Schätzung für $V^\pi(s)$ zu berechnen, ist die Temporal Difference Methode (TD) [29, Kapitel 6]. Um diese anwenden zu können, müssen wir zunächst die Definition aus (1.4) in die sogenannte Bellman Gleichung [29, Abschnitt 3.5] umformen:

$$\begin{aligned}
V^\pi(s_t) &= \mathbb{E}_\pi[G_t | S_t = s_t] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s_t] \\
&= \mathbb{E}_\pi[R_{t+1} | S_t = s_t] + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s_t] \\
&\overset{*}{=} \mathbb{E}_\pi[R_{t+1} | S_t = s_t] + \gamma \mathbb{E}_\pi[\mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] | S_t = s_t] \\
&= \mathbb{E}_\pi[R_{t+1} | S_t = s_t] + \gamma \mathbb{E}_\pi[V^\pi(s_{t+1}) | S_t = s_t] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(s_{t+1}) | S_t = s_t],
\end{aligned} \tag{1.5}$$

wobei $(*)$ mithilfe des „Law of Iterated Expectations“ gilt. Diese Beziehung zwischen dem Zustandswert $V^\pi(s_t)$ im Zustand s und $V^\pi(s_{t+1})$ im folgenden Zustand s_{t+1} ermöglicht es uns die Zustandswert-Funktion mit einem iterativen Verfahren zu approximieren. Sei dazu $T : \mathcal{D} \to \mathbb{R}$ ein Operator, wobei $\mathcal{D}$ die Menge der möglichen Funktionen $V : \mathcal{S} \to \mathbb{R}$ ist und

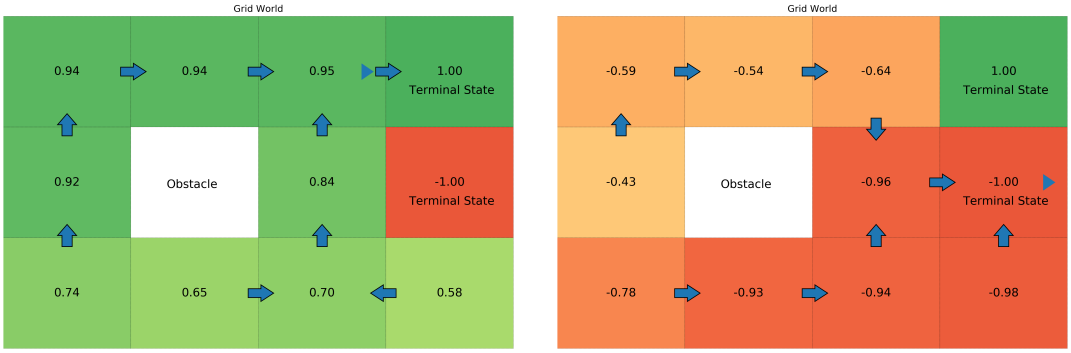
$$T(V(s_t)) = \mathbb{E}_\pi[R_{t+1} + \gamma V(s_{t+1})|S_t = s_t]. \tag{1.6}$$

Dann folgt aus der Bellman Gleichung (1.5), dass V^π eine Fixpunktlösung des Operators T sein muss. Der Fixpunkt V^π kann nun mit der TD Methode berechnet werden, indem das MDP simuliert wird und dessen Zustandswerte nach jeder ausgeführten Aktion um die erhaltene Belohnung und den Zustandswert des nächsten Zustands korrigiert werden

$$V^{\pi,\text{new}}(s_t) \coloneqq (1-\alpha)V^\pi(s_t) + \alpha(r_{t+1} + \gamma V^\pi(s_{t+1})), \tag{1.7}$$

wobei α die Lernrate ist und r_{t+1} die Belohnung, die durch das Erreichen des Zustands s_{t+1} erhalten wird. Der Vorteil dieses Verfahrens gegenüber der Monte-Carlo-Methode ist, dass die Zustandswerte nach jeder ausgeführten Aktion aktualisiert werden können und nicht bis zum Ende einer Episode gewartet werden muss, sodass man diese auch für kontinuierliche Probleme nutzen kann. Außerdem muss sowohl bei der Monte-Carlo als auch der TD-Methode die Dynamik des Modells $p(s', r|s, a)$ nicht bekannt sein um V^π zu approximieren, weswegen diese Methoden auch als „Modell-Frei“ bezeichnet werden.

Beispiel 1.2.1. In Abbildung 1.3(a) sind die Zustandswerte der Gitterwelt für die Strategie aus Beispiel 1.1.3 dargestellt, die mithilfe der TD Methode mit Lernrate $\alpha = 0.1$ und Diskontierungsfaktor $\gamma = 0.99$ berechnet wurden. Die Strategie wurde dabei über 1000 Episoden simuliert, die bis zum Erreichen einer der beiden Endzustände andauern. Der Agent startet dabei nach jeder Episode in einem zufälligen Zustand, der kein Endzustand ist. Wie man sieht, haben die Zustände auf dem unteren Weg niedrigere Zustandswerte als auf dem Oberen. Dies liegt daran, dass die Wahrscheinlichkeit in die Schrottpresse auszurutschen auf dem unteren Weg höher ist als auf dem oberen. Die berechneten Werte beziehen sich allerdings nur auf die dargestellte Strategie. Wenn man stattdessen eine



(a) Gute Strategie

(b) Schlechte Strategie

Abbildung 1.3: Berechnete Zustandswerte der Gitterwelt mithilfe der TD-Methode (Gleichung (1.7)) für unterschiedliche Strategien

andere, schlechtere Strategie auswerten würde, erhält man andere Zustandswerte, wie man in Abbildung 1.3(a) sieht.

1.3 Strategie Kontrolle

Jetzt wissen wir, wie man eine Strategie evaluiert, doch wie findet man eine gute Strategie? Eine Strategie π ist besser als eine Strategie π', wenn $V^{\pi}(s) \geq V^{\pi'}(s)\ \forall s \in \mathcal{S}$ gilt [5, VL2], wobei dies nur eine partielle Ordnung ist. Somit ist eine optimale Strategie π^* definiert durch

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \tag{1.8}$$

und der zugehörigen optimalen Zustandswertefunktion

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad \forall s \in \mathcal{S} \tag{1.9}$$

Eine naive Möglichkeit die optimale Strategie zu finden, ist es alle möglichen Strategien zu durchsuchen. Da es aber $|\mathcal{A}|^{|\mathcal{S}|}$ deterministische Strategien gibt, ist dies selbst für endliche MDPs zu teuer. Doch um eine Strategie berechnen zu können müssen wir nicht

nur Zustände bewerten sondern auch Aktionen. Um dies zu erreichen, können wir die Aktionswerte-Funktion $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ wie folgt definieren:

$$Q^\pi(s_t, a_t) \coloneqq \mathbb{E}[R_{t+1} + \gamma V^\pi(s_{t+1}) | S_t = s_t, A_t = a_t] \tag{1.10}$$

Hierdurch wird gemessen wie gut es ist zunächst Aktion $a \in \mathcal{A}$ auszuführen und anschließend der Strategie π zu folgen. Nun kann man zeigen, dass mit

$$\pi_{i+1} = \arg\max_a Q^{\pi_i}(s, a) \quad \forall s \in \mathcal{S}, \tag{1.11}$$

die Strategie π_{i+1} besser ist als π_i, also dass $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$ gilt:

$$\begin{aligned}
V^{\pi_i}(s_t) &\leq \max_a Q^{\pi_i}(s_t, a) \\
&\overset{(1.10)}{=} \max_a \mathbb{E}[R_{t+1} + \gamma V^{\pi_i}(s_{t+1}) | S_t = s_t, A_t = a_t] \\
&\overset{(1.11)}{=} \mathbb{E}[R_{t+1} + \gamma V^{\pi_i}(s_{t+1}) | S_t = s_t, A_t = \pi_{i+1}(s_t)] \\
&\leq \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^{\pi_i}(s_{t+1}, a') | S_t = s_t, A_t = \pi_{i+1}(s_t)] \\
&= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma V^{\pi_i}(s_{t+2}) | s_{t+1}, \pi_{i+1}(s_{t+1})] | s_t, \pi_{i+1}(s_t)] \\
&\vdots \\
&\leq \mathbb{E}[R_{t+1} + \ldots + \gamma \mathbb{E}[R_{T-1} + \gamma V^{\pi_i}(s_{T-1}) | s_{T-2}, \pi_{i+1}(s_{T-2})] \ldots | s_t, \pi_{i+1}(s_t)] \\
&= \mathbb{E}[R_{t+1} + \ldots + \gamma \mathbb{E}[R_{T-1} + \gamma \mathbb{E}[R_T | s_{T-1}] | s_{T-2}, \pi_{i+1}(s_{T-2})] \ldots | s_t, \pi_{i+1}(s_t)] \\
&\overset{\text{LIE}}{=} \mathbb{E}[\sum_{k=t}^{T} \gamma^{k-t} R_{k+1} | S_t = s_t, A_t = \pi_{i+1}(s_t)] \\
&= V^{\pi_{i+1}}(s_t).
\end{aligned} \tag{1.12}$$

Somit gilt $\pi_i \leq \pi_{i+1}$. Die Aktionswertefunktion der optimalen Strategie sind analog zur Zustandswertefunktion durch

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \tag{1.13}$$

definiert. Da π^* bereits die optimale Strategie ist, muss für die erste Ungleichung aus Gleichung (1.12) sogar

$$V^*(s) = \max_a Q^*(s, a) \qquad \forall s \in \mathcal{S} \tag{1.14}$$

gelten. Insbesondere gilt dadurch

$$\begin{aligned} Q^*(s_t, a_t) &= \mathbb{E}[R_{t+1} + \gamma V^*(s_{t+1}) | S_t = s_t, A_t = a_t] \\ &= \mathbb{E}[R_{t+1} + \gamma \max_a Q^*(s_{t+1}, a) | S_t = s_t, A_t = a_t] \end{aligned} \tag{1.15}$$

Analog zu Gleichung (1.6) können wir einen Operator $U : \mathcal{D} \to \mathbb{R}$ definieren, wobei $\mathcal{D}$ die Menge der möglichen Funktionen $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ ist und

$$U(Q(s_t, a_t)) = \mathbb{E}[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) | S_t = s_t, A_t = a_t]. \tag{1.16}$$

Nach Gleichung (1.15) ist der Fixpunkt des Operators U die optimale Aktionswertefunktion Q^*. Damit diese eindeutig exisitiert reicht es zu zeigen, dass U ein Kontraktionsoperator bezüglich der L_∞ Norm

$$||Q||_{L_\infty} \coloneqq \max_{s,a} |Q(s, a)| \tag{1.17}$$

ist. Sei dazu $Q_i, Q_j \in \mathcal{D}$. Dann gilt

$$\begin{aligned} ||UQ_i - UQ_j||_{L_\infty} &= \max_{s_t, a_t} |\mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_i(s_{t+1}, a') | s_t, a_t] \\ &\qquad - \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_j(s_{t+1}, a') | s_t, a_t]| \\ &= \max_{s_t, a_t} |\mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_i(s_{t+1}, a') - R_{t+1} - \gamma \max_{a'} Q_j(s_{t+1}, a') | s_t, a_t]| \\ &= \max_{s_t, a_t} |\mathbb{E}[\gamma \max_{a'} Q_i(s_{t+1}, a') - \gamma \max_{a'} Q_j(s_{t+1}, a') | s_t, a_t]| \\ &\leq \gamma \max_{s_t, a_t} |\mathbb{E}[\max_{a'} |Q_i(s_{t+1}, a') - Q_j(s_{t+1}, a')| | s_t, a_t]| \\ &\leq \gamma \max_{s_t, a_t} |\mathbb{E}[\max_{s,' a'} |Q_i(s', a') - Q_j(s', a')| | s_t, a_t]| \\ &= \gamma \max_{s_t, a_t} |\mathbb{E}[||Q_i(s', a') - Q_j(s', a')||_{L_\infty} | s_t, a_t]| \\ &= \gamma ||Q_i(s', a') - Q_j(s', a')||_{L_\infty} \max_{s_t, a_t} |\mathbb{E}[1 | s_t, a_t]| \\ &= \gamma ||Q_i(s', a') - Q_j(s', a')||_{L_\infty}. \end{aligned} \tag{1.18}$$

Somit ist U für $\gamma < 1$ ein Kontraktionsoperator und nach dem Banachschen Fixpunktsatz existiert die eindeutige Lösung Q^*.

Ein Algorithmus, mit dem man eine optimale Strategie π^* berechnen kann, ist der sogenannte Q-Learning Algorithmus [29, Abschnitt 6.5], der auf dem Prinzip der TD Methode aus dem letzten Kapitel basiert. Dabei wird $Q(s,a)$ approximiert, indem das MDP simuliert wird und nach jeder Handlung des Agenten die Werte für Q wie folgt aktualisiert werden:

$$Q^{\pi}_{k+1}(s_t,a_t) = Q^{\pi}_k(s_t,a_t) + \alpha_k(r_{t+1} + \gamma \max_{a'} Q^{\pi}_k(s_{t+1},a') - Q_k(s_t,a_t)) \tag{1.19}$$

Die Konvergenz des Q-Learning Verfahrens ist durch den folgenden Satz gewährleistet:

Satz 1.3.1. *Sei $(\mathcal{S},\mathcal{A},\mathcal{R},r)$ ein endliches MDP und π eine beliebige Strategie, mit der das MDP simuliert wird. Dann konvergiert der Q-Learning Algorithmus, der durch* (1.19) *gegeben ist, für $k \to \infty$ und eine beliebige Initialisierung $Q_0(s,a)$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$ mit Wahrscheinlichkeit 1 gegen die optimale Aktionswerte-Funktion Q^*, solange*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \textit{und} \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{1.20}$$

und

$$\mathbb{P}_{\pi}[A_t = a | S_t = s] > 0 \qquad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{1.21}$$

gilt.

Beweis. Siehe [23, Theorem 1]. □

Eine mögliche Wahl für die Lernrate damit die Bedingungen für die Konvergenz erfüllt sind, ist $\alpha_k = \frac{1}{k}$. Allerdings führt dies zu einem sehr langsamen Lernprozess, weswegen man in der Praxis oft eine konstante Lernrate wählt. Dadurch wird die Q-Funktion durch die zuletzt erhaltenen Belohnungen verzerrt, aber es führt erfahrungsgemäß dennoch zu guten und schnelleren Ergebnissen [29, Abschnitt 2.5]. In nicht stationären MDPs, ist dieses Verhalten sogar wünschenswert. Eine intuitive Wahl für die Strategie π mit der das MDP während des Q-learning Algorithmus simuliert werden kann, besteht darin den maximalen Werten der approximierten Q-Funktion zu folgen:

$$\pi(s_t) = \arg\max_a Q(s_t,a) \tag{1.22}$$

Allerdings impliziert Satz 1.3.1, dass alle Zustandsaktionspaare $(s, a) \in \mathcal{S} \times \mathcal{A}$ unendlich oft besucht werden müssen. Wenn der Agent jedoch immer nur den höchsten Q-Werten folgen würde, so kann es passieren, dass eine Aktion, die eigentlich sehr gut wäre nie ausgeführt wird, da diese mit einem niedrigen Q-Wert initialisiert wurde. Dies führt zu dem Dilemma zwischen Erkundung und Ausbeutung (Engl.: Exploration and Exploitation) [29, Abschnitt 1.1]. Der Agent sollte in der Lage sein, vor allem am Anfang, wenn die Q-Werte noch von einer willkürlich gewählten Initialisierung abhängen, verschiedene Zustände und Aktionen zu erkunden. Auf der anderen Seite sollte er auch seine Erfahrung ausnutzen, um seine Belohnung zu maximieren. Um dieses Problem zu lösen, gibt es verschiedene Möglichkeiten:

Eine optimistische Initialisierung der Aktionswerte, mit Werten die deutlich höher sind als die wahren optimalen Q-Werte, zwingt den Agenten alle Zustände und alle Aktionen zu besuchen, da diese anschließend durch die Aktualisierung (1.19) nach unten korrigiert werden. Bei einer zu optimistischen Initialisierung, kann es jedoch den Lernprozess verlangsamen und es ist auch nicht immer klar, was eine optimistische Initialisierung ist.

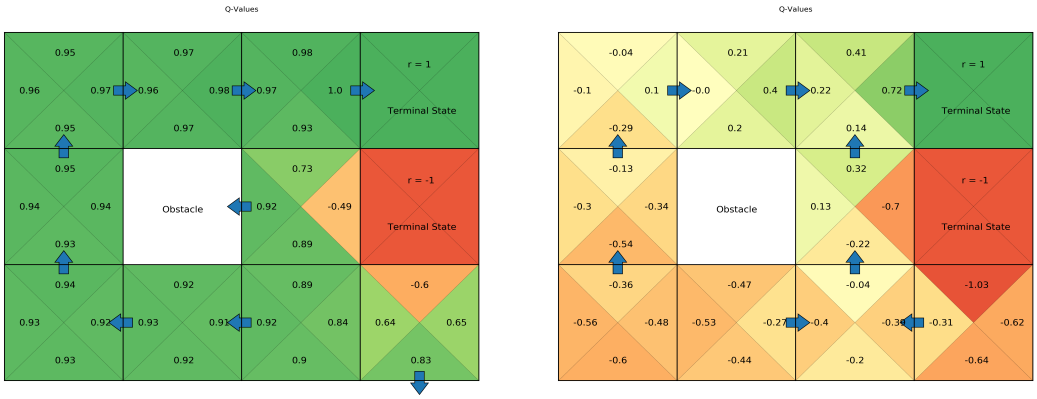
Eine weitere Möglichkeit ist es die „ε-greedy“ Strategie [29, Abschnitt 2.2] zu verfolgen, die mit einer Wahrscheinlichkeit von $\varepsilon \in [0, 1]$ eine zufällige Aktion durchführt und sonst (1.22) verfolgt. Der Parameter ε sollte dabei im Laufe der Simulation schrumpfen, da der Agent immer erfahrener wird. Dies erreicht man, indem man nach jeder Aktion oder Episode ε mit einem Zerfallsparameter $\delta_\varepsilon \in [0, 1]$ multipliziert:

$$\varepsilon_{t+1} = \varepsilon_t \cdot \delta_\varepsilon \tag{1.23}$$

Des weiteren ist es möglich die „Softmax Aktionswahl“ (Engl.: Softmax Actionselection) durchzuführen:

$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_a e^{Q(s,a)/\tau}} \tag{1.24}$$

Im Gegensatz zur ε-greedy Strategie, in der keine Priorisierung der Aktionen stattfindet, hängt hier die Wahrscheinlichkeit eine Aktion auszuwählen von den entsprechenden Q-Werten ab. Dabei ist τ der Temperaturparameter, der bestimmt wie groß der Unterschied



(a) $r = 0$ (b) $r = -0.2$

Abbildung 1.4: Optimale Strategien und Aktionswerte des Q-Learning Verfahrens. Links erhält der Agent für jede ausgeführte Aktion eine Belohnung von 0, während rechts jede Aktion mit einer negativen Belohnung bestraft wird.

in den Wahrscheinlichkeiten ist, bezogen auf die Differenz der Q-Werte. Ein hoher τ Parameter führt zu einer uniformen Wahrscheinlichkeitsverteilung.
Es ist zu beachten, dass der Q-Learning Algorithmus durch $\max_a Q(s_{t+1}, a)$ aus Gleichung (1.19) davon ausgeht, dass der Agent im Zustand s_{t+1} die Aktion

$$a = \arg\max_a Q(s_{t+1}, a) \tag{1.25}$$

durchführt. Durch die Einführung einer Erkundungsstrategie (z.B. ε-greedy) ist dies jedoch nicht mehr notwendigerweise erfüllt. Aus diesem Grund wird das Q-Learning Verfahren als „off-policy“ Verfahren bezeichnet. Dies hat allerdings noch keine Auswirkung auf die Konvergenz des hier vorgestellten Verfahrens, wird aber wie später in Beispiel 1.4.1 zu sehen zu Problemen führen.

Beispiel 1.3.2. In Abbildung 1.4(a) ist die optimale Strategie für das MDP aus Beispiel 1.2.1 dargestellt, die mithilfe des Q-Learning Verfahrens und ε-greedy Strategie, berechnet wurde. Dies erscheint auf den ersten Blick nicht besonders gut, da der Roboter in zwei Feldern versucht gegen eine Wand zu laufen. Dies liegt aber daran, dass er in

diesen Feldern eine 10% Wahrscheinlichkeit hat, den Endzustand mit negativer Belohnung zu erreichen, wenn dabei die Strategie aus Abbildung 1.2(b) verfolgt werden würde. Da der Roboter keinen Zeitdruck hat, läuft er lieber gegen eine Wand als zu riskieren eine negative Belohnung zu erhalten. Wir können den Roboter allerdings zwingen den riskanteren Weg zu wählen, indem wir ihm eine Belohnung $r = -0.2$ für jede Aktion zuweisen, die nicht in einem Endzustand endet. Das Ergebnis ist in Abbildung 1.4(b) zu sehen. Die optimale Strategie hängt also sehr stark von den modellierten Belohnungen des Problems ab. Aufgrund der negativen Belohnungen, die der Agent für jede ausgeführte Aktion erhält, sinken dementsprechend auch die Q-Werte des MDPs.

1.4 Funktionsapproximation

Obwohl die im letzten Kapitel vorgestellten Methoden gute Konvergenz-Eigenschaften besitzen, ist die Anwendbarkeit dieser durch die Voraussetzung eines endlichen MDPs beschränkt. Wenn man z.B. ein selbstfahrendes Auto mithilfe von Bilddaten trainieren möchte, würde der Zustandsraum $\mathcal{S}$ aus allen Kombinationen der verschiedenen Pixelwerte der Bilder bestehen und somit wäre die Kardinalität des Zustandsraumes viel zu groß. Eine der Voraussetzungen für die Konvergenz des Q-Learning Verfahrens ist, dass alle Zustandsaktionspaare unendlich oft besucht werden müssen, sodass es auf Problemen mit großen oder gar unendlichen Zustandsräumen nicht anwendbar ist. Für solche Probleme wird ein einzigartiger Zustand nur selten während des Trainings besucht, sodass der Agent die Fähigkeit braucht, aus diesem Zustand eine Strategie für alle Zustände zu generalisieren. Dies können wir erreichen, indem wir die wahren Aktionswerte mithilfe einer Funktion $\hat{Q}^*$ approximieren

$$Q^*(s_t, a_t) = \hat{Q}^*(s_t, a_t; \theta). \tag{1.26}$$

Hierbei sind θ die Parameter der approximierten Funktion $\hat{Q}\pi : \mathcal{S} \to \mathbb{R}$. Dies können z.B. die Parameter einer linearen Funktion sein, oder auch die Gewichte eines neuronalen Netzwerks darstellen. Wir können diese Funktion approximieren, indem wir die Verlustfunktion

$$J(\theta) \coloneqq \frac{1}{2}\mathbb{E}[(Q^*(S,A) - \hat{Q}^*_\theta(S,A))^2] \tag{1.27}$$

bezüglich θ minimieren. Dies kann auch als Supervised Machine Learning Problem betrachtet werden. Wir können nun das Gradientenverfahren anwenden, das darin besteht den Gradienten der Verlustfunktion (1.27) bezüglich θ zu berechnen und anschließend die Parameter in die Richtung des steilsten Abstiegs zu korrigieren:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta J(\theta_k) \tag{1.28}$$

wobei α die Lernrate ist und

$$\nabla J(\theta) = -\mathbb{E}[(Q^*(S,A) - \hat{Q}^*_\theta(S,A)) \cdot \nabla \hat{Q}^*_\theta(S,A)] \tag{1.29}$$

gilt. Nun haben wir jedoch das Problem, dass wir noch nichts über die wahre Funktion Q wissen und somit den Gradienten aus Gleichung (1.29) nicht berechnen können. Um dieses Problem zu lösen, können wir die wahre Funktion $Q^*(s,a)$ mit dem Temporal Difference Ansatz aus Kapitel 1.3 approximieren:

$$Q^*(s_t, a_t) \approx R_{t+1} + \max_a \hat{Q}^*_\theta(s_{t+1}, a) \tag{1.30}$$

Der Agent generiert somit approximierte Beobachtungen der wahren Funktion Q, indem er die wahre Belohnung die er für Aktion a_t erhalten hat mit einer Vorhersage der Funktion Q des nächsten Zustands s_{t+1} verbindet.

Doch das Approximieren einer Funktion mit approximierten Daten hat seinen Preis, denn wir müssen dadurch die Garantie für die Konvergenz des Verfahrens aufgeben. Dies soll durch das folgende Beispiel illustriert werden:

Beispiel 1.4.1. Baird's Gegenbeispiel

Um die Probleme der Temporal Difference Methode mit Funktionsapproximation darzustellen, hat Baird in seiner Arbeit „Residual Algorithms: Reinforcement Learning with Function Approximation“ [3] das folgende Beispiel konstruiert. Sei $\mathcal{M}$ das MDP, das in

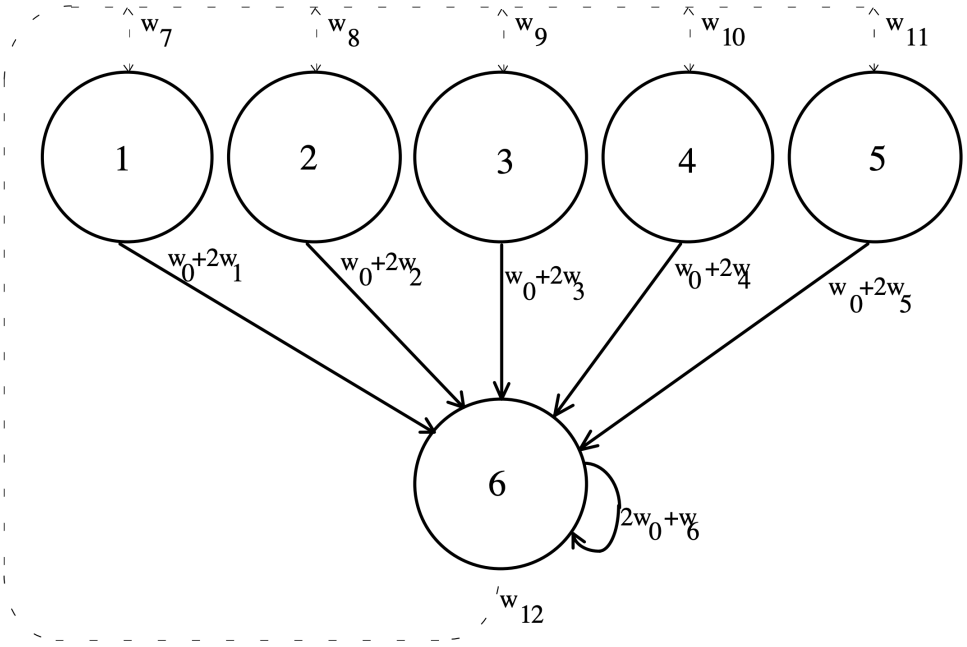


Abbildung 1.5: Darstellung des MDP's des Gegenbeispiels von Baird[3, Errata]

Abbildung 1.5 dargestellt ist. Das MDP besteht aus 6 Zuständen, die durch die Kreise dargestellt sind. Der Agent kann sich in jedem Zustand entweder für die durchgezogenen oder die gestrichelten Linien entscheiden. Die durchgezogene Linie bringt den Agenten immer in den unteren Zustand, während die gestrichelte Linie mit gleichmäßiger Wahrscheinlichkeit in einen der oberen Zustände führt. Um das Gradientenverfahren auf dieses Beispiel anzuwenden parametrisieren wir die Q-Werte mit einer linearen Funktion:

$$Q(s_t, a_t) = \phi(s_t, a_t)^T \cdot w \tag{1.31}$$

Dabei sind $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{13}$ die Merkmale der Zustände und Aktionen, z.B. $\phi(1, \longrightarrow) = (1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. $w \in \mathbb{R}^{13}$ sind die Parameter der linearen Funktion, die wir approximieren wollen. Da das MDP nur 6 Zustände und 2 Aktionen besitzt, können die Aktionswerte nun wie in Tabelle 1.1 dargestellt werden.

Die Belohnung, die der Agent in jedem Zustand für jede Aktion erhält, ist immer Null und dementsprechend sind auch die wahren Q-Werte alle gleich Null. Somit sollte die

Action/State	$Q(1,.)$	$Q(2,.)$	$Q(3,.)$	$Q(4,.)$	$Q(5,.)$	$Q(6,.)$
$Q(.,\longrightarrow)$	$w_0 + 2w_1$	$w_0 + 2w_2$	$w_0 + 2w_3$	$w_0 + 2w_4$	$w_0 + 2w_5$	$2w_0 + w_6$
$Q(.,\dashrightarrow)$	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}

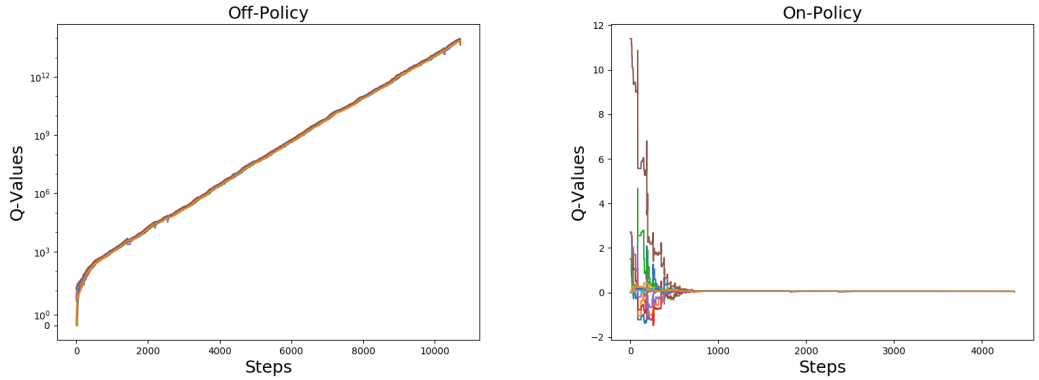
Tabelle 1.1: Parametrisierung der Aktionswerte für Baird's Gegenbeispiel

Approximation durch eine lineare Funktion kein Problem sein, die Parameter w müssten dazu lediglich gegen Null konvergieren, oder gegen einer der unendlich vielen anderen Linearkombinationen für die die Q-Werte alle Null ergeben. Es stellt sich jedoch heraus, dass insbesondere „off-policy" Verfahren zur Divergenz der Q-Werte führen können. Um dies zu zeigen, kann die Q-Learning Methode mithilfe des Gradientenverfahrens angewandt werden, das durch den Aktualisierungsschritt

$$\begin{aligned} w^{k+1} &= w^k + \alpha \cdot (R_{t+1} + \gamma \cdot \max_a \hat{Q}(s_{t+1}, a) \quad - \quad \hat{Q}(s_t, a_t)) \cdot \nabla_w Q(s_t, a_t) \\ &= w^k + \alpha \cdot (R_{t+1} + \gamma \cdot \max_a \phi(s_{t+1}, a)^T \cdot w^k - \phi(s_t, a_t)^T \cdot w^k) \cdot \phi(s_t, a_t) \end{aligned} \tag{1.32}$$

gegeben ist. Um zu gewährleisten, dass jeder Zustand und jede Aktion unendlich mal gesehen und ausgeführt wird, entscheidet sich der Agent mit einer Wahrscheinlichkeit von $\frac{1}{6}$ für die durchgezogene Linie und sonst für die gestrichelte Linie. Das Ergebnis des Q-Learning Verfahrens ist in Abbildung 1.6(a) dargestellt mit Lernrate $\alpha = 0.1$, Diskontierungsfaktor $\gamma = 0.999$ und Parameterinitialisierung $w^0 = (1, 1, 1, 1, 1, 1, 10, 0, 0, 0, 0, 0, 0)$. Wie man sieht, divergieren alle Q-Werte gegen unendlich, obwohl es ein triviales Problem ist. Dies liegt daran, dass durch die Aktualisierung eines der oberen Zustandswerte gleichzeitig auch der untere Zustandswert verändert wird, da alle Zustände von w_0 abhängen. Da $\max_a Q(6, a)$ so initialisiert wurde, dass es größer als alle anderen Werte ist, werden die Werte der oberen Zustände weiter nach oben gezerrt, wodurch $\max_a Q(6, a)$ noch größer wird. Der Wert für $Q(6, \longrightarrow)$ wird zwar bei jeder Aktualisierung nach unten geschätzt, wird jedoch durch die Strategie nicht oft genug besucht.

Interessanterweise tritt dieses Problem nur bei Off-Policy Verfahren auf, d.h. dass die Werte mit zukünftigen Aktionen approximiert werden, die vom Agent möglicherweise



(a) Off-Policy (b) On-Policy

Abbildung 1.6: Konvergenzverhalten aller Aktionswerte des Gegenbeispiels von Baird. Links wurden die Q-Werte mithilfe des Off-Policy Q-Learning Verfahrens berechnet. Rechts wurde das On-Policy SARSA genutzt. Es ist zu erkennen, dass die Werte des Q-Learning Verfahrens divergieren, während das SARSA Verfahren gegen die wahren Aktionswerte konvergiert

gar nicht ausgeführt werden. Beim Q-Learning Verfahren wird z.B. $Q(s_{t+1}, a_{t+1})$ durch $\max_a \hat{Q}(s_{t+1}, a)$ ersetzt obwohl $a_{t+1} \neq \arg\max_a Q(s_{t+1}, a)$ gelten kann. Wenn der Agent sich stattdessen für eine Aktion im nächsten Zustand entscheidet, bevor die Aktualisierung des aktuellen Zustands stattfindet, so kann das Beispiel stabilisiert werden. In Abbildung 1.6(b) sind die Q-Werte zu sehen, wenn $\max_a \hat{Q}(s_{t+1}, a)$ aus Gleichung (1.32) mit $Q(s_{t+1}, a_{t+1})$ ersetzt wird. Dies wird als SARSA (State Action Reward State Action) Methode [29, Abschnitt 6.4] bezeichnet und ist ein „On-Policy“ Verfahren. Der Nachteil eines On-Policy Verfahrens ist jedoch zum einen, dass die Aktualisierung der Werte und Entscheidungen des Agenten asynchron stattfinden und zum anderen, dass die Erkundungsstrategie gleichzeitig Teil der gelernten Strategie sein muss, was zu einer schlechteren Performanz führen kann.

Die Konvergenz von On-Policy Temporal Difference Verfahren mit linearer Funktionsapproximation wurde unter anderem von Tsitsiklis und Van Roy [19] untersucht. Darin

wurde eine Garantie für die Konvergenz solcher Verfahren unter einigen Nebenbedingungen hergeleitet. Gleichzeitig haben die Autoren auch ein Beispiel konstruiert, welches zeigen soll, dass die Konvergenz der Verfahren mit nichtlinearer Funktionsapproximation nicht gewährleistet werden kann. Doch obwohl man für lineare Funktionsapproximationen eine Konvergenzgarantie hat, sind diese für viele reale Probleme dennoch nicht geeignet, da diese höchstens gegen die beste lineare Funktion konvergieren. Somit funktioniert Temporal Difference Learning mit nichtlinearer Funktionsapproximation in der Praxis oft sogar besser als mit linearer Funktionsapproximation. Tatsächlich schreiben die Autoren der Arbeit, dass das theoretische Gegenbeispiel nicht von der Nutzung solcher Verfahren abhalten sollte, sondern die Probleme illustrieren sollten, die dabei auftreten können. [19, Einleitung]

1.5 Neuronale Netzwerke

Neuronale Netzwerke stammen ursprünglich aus dem Supervised Learning Bereich und sind aktuell sehr beliebt um nichtlineare Zusammenhänge zwischen gegebenen Daten herzustellen. Neuronale Netze sollen biologische Neuronen und ihre Funktionsweise in unserem Gehirn imitieren. Typischerweise hat ein neuronales Netzwerk mehrere Schichten, nämlich eine Eingangsschicht (Input Layer), mindestens eine versteckte Schicht (Hidden Layer) und eine Ausgangsschicht (Output Layer). In Abbildung 1.7 ist ein neuronales Netzwerk beispielhaft dargestellt. Man kann sich ein neuronales Netzwerk wie eine Verkettung von linearen Funktionen vorstellen mit Parametern w, die durch Aktivierungsfunktionen zwischen den Schichten zu einer großen nichtlinearen Funktion werden. Hierfür können unterschiedliche Aktivierungsfunktionen genutzt werden, jedoch sollte sie zumindest schwach differenzierbar sein und eine einfache Ableitung besitzen, zum Beispiel die Sigmoid oder Tangens Hyperbolicus Funktion. Der Prozess des Lernens des Netzwerks beinhaltet die Optimierung der Gewichte, um eine Zielfunktion namens Verlustfunktion zu minimieren. Eine sehr beliebte Verlustfunktion ist der mittlere quadratische Fehler

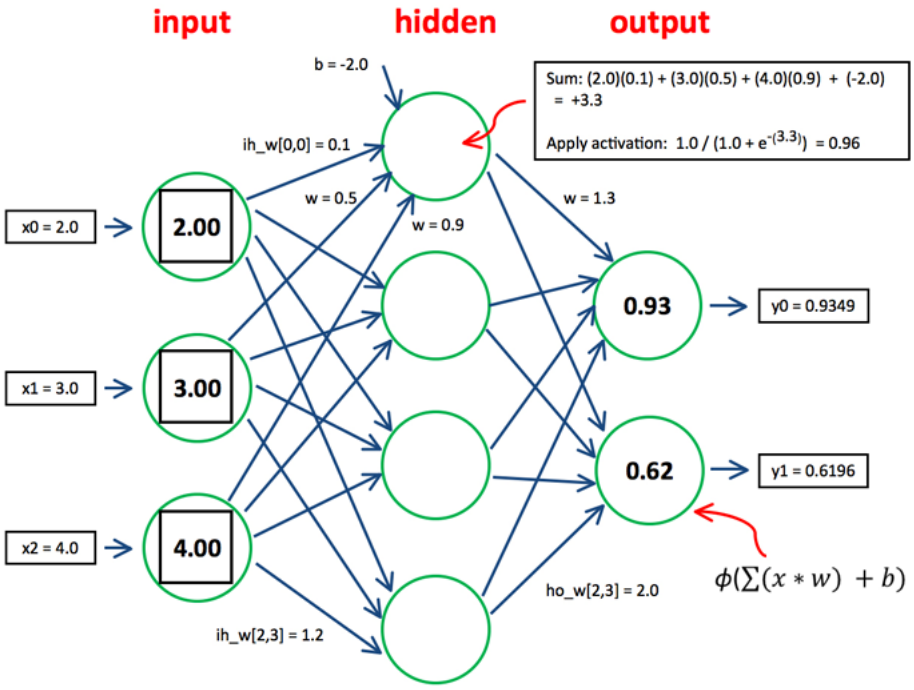


Abbildung 1.7: Neuronales Netzwerk. Bild aus [10, Fig. 2]

$$MSE(w) = \frac{1}{2}\mathbb{E}[(Y - f(X; w, b))^2]. \tag{1.33}$$

Dabei sind Y die Werte die vorhergesagt werden sollen und von den Inputdaten X abhängen. Um ein neuronales Netzwerkmodell anzulernen, wird typischerweise der Backpropagationsalgorithmus verwendet [27], der aus zwei Schritten besteht. Der erste Schritt besteht in der Weiterleitung der Daten (in Vektorform) über das Netzwerk, um die Ausgabewerte zu erzeugen, indem die Eingänge mit den jeweiligen Verbindungsgewichten multipliziert werden und die spezifische Aktivierungsfunktion auf diese Werte angewendet wird und die Biaswerte addiert werden. Der erzeugte Ausgangsvektor wird mit dem wahren Ausgangsvektor für diesen Eingang verglichen und die Fehlerdifferenzen an den einzelnen Ausgabeeinheiten berechnet. Der zweite Schritt im Algorithmus besteht aus der Berechnung der Gewichtsgradienten bzgl. des Fehlers. Diese beiden Schritte werden mehrfach im Rahmen der numerischen Lösung des nichtlinearen Ausgleichproblems (1.33)

benutzt, bis das Netzwerk die spezifische Verlustfunktion in einem zufriedenstellenden Maße reduziert hat.

1.6 Netzwerkarchitekturen

Eines der in dieser Arbeit verwendeten neuronalen Netze ist das **Convolutional Neuronal Network (CNN)** [20], das die spezifische Funktionalität des menschlichen Gehirns, den „visuellen Kortex“, nachbildet, der visuelle Informationen verarbeitet. CNNs führen Faltungsschichten (Engl.: Convolutional Layer) ein, die aus Filtern bestehen, die relativ kleine Matrizen darstellen, die mit den Eingangsdaten multipliziert werden. Diese Filter gleiten über die Daten, was zur endgültigen Ausgabe aus einer Faltungsschicht führt. Nach (mehreren) Faltungsschichten werden so genannte Pooling-Schichten verwendet, um die Eingabegröße und die Anzahl der lernbaren Parameter zu reduzieren und ein Overfitting zu vermeiden. Pooling-Filter gleiten über die Daten und reduzieren sie, indem sie ein Aggregat wie das Maximum oder den Durchschnitt eines bestimmten Teilbereichs der Daten nehmen.

Ein weiteres hier verwendetes Netzwerk ist das **Long Short-Term Memory Netwerk (LSTM)** [17]. LSTMs gehören zu der Klasse der recurrent neuronal networks (RNN), die aus Netzwerken besteht, die Schleifen haben, um Informationen basierend auf vergangenem Wissen zu erhalten. In Abbildung 1.8 ist der Aufbau einer entfalteten RNN Schicht zu sehen. Im Gegensatz zu einem vollständig verbundenem neuronalen Netzwerk, in dem nur der statische Input x einen Einfluss auf den Output hat, wird in einem RNN für jeden Zeitschritt x_t auch ein sogenannter „hidden state“ h_t berechnet, der sowohl den output o_{t+1} als auch den nächsten hidden state h_{t+1} beeinflusst. Dieser hidden state lässt sich als „Gedächtnis“ des Netzwerks interpretieren, da dieser die Informationen der Daten aus den vorherigen Zeitschritten beinhaltet.
Somit sind RNNs in der Lage, sich den vergangenen Kontext der Eingabedaten zu merken,

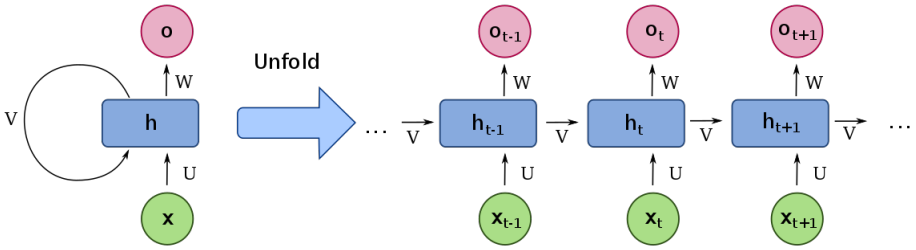


Abbildung 1.8: Aufbau einer RNN Schicht. Bild aus [10]

sind aber mit zunehmender Sequenzgröße nicht in der Lage, den langfristigen Kontext über die Zeit in der Sequenz zu erfassen. Des Weiteren leiden RNNs oft unter dem „verschwindenden Gradienten Problem“, d.h. dass durch zu viele Zeitschrittiterationen die Gradienten gegen Null konvergieren, sodass sich die Gewichtsparameter des Modells beim Lösungsverfahren nicht mehr verändern können.

LSTMs sind hingegen in der Lage, Informationen über lange Sequenzen zu speichern und lösen das „verschwindende Gradienten Problem“. In Abbildung 1.9 ist der Aufbau einer LSTM Zelle dargestellt. Der entscheidende Unterschied zu einem normalen RNN ist der „Cell State“ c_t, der in der Lage ist, sich relevante Daten aus der fernen Vergangenheit zu merken. Dies geschieht mithilfe der drei „Gates“ in der LSTM Zelle, die wie folgt definiert sind

$$
\begin{aligned}
F_t &= \sigma(V_F x_t + W_F h_{t-1} + b_F) \\
I_t &= \sigma(V_I x_t + W_I h_{t-1} + b_I) \\
O_t &= \sigma(V_O x_t + W_O h_{t-1} + b_O) \\
c_t &= F_t \circ c_{t-1} + I_t \circ tanh(V_c x_t + W_c h_{t-1} + b_c) \\
h_t &= O_t \circ tanh(c_t) \\
c_0 &= 0, h_0 = 0
\end{aligned}
$$

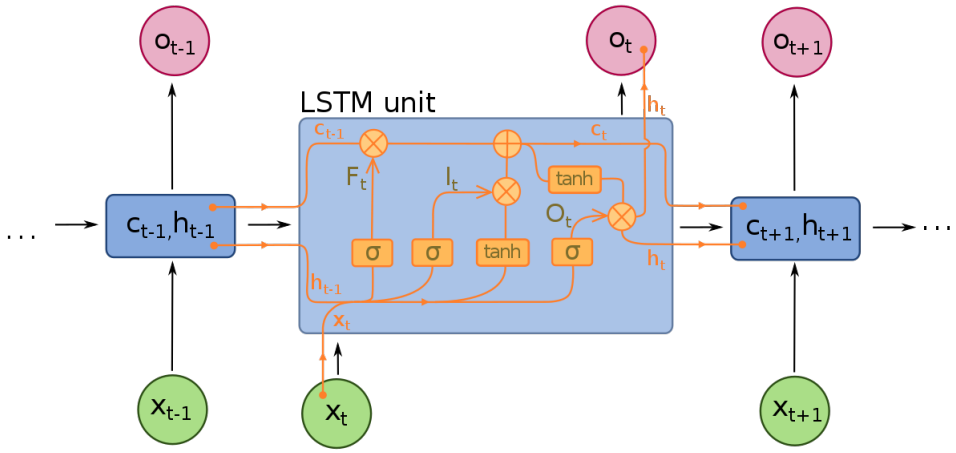


Abbildung 1.9: LSTM Zelle für einen Zeitschritt t. Bild aus [10]

Dabei sind $V_{F,I,O,c} \in \mathbb{R}^{n \times f}$ und $W_{F,I,O,c} \in \mathbb{R}^{n \times n}$ die Gewichtungsmatrizen und $b_{F,I,O,c} \in \mathbb{R}^n$ die Biasvektoren, die während des Trainings optimiert werden, mit $n \mathrel{\widehat{=}}$ Anzahl der versteckten Knoten und $f \mathrel{\widehat{=}}$ Anzahl der Features.

Das „Forget Gate " F_t reguliert welche Daten in c_t behalten bzw. vergessen werden sollen, in dem c_t elementweise mit einem Wert zwischen 0 und 1 multipliziert wird, der durch die Sigmoid Funktion am Ende von F_t erzeugt wird. Im „Input Gate" I_t wird c_t mit den relevanten Informationen aus dem Input x_t aktualisiert. Anschließend wird im „Output Gate" O_t der hidden state h_t in Abhängigkeit von c_t berechnet, der nach wie vor die Informationen aller vergangenen Daten mit sich trägt.

Eine weitere Netzwerkarchitektur sind die **Gated Recurrent Units (GRUs)** [12]. Diese wurden 2014 als Alternative zu den LSTM-Einheiten entwickelt, somit ist der Aufbau der beiden Netzwerke sehr ähnlich. Das GRU verzichtet jedoch auf den Cell State und speichert stattdessen alle relevanten Daten im Hidden State. Außerdem enthält eine GRU-Zelle nur zwei Gates statt der drei Gates des LSTMs. Diese sind wie folgt aufgebaut

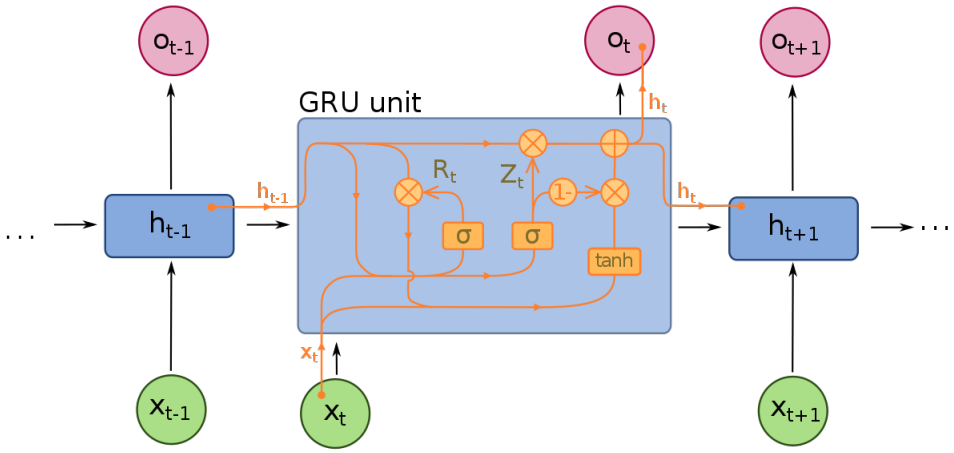


Abbildung 1.10: GRU Zelle für einen Zeitschritt t. Bild aus [10]

$$R_t = \sigma(V_R x_t + W_R h_{t-1} + b_R)$$

$$Z_t = \sigma(V_Z x_t + W_Z h_{t-1} + b_Z)$$

$$h_t = (1 - Z_t) \circ h_{t-1} + Z_t \circ tanh(V_h x_t + W_h(R_t \circ h_{t-1}) + b_h)$$

$$h_0 = 0$$

Dabei sind $V_{F,I,O,c} \in \mathbb{R}^{n \times f}$, $W_{F,I,O,c} \in \mathbb{R}^{n \times n}$, $b_{F,I,O,c} \in \mathbb{R}^n$ wieder die Gewichtungsmatrizen und Biasvektoren, die trainiert werden.

Zum einen ist das „Reset Gate“ R_t dazu da, irrelevante Daten aus dem hidden state auszusortieren und entspricht somit dem Forget Gate des LSTMs. Das „Update Gate“ Z_t aktualisiert den hidden state mit dem neuen Input x_t, analog zum Input Gate des LSTMs. Da es im GRU keinen Cell State gibt, entfällt die Notwendigkeit eines Output Gates, wodurch ein beachtlicher Teil an Berechnungen wegfällt. Somit ist das GRU deutlich effizienter als das LSTM Netzwerk.

1.7 Deep Q-Network

In der Arbeit „Human-level control through deep reinforcement learning“ von Mnih et al. wurde das „Deep Q-Network“ entwickelt [24]. Das DQN ist in der Lage die Q-Funktion mithilfe eines neuronalen Netzwerks zu approximieren. Um das DQN zu testen haben die Autoren es auf 49 verschiedenen Atari Computerspielen angewandt und diese mit dem bisher besten linearen Funktionsapproximations-Verfahren verglichen. Zusätzlich haben sie es mit der Performanz eines professionellen menschlichen Spielers verglichen. Die Ergebnisse sind in Abbildung 1.11 dargestellt. Wie man sieht, schlägt sich das DQN in den meisten Fällen besser als das lineare Verfahren, obwohl wir für Letzteres eine Konvergenzgarantie haben. Zudem erreicht das DQN bei 29 aus 49 Spielen eine höhere Punktzahl als ein Mensch, wobei die Performanz eines durchschnittlichen Spielers auf 75% der Performanz eines professionellen Spielers gesetzt wurde. Die Prozentzahlen ergeben sich wie folgt

$$\text{Relative DQN Performance} = \frac{Score_{DQN} - Score_Z}{Score_{ME} - Score_Z} \tag{1.34}$$

Dabei ist $Score_Z$ der Punktestand eines Agenten der jede Aktion mit gleichmäßiger Wahrscheinlichkeit ausführt und $Score_{ME}$ der Punktestand eines menschlichen Experten. Bemerkenswerterweise wurden alle unterschiedlichen Spiele mit den gleichen Parametern und ohne externes Wissen über die Umgebungen trainiert, während die linearen Approximationen alle für ihre Umgebung optimiert wurden. Dies zeigt, dass das DQN über verschiedene Aufgaben generalisierungsfähig ist.

Um das Training einigermaßen zu stabilisieren, haben die Autoren einige neue Techniken entwickelt. Der Zustandsraum $\mathcal{S}$ besteht bei jedem Spiel aus den Pixelwerten des angezeigten Spielzustands. Doch ein Bild reicht in der Regel nicht aus um das MDP markovsch zu machen, da man so z.B. bei dem Spiel „Breakout“ nicht weiß ob sich der Ball nach oben oder unten bewegt. Aus diesem Grund wurden vier aufeinanderfolgende

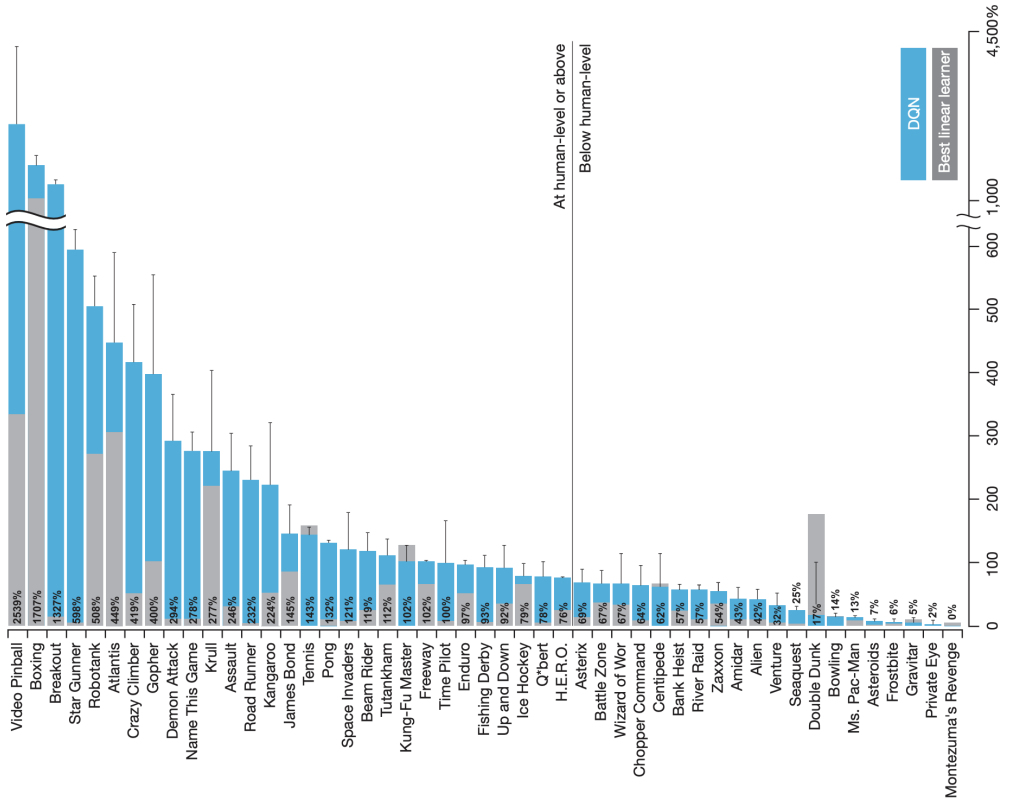


Abbildung 1.11: Ergebnisse des Deep Q-Networks auf 49 unterschiedlichen Atari Spielen [24, Fig. 3]

Bilder als ein Zustand definiert. Die Pixelwerte dieser Zustände wurden anschließend mit einem CNN verarbeitet. Die Aktionen werden hingegen durch den Output des Netzwerks modelliert, da ein neuronales Netzwerk in der Lage ist mehr als einen Output gleichzeitig zu generieren. Somit ist das Neuronale Netzwerk eine Funktion $\hat{Q} : S \to \mathbb{R}^{|\mathcal{A}|}$ für die jeder Output dem Aktionswert einer Aktion $a \in \mathcal{A}$ entspricht, d.h.

$$Q(s_t, a_t) \approx \delta_{a_t}^T \cdot \hat{Q}(s_t; \theta), \tag{1.35}$$

wobei $\delta_{a_t} \in \{0, 1\}^{|\mathcal{A}|}$ der Einheitsvektor bzgl. Aktion a_t ist.

Eine wichtige Eigenschaft, die beim Supervised Learning in der Regel angenommen wird ist, dass die Daten mit denen ein Modell trainiert wird unabhängig und identisch verteilt

sind. Dies ist insbesondere bei on-policy Verfahren nicht erfüllt, da die Beobachtungen mit denen die zu approximierende Funktion aktualisiert wird, zeitlich korreliert sind. Aus diesem Grund wird für das DQN ein „Erfahrungs"-Vektor $\mathcal{D}$ eingeführt, indem das Tupel $(s_t, a_t, r_{t+1}, s_{t+1})$ nach jeder ausgeführten Aktion gespeichert wird. Statt die Gewichte des Netzwerks mit der aktuellen Beobachtung zu aktualisieren, wird nun nach jeder ausgeführten Aktion eine gleichmäßig verteilte Stichprobe der Größe N des Erfahrungsvektors $d \sim \mathcal{D}$ gezogen. Mithilfe dieser Stichprobe d können die Gewichte des Netzwerks mit dem stochastischen Gradientenverfahren wie folgt aktualisiert werden

$$\theta_{k+1} = \theta_k - \alpha \sum_{i=1}^{N} \nabla J_{d_i}(\theta_k), \tag{1.36}$$

wobei die Verlustfunktion $J(\theta)$ dem Temporal Difference Fehler des Q-Learning Verfahrens entspricht

$$\nabla_\theta J_{d_i}(\theta) = -(r_{t+1} + \max_a \hat{Q}_{d_i}(s_{t+1}, a; \theta) - \hat{Q}_{d_i}(s_t, a_t; \theta)) \cdot \nabla_\theta \hat{Q}_{d_i}(s_t, a_t; \theta). \tag{1.37}$$

Aus der Sicht des Supervised Learning Bereichs ist diese Verlustfunktion jedoch nach wie vor problematisch. Denn wenn die Gewichte des Netzwerks nach (1.37) aktualisiert werden, so wird gleichzeitig auch die „wahre" abhängige Variable verändert, die wir mit

$$Q(s_t, a_t) \approx r_{t+1} + \max_a \hat{Q}(s_t, a; \theta) \tag{1.38}$$

lediglich approximieren. Die Probleme, die dabei insbesondere in Verbindung mit Q-Learning auftreten können, wurden bereits in Baird's Gegenbeispiel dargestellt. Um die abhängige Variable zu stabilisieren wurde ein zweites sogenanntes Zielnetzwerk $\hat{Q}'(s_t, a_t; \theta')$ definiert. Dieses Zielnetzwerk wird nun benutzt um die Aktionswerte des nächsten Zustands zu approximieren, ohne dabei dessen Gewichte zu verändern

$$J(\theta) = \frac{1}{2}\mathbb{E}[(R_{t+1} + \hat{Q}'(s_t, a_t; \theta') - \hat{Q}(s_t, a_t; \theta)]. \tag{1.39}$$

Nach n Lerniterationen, werden die Zielgewichte anschließend mit den Gewichten des „echten“ Netzwerks aktualisiert:

$$\theta'_k = \theta_k \qquad \forall k : \quad k \,\%\, n = 0 \tag{1.40}$$

In Algorithmus 1 ist der vollständige DQN Algorithmus als Pseudocode dargestellt.

Algorithm 1 Deep Q-Network Algorithmus

Initialisiere Erfahrungsvektor $\mathcal{D}$ als leere Liste
Initialisiere Aktionswerte Netzwerk $\hat{Q}$ mit zufälligen Gewichten θ
Initialisiere Zielnetzwerk $\hat{Q}^{'}$ mit Gewichten $\theta^{'} = \theta$
Initialsiere $\varepsilon = 1$
for Episode $= 1 \rightarrow M$ **do**
    Initialisiere s_1
    for t$= 1 \rightarrow T$ **do**
        if $\varepsilon > rand(0, 1)$ **then**
            Wähle zufällige Aktion a_t aus $\mathcal{A}$
        else
            Setzte $a_t = \arg\max_a \hat{Q}(s_t, a; \theta)$
        Führe Aktion a_t in der Umgebung aus und erhalte Belohnung r_{t+1} und nächsten Zustand s_{t+1}
        Speichere Tupel $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
        for $i = 1 \rightarrow N$ **do**
            Ziehe zufällige Stichproben $(s_i, , a_i, r_{i+1}, s_{i+1}) \sim \mathcal{D}$
            if $s_{i+1} = s_T$ **then**
                Setze $y_i = r_{i+1}$
            else
                Setze $y_i = r_{i+1} + \gamma \max_a \hat{Q}^{'}(s_{t+1}, a; \theta^{'})$
        optimiere Gewichte θ in die Richtung des steilsten Abstiegs bzgl. der Verlustfunktion $(y - \hat{Q}(s, a; \theta))^2$
        Aktualisiere alle n Lernschritte $\theta^{'} = \theta$

2 Energiemarktanalyse

In diesem Kapitel wird die Anwendung von Reinforcement Learning Algorithmen zur agentenbasierten Modellierung des Energiemarktes vorgestellt. Zunächst wird der Day-Ahead Markt des Elektrizitätssektors betrachtet, bei dem mehrere Agenten gegeneinander an einer Auktion zum Verkauf ihrer produzierten Elektrizität teilnehmen. Dabei wird insbesondere eine Dissertation, die sich bereits mit dem Thema beschäftigt hat, analysiert und es werden Vorschläge zur Verbesserung der Ergebnisse vorgestellt. Anschließend wird der Elektrizitätshandel mit Energiespeichern betrachtet.

2.1 Day-Ahead Markt

Im Rahmen der Dissertation „Engineering Interrelated Electricity Markets“ von Dr. Anke Weidlich [31] wurde eine Multiagentenbasierte Modellierung des Energiemarktes untersucht. Zunächst wurde die Anwendung unterschiedlicher Verfahren auf einem einfachen Modellbeispiel überprüft. Anschließend wurde ein komplexes Modell des Energiemarktes erstellt, worauf die Verfahren, die im einfachen Beispiel erfolgreich waren, angewandt wurden. Die Simulationsergebnisse wurden dabei mit realen Datensätzen des Deutschen Energiemarktes des Jahres 2006 verglichen. Nachdem die Verfahren durch die realen Datensätze bestätigt worden sind, wurde durch Änderungen des Modells, der Effekt von politisch motivierten Regulierungen des Marktes überprüft. So wurde z.B. gemessen, wie sich der Preis von Elektrizität für den Verbraucher verändert, wenn der oligopolistische Markt, der zum größten Teil von vier großen Unternehmen kontrolliert wird, in einen kompetitiven Markt aufgespalten wird. Das Verfahren, das sich dabei am besten bewährt

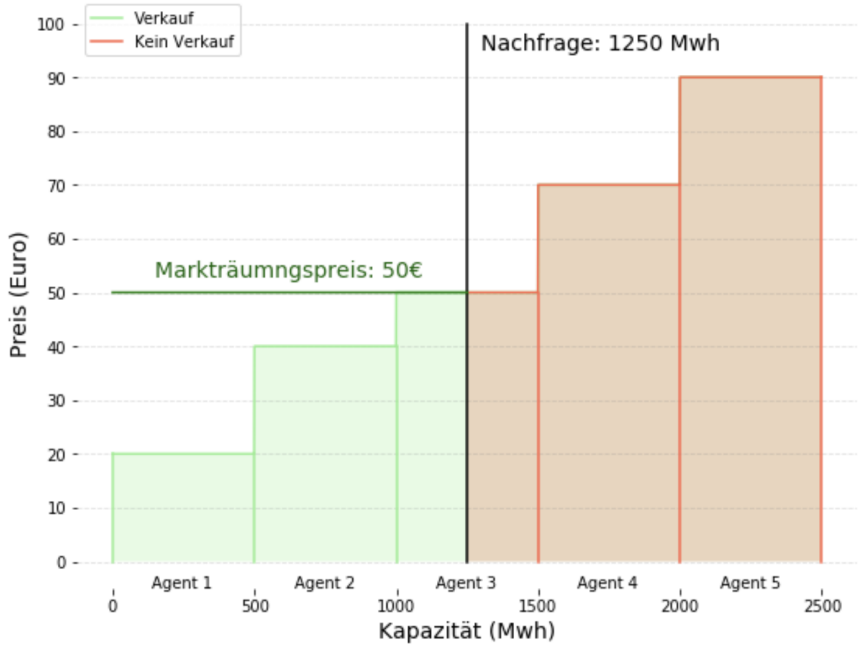
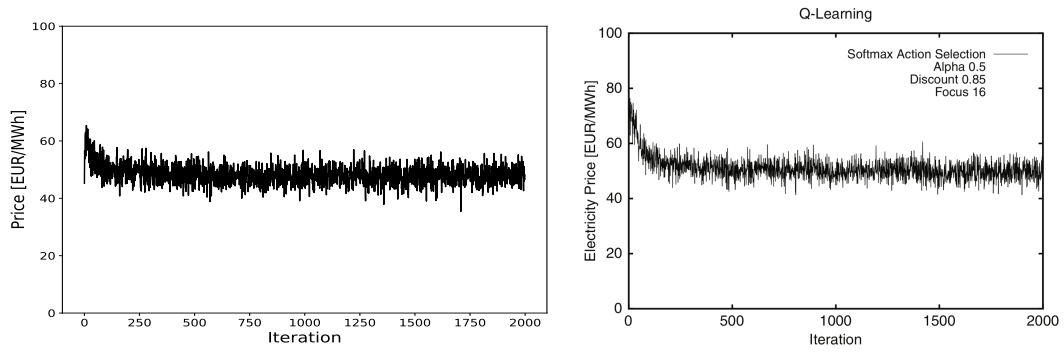


Abbildung 2.1: Beispiel einer Markträumung einer Day Ahead Auktion mit fünf Energieproduzenten

hat, ist das Q-Learning Verfahren.

Das einfache Modell, das in der Dissertation in Kapitel 4 vorgestellt wurde, modelliert fünf Agenten, die jeweils über einen Generator verfügen und versuchen ihre damit hergestellte Energie auf dem „Day-Ahead“ Markt zu einem möglichst hohen Preis zu verkaufen. Der Markträumungspreis wird dabei über eine Auktion ermittelt, in der jeder Agent einen Preis pro MWh und die Menge, die er bereit ist zu verkaufen, bietet. Nachdem alle ihre Gebote abgegeben haben, werden diese Preise aufwärts sortiert. Anschließend wird der Preis bestimmt, zu dem die nachgefragte Menge an Strom von der angebotenen Menge gedeckt wird. Dieser Preis ist der Markträumungspreis, der an alle Agenten gezahlt wird, die unter oder genau diesen Preis geboten haben, während die die mehr geboten haben leer ausgehen. Die Agenten verpflichten sich dazu, die angebotene Energie zum resultierenden Marktpreis 24 Stunden nach der Markträumung zu liefern.

In Abbildung 2.1 ist der beschriebene Marktmechanismus exemplarisch dargestellt. Alle



(a) Rekonstruktion (b) Weidlich

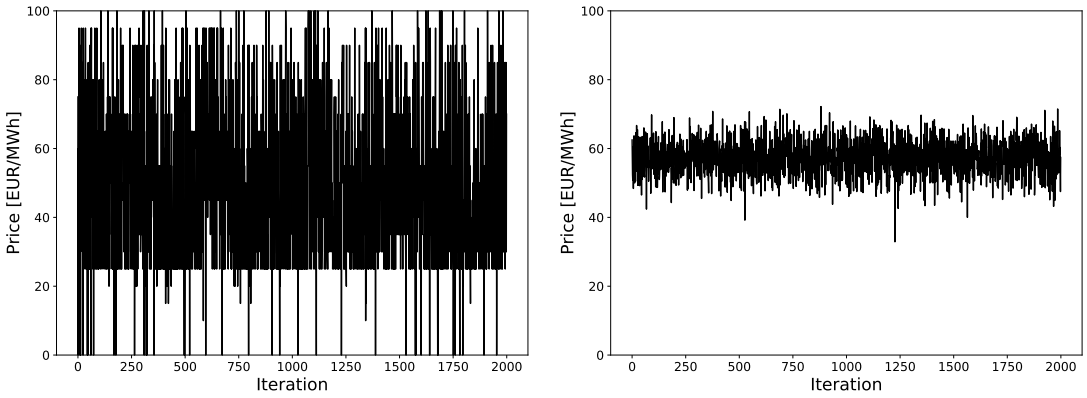
Abbildung 2.2: Vergleich der Ergebnisse mit Q-Learning. Mittelwert über 50 Simulationen

Agenten bieten dabei 500MWh zu unterschiedlichen Preisen an. Da die nachgefragte Energie 1250MWh entspricht, schaffen es nur drei der fünf Agenten ihre Energie erfolgreich zu verkaufen. Doch obwohl die ersten beiden Agenten jeweils 20€ bzw. 40€ geboten haben, kassieren sie dennoch den gesamten Markträumungspreis von 50€ pro MWh.

Alle Generatoren der Agenten verfügen über eine Kapazität von 500 MWh, die sie innerhalb einer Stunde herstellen können, jedoch unterscheiden sich ihre Marginalen Kosten (Abbildung 2.5(a)). Die Nachfrage wird dabei über den gesamten Simulationszeitraum als konstant und Preis-inelastisch angenommen und wurde für die Rekonstruktion auf 1250MWh gesetzt. Um das Problem mit dem Q-Learning Verfahren lösen zu können, muss ein geeignetes MDP definiert werden. Die Zustände bestehen dabei aus dem zuletzt gebotenem Preis und einem Indikator, ob der Agent damit erfolgreich war. Dies wurde in 6 unterschiedliche Zustände diskretisiert:

$$\begin{aligned}\mathcal{S} = \{&\textit{niedriger Preis/erfolgreich, niedriger Preis/nicht erfolgreich}\\ &\textit{mittlerer Preis/erfolgreich, mittlerer Preis/nicht erfolgreich}\\ &\textit{hoher Preis/erfolgreich, hoher Preis/nicht erfolgreich}\}\end{aligned} \tag{2.1}$$

Der Aktionsraum hingegen besteht aus 21 diskreten Preisen die zwischen 0€ und dem maximalen Preis liegen. In der Realität gibt es zwar keine Einschränkung an den gebotenen



(a) Markträumngspreis einer Simulation (b) Markträumungspreis mit zufälligen Aktionen

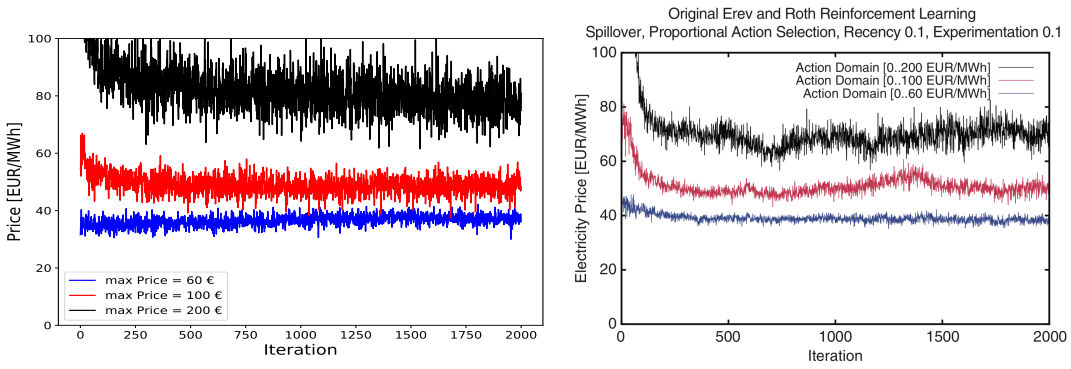
Abbildung 2.3: Links: Markträumungspreis nach einem Durchlauf über 2000 Markräumungen. Rechts: Mittelwert des Markträumungspreises über 50 Durchläufe mit zufälligen Aktionen

Preis, aber aus technischen Gründen muss eine Obergrenze eingeführt werden. Dieser Parameter hat auch einen großen Einfluss auf den Marktpreis wie man später sieht. Zu jedem gebotenen Preis gibt es zusätzlich fünf diskrete Mengen, die die Agenten jeweils anbieten können: $\{100MW, 200MW, 300MW, 400MW, 500MW\}$. Somit ist $|\mathcal{A}| = 21 \cdot 5 = 105$. Die Belohnungen der Agenten betragen den Gewinn den sie während einer Auktion erhalten haben, relativ zu dem Gewinn, den Sie maximal erhalten können:

$$R_{t+1} := \frac{(mrp_t - GK) \cdot E_t}{\max R}, \tag{2.2}$$

wobei mrp_t der Markträumungspreis zur Stunde t ist, GK die Grenzkosten des Agenten und E_t die Menge der Elektrizität, die der Agent verkaufen will. Somit gilt $|R| < 1$. Falls die angebotene Menge aller Agenten kleiner ist als die Nachfrage, so erhalten alle Agenten eine Belohnung von 0.

Die Ergebnisse aus der Dissertation von Weidlich konnten mithilfe des Q-Learning Verfahrens rekonstruiert werden, obwohl nicht alle Parameter aus den Experimenten gegeben waren. Von einer erfolgreichen Anwendung eines Verfahrens wird unter anderem erwartet, dass der Markträumungspreis innerhalb 2000 Iterationen konvergiert.



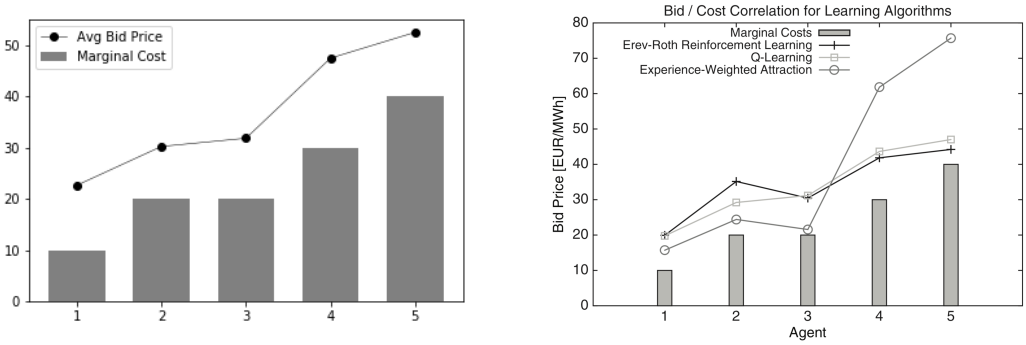
(a) Rekonstruktion Q-Learning

(b) Erev-Roth Algorithmus

Abbildung 2.4: Vergleich der Ergebnisse mit unterschiedlichem maximalen Preis

Aus Abbildung 2.2(b) wurde geschlossen, dass dies für das Q-Learning Verfahren der Fall ist, was aufgrund der relativ hohen Varianz eher fragwürdig ist. Außerdem sind in diesem Plot lediglich die Mittelwerte über 50 unterschiedliche Simulationen zu jeder Iteration dargestellt. Schaut man sich hingegen das Ergebnis aus nur einer Simulation in Abbildung 2.3(a) an, so erhärten sich die Zweifel an der Konvergenz nur noch mehr. Ob das Experiment aus der Dissertation das gleiche Verhalten zeigt, kann allerdings nicht überprüft werden, da die Untersuchung der Konvergenz nicht über die bereits vorgestellten Ergebnisse hinausgeht. In Abbildung 2.3(b) ist weiterhin das Simulationsergebnis durch komplett zufällige Entscheidungen aller Akteure dargestellt. Die Varianz ist im Vergleich zu 2.2(b) zwar etwas höher, jedoch besteht auch eine unverkennbare Ähnlichkeit. Außerdem hängt das Verfahren stark von den gewählten Parametern ab. In Abbildung 2.4(b) werden drei Simulationen verglichen, die sich lediglich in der Wahl des maximalen Preises unterscheiden. Die Ergebnisse aus 2.4(b) basieren zwar auf dem Erev-Roth Algorithmus, jedoch wurde mit Q-Learning das gleiche Verhalten beschrieben.

Was das Verfahren jedoch rettet, ist die individuelle Rationalität der Agenten. In Abbildung 2.5(b) sind die durchschnittlichen Gebote der Agenten in Abhängigkeit ihrer Marginalen Kosten dargestellt. Wie man sieht, schaffen es die Agenten mit niedrigen



(a) Rekonstruktion (b) Weidlich

Abbildung 2.5: Mittelwerte des gebotenen Preises und Marginale Kosten

Marginalen Kosten ihren strategischen Vorteil auszunutzen, indem sie ihre Energie für einen niedrigeren Preis anbieten als ihre Konkurrenten mit höheren Marginalen Kosten. Allerdings sind dies die Mittelwerte über 50 Simulationen, die wiederum über 2000 Simulationsschritte gemittelt wurden.

Für das beschriebene Verhalten kann es verschieden Gründe geben: Eine wichtige Eigenschaft der Modellierung des Energiemarktes, ist dass es sich dabei um ein Multi-Agentenbasiertes Problem handelt. Dies impliziert, dass das zugrundeliegende MDP nicht stationär ist, da die erhaltenen Belohnungen eines Agenten nicht nur von den eigenen Aktionen und Zuständen abhängen, sondern auch von denen der anderen Agenten, die ihr Verhalten während einer Simulation anpassen. Zwar existieren bereits einige Verfahren, die versuchen mit dieser Instationarität besser umzugehen [25, 22, 7], jedoch ist das Lösen von Multi-Agentenbasierten Problemen nach wie vor Gegenstand aktueller Forschung.

Des Weiteren ist eine hinreichende Bedingung zur Konvergenz des Q-Learning Verfahrens, dass alle Zustandsaktionspaare unendlich oft besucht werden müssen. Dies kann für das beschriebene Modell jedoch nicht mal ansatzweise erfüllt werden, da es mit 6 Zuständen und 105 Aktionen insgesamt 615 unterschiedliche Zustandsaktionspaare gibt. Da eine Simulation lediglich aus 2000 Markträumungen besteht, können alle Zustandsaktionspaare

- bei einer gleichmäßigen Ausführung - höchsten 3-4 mal besucht und ausgeführt werden.

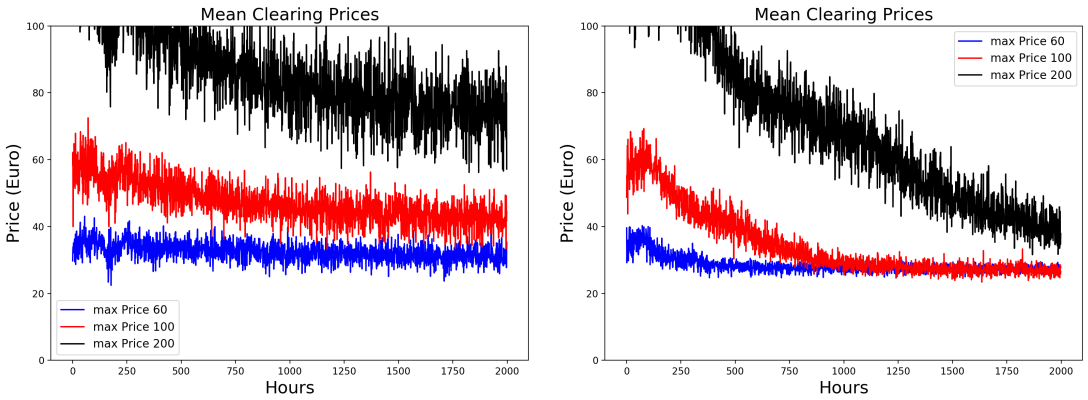
Ein noch größeres Problem dieser Modellierung ist jedoch die Wahl des Zustandraumes. Denn es geht aus keinem der 6 Zustände hervor, welche Aktion in diesen Zuständen optimal wäre. Zudem sind alle Informationen, die durch den Zustandsraum modelliert sind, bereits durch den Algorithmus selbst implizit gegeben. Die Höhe des zuletzt gebotenen Preises ist durch den Aktionsraum gegeben, während dessen Erfolg durch die Belohnungen modelliert ist. Somit gilt für dieses Problem:

$$Q(s,a) = Q(s',a) \qquad \forall s, s' \in \mathcal{S}, a \in \mathcal{A} \tag{2.3}$$

Die Q-Funktion hängt somit lediglich von den Aktionen ab und nicht von den Zuständen, wie sie hier definiert wurden. Trotzdem werden alle Q-Werte für jeden Zustand einzeln approximiert, wodurch das Problem komplexer gemacht wird, als es sein muss. Außerdem wird durch das Q-Learning Verfahren versucht die Bellman Gleichung (1.10) auszunutzen, obwohl es keinen Zusammenhang zwischen einer Aktion zur Stunde t und der zur Stunde $t+1$ gibt. Wenn man die Zustände aus diesem Problem rausnimmt, so kann man die Aktionswerte als Funktion $Q : \mathcal{A} \to \mathbb{R}$ definieren. Ein solches Problem wird auch als „K-Armed-Bandits“ Problem [29, Abschnitt 2.1] bezeichnet, da man es analog zu einem einarmigen Banditen mit $K \in \mathbb{N}$ Armen betrachten kann, bei dem man für jeden Arm eine unterschiedliche Belohnung erhält. Das Ziel ist es den Arm (bzw. die Arme) zu finden, für den man die höchste Belohnung erhält. Die Aktionswerte können auch für ein K-Armed-Bandit Problem mithilfe eines Temporal Difference Ansatz approximiert werden. Doch da es keine Zustände mehr gibt, vereinfacht sich die Aktualisierungsformel (1.19) des Q-Learning Verfahrens zu:

$$Q_{k+1}(a_t) = Q_k(a) + \alpha(R_{t+1} - Q_k(a)) \tag{2.4}$$

Wenn man diese Regel verwendet um die Aktionswerte der Agenten zu berechnen, so ergibt sich ein Verlauf des Markträumungspreis wie er in Abbildung 2.6(a) dargestellt ist. Wie man sieht ist der gemittelte Verlauf des Markträumungspreises kaum von dem des Q-Learning Verfahrens in Abbildung 2.4(a) zu unterscheiden. Dennoch ist



(a) K-Armed-Bandit mit Belohnung nach (2.2) (b) K-Armed-Bandit mit Belohnung nach (2.5)

Abbildung 2.6: Markträumungspreis des K-Armed-Bandit Problems

insbesondere für die höheren maximalen Preise eine leicht absteigende Tendenz des Markträumungspreises zu erkennen. Dies spricht für eine Annäherung an die Realität, da sich der Markträumungspreis in der Regel nach den Agenten mit den niedrigsten Grenzkosten richtet. Da die Energien der 3 Agenten mit den niedrigsten Grenzkosten ausreichen um die nachgefragte Energie zu decken, sollte der Marktpreis somit gegen 30€ konvergieren, um die beiden Agenten mit Grenzkosten von 30€ bzw. 40€ vom Markt zu drängen.

Eine weitere Möglichkeit das Modell zu stabilisieren ist es, die Belohnungen der Agenten anzupassen:

$$r_{t+1} = \begin{cases} \frac{(mrp_t - GK)\cdot E_t}{\max R}, & \text{falls } a_t \text{ erfolgreich,} \\ \frac{(GK - p_t)}{\max R}, & \text{falls } a_t \text{ nicht erfolgreich,} \end{cases} \tag{2.5}$$

Im Falle eines erfolgreichen Gebots erhält ein Agent somit die gleiche Belohnung wie in (2.2). Wenn Aktion a_t jedoch nicht erfolgreich war, so wird der Agent mit einer negativen Belohnung bestraft, die größer ist, je höher der nicht erfolgreiche gebotene Preis p_t des Agenten war. Dadurch ist ein Agent in der Lage nicht nur zu erkennen, dass die Aktion falsch war, sondern auch wie stark diese Aktion von einer guten Entscheidung

abweicht. Dies führt auch dazu, dass das Problem stationärer wird, da die Belohnungen nicht mehr komplett von den Aktionen der konkurrierenden Agenten abhängen sondern auch von den eigenen Aktionen, wenn sie nicht erfolgreich sind. Die Ergebnisse mit Belohnungen nach (2.5) sind in Abbildung 2.6(b) dargestellt. Wie man sieht konvergiert der Verlauf des Markträumungspreises für alle maximalen Preise gegen einen einheitlichen Markträumungspreis von 30€, wie man es in der Realität auch erwarten würde.

Das hier vorgestellte Modell bietet allerdings noch keine vollständig realistische Abbildung des Day-Ahead Marktes, da wir hier von einer konstanten Elektrizitätsnachfrage ausgehen. Allerdings kann die Nachfrage in der Realität sehr volatil sein, wie man insbesondere im nächsten Abschnitt sehen wird. Dies führt in der Regel auch zu einem äußerst volatilen Markträumungspreis, da durch eine höhere Nachfrage auch Energieproduzenten mit höheren Grenzkosten zum Zuge kommen können. Somit hängt der Erfolg eines gebotenen Preises von der nachgefragten Menge an Elektrizität ab und es bietet sich an, die Nachfrage als Zustand eines Agenten zu definieren. Das Q-Learning Verfahren wäre allerdings auch hier nicht die richtige Wahl, da eine Preis-inelastische Nachfrage nicht von den Aktionen der Agenten beeinflusst wird. Stattdessen kann es als Kontextuelles Banditen Problem [29, Abschnitt 2.9] definiert werden, indem die Belohnungen eines K-Armigen-Banditen von einem externen Zustand (z.B. der Nachfrage) abhängen. Dies wurde bereits für ähnliche Problemstellungen erfolgreich eingesetzt [13]. Die Analyse eines solchen Problems würde jedoch über den Rahmen dieser Masterarbeit hinausgehen.

2.2 Energiespeicher-Markt

Eines der größten Hindernisse des Elektrizitätsmarktes ist, dass die produzierte Energie zum gleichen Zeitpunkt verbraucht werden muss. Gleichzeitig muss die Nachfrage an Elektrizität immer gedeckt sein, da die meisten gesellschaftlichen Strukturen darauf angewiesen sind. Dies hat zur Folge, dass der Handel von Elektrizität durch unterschiedliche zusammenhängende Marktorgane fließen muss, um die Stabilität des Stromnetzes zu gewährleisten. Außerdem führt dies zu einem sehr instabilen Marktreis pro MWh. Da nachts deutlich weniger Energie verbraucht wird als tagsüber, kann der Preis für eine produzierte MWh innerhalb 24 Stunden zwischen 4€ und 80€ liegen.

Eine naheliegende Lösung für dieses Problem ist es, Energiespeicherkraftwerke zu entwickeln, die in der Lage sind Energie zu speichern, solange die Nachfrage niedrig ist und diese wieder in das Stromnetz zu entladen, wenn sie gebraucht wird. Solche Energiespeicherkraftwerke sind zudem eine Voraussetzung für eine erfolgreiche Energiewende, die sich die Bundesregierung bis zum Jahr 2050 zum Ziel gesetzt hat. Da die Sonne nicht immer nur dann scheint, wenn wir sie brauchen, müssen wir die überschüssige Energie speichern um sie an regnerischen Tagen verwenden zu können. Es existieren schon viele Ansätze zur Speicherung von großen Mengen an Energie, jedoch nur wenige die bereits umgesetzt wurden [11]. Dennoch sollte der Einfluss solcher Speicherkraftwerke auf den Elektrizitätsmarkt untersucht werden um dessen Einführung von Anfang an richtig regulieren zu können. Aus diesem Grund wird in diesem Abschnitt ein Modell des Energiemarktes mit Speicheragenten vorgestellt, die durch Reinforcement Learning eine profitorientierte Strategie lernen sollen. Wie wir sehen werden, haben solche Speicherkraftwerke eine stabilisierende Wirkung auf den Marktpreis.

Im Jahr 2000 wurde in Deutschland das Erneuerbare Energie Gesetz (EEG) [15] eingeführt. Dieses Gesetz garantiert unter anderem den Vorrang für erneuerbare Energien zur Einspeisung in das Stromnetz. Aus diesem Gesetz entstand der Begriff der residualen Last, der auf eine Arbeit des Fraunhofer Instituts zurückzuführen ist [28]. Die residuale Last beschreibt die nachgefragte Elektrizität, die noch nicht von erneuerbaren Energien

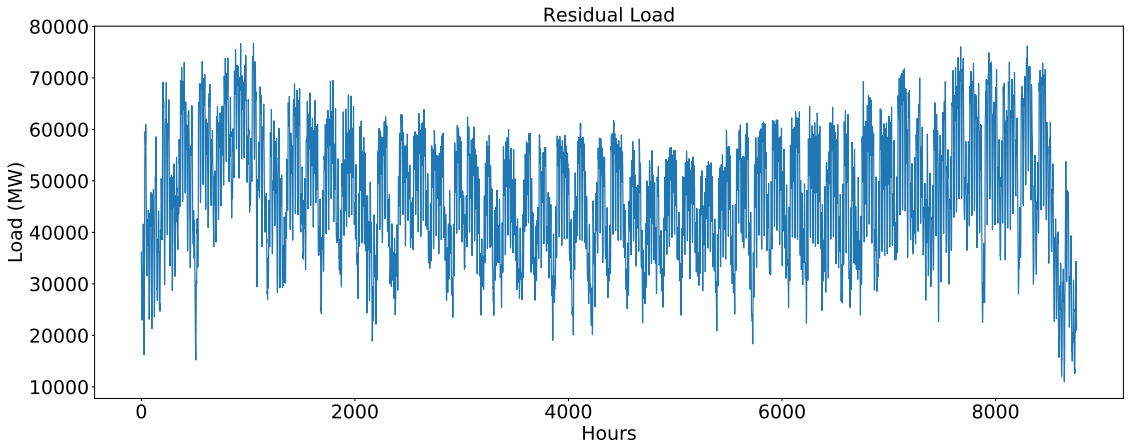


Abbildung 2.7: Residuale Last 2012

gedeckt werden kann und somit von herkömmlichen Stromerzeugern wie Kohlekraftwerken erzeugt werden muss. In Abbildung 2.7 ist die residuale Last Deutschlands für das Jahr 2012[1] stündlich aufgelöst dargestellt. Die Daten wurden vom Institut für Technische Thermodynamik des DLRs zur Verfügung gestellt. Wie man sehen kann ist die residuale Last im Winter relativ hoch und sinkt im Sommer wieder ab. Dies liegt zum einem an dem höheren Stromverbrauch im Winter und zum anderen an der größeren Verfügbarkeit an erneuerbaren Energien im Sommer. Noch bemerkenswerter als die saisonale Varianz ist jedoch die tägliche Varianz, sodass es sich für Speicherkraftwerkbetreiber lohnt, Energie mehrmals am Tag zu laden und zu entladen. Zur Modellierung des Marktes kann der Marktpreis pro MW h zu jeder Stunde t in abstrakter Form[2] durch ein Polynom dritten Grades wie folgt beschrieben werden:

$$\begin{aligned} p(x_t) =& 3{,}3 \cdot 10^{-13}\,€/(\mathrm{MW\,h^4}) \cdot x_t^3 - 4 \cdot 10^{-8}\,€/(\mathrm{MW\,h^3}) \cdot x_t^2 \\ &+ 2{,}5 \cdot 10^{-3}\,€/(\mathrm{MW\,h^2}) \cdot x_t - 27{,}8\,€/(\mathrm{MW\,h}) \end{aligned} \tag{2.6}$$

Dabei ist x_t die Residuale Last zur Stunde t in Mega Watt. Allerdings handelt es sich hierbei um ein sehr linear-dominiertes Polynom. Wenn man den Verlauf der residualen Last mit dem daraus resultierenden Preis vergleicht, so ist lediglich bei der Skalierung

[1] $t = 0$ entspricht dem 02.01.2012 um 00:00 Uhr.

[2] Formel wurde vom Institut für technische Thermodynamik des DLR zur Verfügung gestellt.

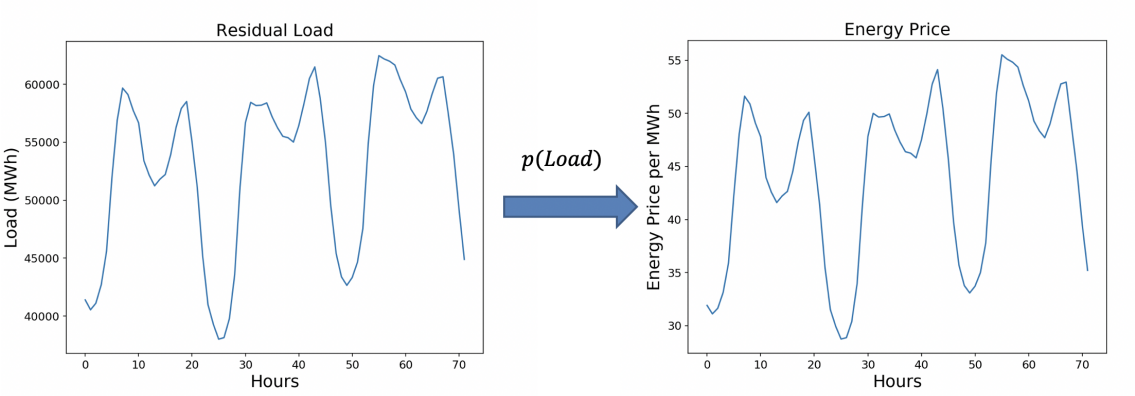
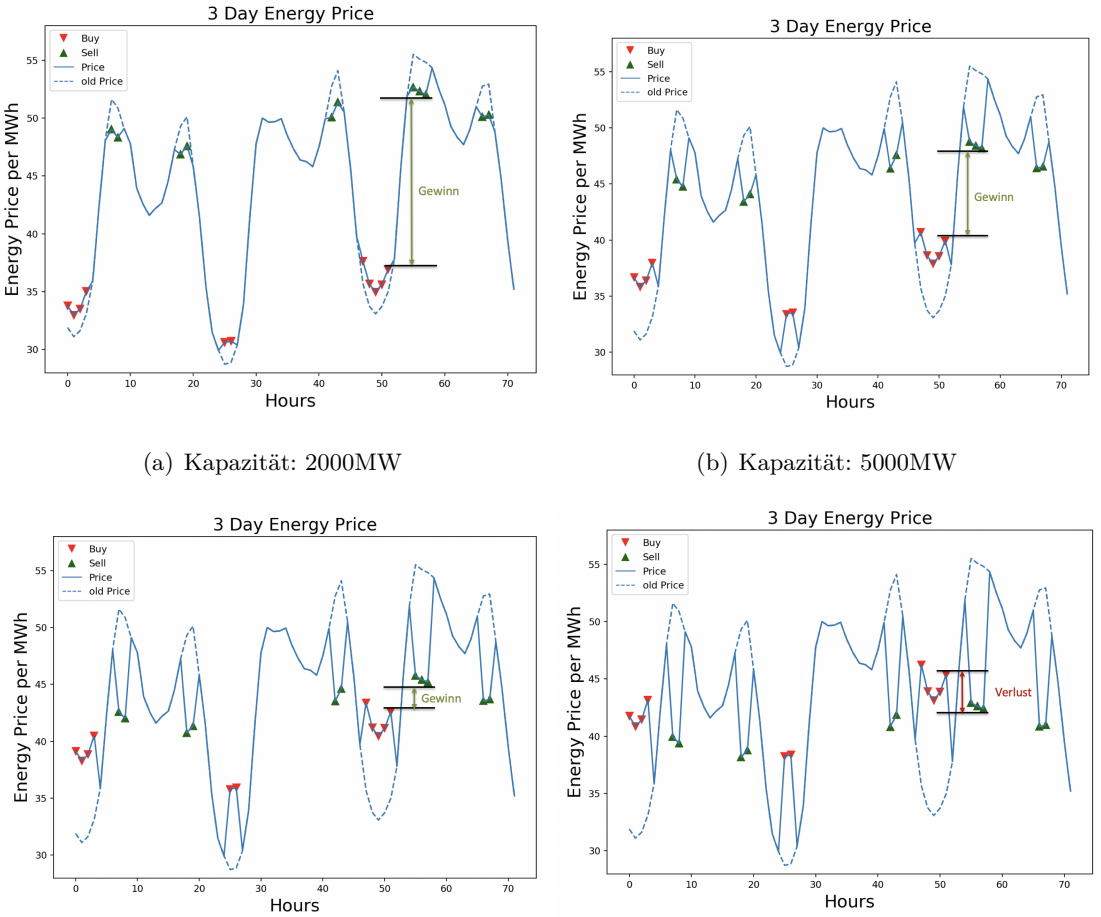


Abbildung 2.8: Residuale Last und Marktpreis für 3 Tage

ein signifikanter Unterschied zu erkennen, wie in Abbildung 2.8 zu sehen ist.

Wenn sich nun Speicherkraftwerke am Elektrizitätshandel beteiligen, beeinflussen sie durch ihre Handlungen den Verlauf der residualen Last und dementsprechend auch den daraus resultierenden Marktpreis. Wenn sie Energie kaufen möchten, so müssen herkömmliche Kraftwerke mehr Energie produzieren, wodurch der Marktpreis steigt. Wenn das Kraftwerk die geladene Energie wieder verkaufen soll, müssen herkömmliche Kraftwerke wiederum weniger produzieren, sodass der Marktpreis sinkt. Dieser Mechanismus ist in Abbildung 2.9 dargestellt. Jedes Dreieck stellt dabei einen Kauf bzw. Verkauf von Elektrizität eines Kraftwerkspeichers dar. Je höher die Kapazität des Speichers ist, desto kleiner wird auch die Gewinnspanne pro geladene und entladende MWh, bis der Punkt erreicht ist, an dem es sich gar nicht mehr lohnt (Abbildung 2.9(d)). Es ist jedoch zu beachten, dass die Speicherkraftwerke, die mit dem aktuellen Stand der Technik gebaut werden können, über deutlich kleinere Kapazitäten verfügen. So hat z.B. die Batterieplantage in Südaustralien, die von Tesla im Jahr 2017 gebaut wurde und einer der größten Kraftwerke seiner Art ist, eine Speicherkapazität von lediglich 129 MW h und einer Leistung von 100 MW [30]. Die Kapazitäten von alternativen Technologien können zwar durchaus größer sein, jedoch liegen diese, abgesehen von den größten Pumpspeicherkraftwerken, selten über 1000 MW h [16]. Somit ist der Effekt eines einzelnen Speicherkraftwerks auf den Marktpreis sehr gering. Wenn jedoch viele



(a) Kapazität: 2000MW

(b) Kapazität: 5000MW

(c) Kapazität: 7500MW

(d) Kapazität: 10000MW

Abbildung 2.9: Auswirkung von Energiespeicher auf den Marktpreis

weitere Agenten miteinsteigen, mit der Annahme, dass sie alle ungefähr zur gleichen Zeit Energie kaufen und verkaufen, so wird dies einen glättenden Effekt auf die residuale Last und den Marktpreis haben. Allerdings nimmt dabei auch die Lukrativität jedes einzelnen Kraftwerks ab.

Das Ziel der Speicherkraftwerkagenten ist es Energie zu kaufen und zu verkaufen, um ihren Gewinn dabei zu maximieren. Dazu müssen die Agenten einer guten Strategie folgen, die mithilfe von Reinforcement Learning gefunden werden kann. Doch um die

gelernten Strategien bewerten zu können, sollte man das Problem zunächst nach einer optimalen Strategie untersuchen.

2.2.1 Existenz und Eindeutigkeit der Optimalen Lösung

Wir wollen nun das oben beschriebene Modell als Optimierungsproblem definieren und zeigen, dass eine eindeutige Lösung dafür existiert. Da das Ziel die Gewinnmaximierung ist, kann man das Energiespeichermodell als nichtlineares Optimierungsproblem wie folgt betrachten

$$\min_{E \in V} f(E) \coloneqq \min_{E \in V} \int_0^T \dot{E} \cdot p(l + \dot{E}) dt \tag{2.7}$$

mit

$$V \coloneqq \{E \in W_0^{1,p}(0,T) : 0 \leq E \leq E_{max}\}, \tag{2.8}$$

wobei $l(t) \in L^\infty$ die residuale Last und $E(t)$ den Ladezustand des Energiespepeichers zum Zeitpunkt t darstellt und $\dot{E}(t)$ dessen zeitliche Ableitung. Der Raum $W_0^{1,p}(0,T)$ bezeichnet dabei den Sobolevraum, d.h. alle Funktionen $v \in L^p(0,T)$ mit der Norm

$$||v||_{W_0^{1,p}} \coloneqq (\sum_{|\alpha| \leq 1} \int_0^T D^\alpha v^p)^{\frac{1}{p}} \tag{2.9}$$

dessen partielle schwache Ableitungen der ersten Ordnung auch in L^p liegen. Um zu gewährleisten, dass das Integral aus (2.7) tatsächlich existiert, sollte $p = 4$ gewählt werden, da $p(E)$ aus Gleichung (2.6) ein Polynom dritten Grades ist. Der Wert $E_{max} \in \mathbb{R}^+$ beschreibt die Kapazität des Speichers, der von der Funktion $E(t)$ nicht überschritten werden darf. Bevor das Minimierungsproblem gelöst werden kann, sollte man zunächst untersuchen ob es auch eine eindeutige Lösung besitzt. Die folgende Analyse dazu basiert zum größten Teil auf dem Buch „Funktionalanalysis“ von Werner [32, Kapitel III.5].

Zunächst werden einige Definitionen benötigt [32, Def. III.5.7]:

Definition 2.2.1. *Sei $f : V \to \mathbb{R}$ ein Funktional auf einem normierten Raum V:*

- *f heißt schwach halbstetig von unten, falls stets*

$$x_n \xrightarrow{n\to\infty} x,\ f(x_n) \leq c \Rightarrow f(x) \leq c. \tag{2.10}$$

- *f heißt koerzitiv, falls*

$$||x_n|| \xrightarrow{n\to\infty} \infty \Rightarrow f(x_n) \xrightarrow{n\to\infty} \infty. \tag{2.11}$$

- *f heißt konvex, falls*

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y) \qquad \forall x, y \in V, 0 \leq \lambda \leq 1. \tag{2.12}$$

Um die Eindeutigkeit des Minimierungsproblems (2.7) zu zeigen kann der folgende Satz, der an [32, Satz III.5.8] angelehnt ist, genutzt werden:

Satz 2.2.2. *Sei X ein reflexiver Banachraum, und sei $V \subset X$ eine abgeschlossene und konvexe Teilmenge von X. Sei weiterhin $f : X \to \mathbb{R}$ schwach halbstetig von unten und koerzitiv. Dann existiert eine Stelle x_0 mit $f(x_0) = \inf_{x\in V} f(x)$.*

Beweis. Sei $m = \inf_{x\in V} f(x)$ und x_n eine Folge in V mit $f(x_n) \to m$. Aufgrund der Koerzitivität von f ist die Folge x_n beschränkt, und da X ein reflexiver Banachraum ist, existiert nach dem Satz von Eberlein-Šmulian eine schwach konvergente Teilfolge $x_{n_k} \to x_0$. Da V eine abgeschlossene und konvexe Teilmenge von X ist und alle Folgenglieder $x_n \in V$ sind, folgt aus dem Lemma von Mazur, dass auch $x_0 \in V$. Sei nun $c > m$. Dann existiert ein $k_0 \in \mathbb{N}$ mit $f(x_{n_k}) \leq c$ für $k \geq k_0$. Da f nach Voraussetzung schwach halbstetig von unten ist, folgt auch $f(x_0) \leq c$. Weil $c > m$ beliebig gewählt wurde und $f(x_{n_k}) \to m$ gilt, muss auch $m \leq f(x_0) \leq m$ gelten und somit folgt $f(x_0) = m$. □

Aus [2, Satz 6.10, Beispiel 6.11(3)] wissen wir, dass $W_0^{1,p}$ ein reflexiver Banachraum ist. Um zu zeigen, dass das Funktional f schwach halbstetig von unten ist, kann das folgende Lemma genutzt werden:

Lemma 2.2.3. *Ist V ein normierter Raum und $f : V \to \mathbb{R}$ konvex und stetig, so ist f schwach halbstetig von unten.*

Beweis. Siehe [32, Lemma III.5.9]. □

Somit reicht es zu zeigen, dass das Funktional f aus Gleichung (2.7) stetig und konvex ist.

Sei $x_n \in V \; \forall n \in \mathbb{N}$ eine Folge mit $x_n \xrightarrow{x\to\infty} x \in V$. Es gilt

$$\dot{x}_n \cdot p(l + \dot{x}_n) - \dot{x} \cdot p(l + \dot{x}) = (\dot{x}_n - \dot{x}) \cdot P_3(\dot{x}_n, \dot{x}, l), \tag{2.13}$$

wobei $P_3(\dot{x}_n, \dot{x}, l)$ ein Polynom dritten Grades ist und da nach Voraussetzung $\dot{x}_n, \dot{x}, l \in L^p$ gilt, folgt $P_3 \in L^{1,\frac{p}{p-1}}$ und da die Folge x_n bzgl. L^p beschränkt ist, folgt auch $||P_3||_{L^{1,\frac{p}{p-1}}} \leq C$. Dann gilt

$$\begin{aligned} |f(x_n) - f(x)| \quad &= \quad |\int_0^T \dot{x}_n \cdot p(l + \dot{x}_n) - \dot{x} \cdot p(l + \dot{x}) dt| \\ &\leq \quad \int_0^T |(\dot{x}_n - \dot{x}) \cdot P_3(\dot{x}_n, \dot{x}, l)| dt \\ &\overset{\text{Hölder-Ungl.}}{\leq} \quad ||x_n - x||_{L^{1,p}} \cdot ||P_3(x_n, x, l)||_{L^{1,\frac{p}{p-1}}} \\ &\leq \quad C \cdot ||x_n - x||_{W^{1,p}}. \end{aligned} \tag{2.14}$$

Daraus folgt, dass wenn $x_n \to x$ konvergiert auch $f(x_n) \to f(x)$ gilt. Somit ist f ein stetiges Funktional. Um nun das Lemma 2.2.3 anwenden zu können, fehlt uns nur noch die Konvexität von f. Um diese zu zeigen benötigen wir jedoch ein weiteres Lemma:

Lemma 2.2.4. *Sei X ein Banachraum und f ein nicht-lineares Funktional auf X, dessen zweite Gâteaux Ableitung $D^2 f : X \times X \to \mathbb{R}$ existiert. Weiterhin gilt*

$$\langle D^2 f|_{x_0} u, u\rangle > 0 \qquad \forall x_0, u \in X, u \neq 0. \tag{2.15}$$

Dann ist f strikt konvex.

Beweis. Sei $\tilde{f} : [0, 1] \to \mathbb{R}$ mit

$$\tilde{f}(\lambda) \coloneqq f(\lambda x_0 + (1 - \lambda)x_0) \qquad x_0, x_1 \in X. \tag{2.16}$$

Dann gilt mithilfe der Kettenregel

$$\begin{aligned} \frac{\partial^2}{\partial\lambda^2}\tilde{f}(\lambda) &\coloneqq \langle D^2 f|_{\lambda x_0 + (1-\lambda)x_1} \cdot (x_0 - x_1), (x_0 - x_1)\rangle \\ &\overset{Vor.}{>} 0. \end{aligned} \tag{2.17}$$

Dies ist hinreichend für die Konvexität von $\tilde{f}$. Da $x_0, x_1 \in X$ beliebig gewählt wurden, gilt insbesondere

$$\begin{aligned} f(\lambda x_0 + (1-\lambda)x_1) &= \tilde{f}(\lambda \cdot 1 + (1-\lambda)\cdot 0) \\ &< \lambda \tilde{f}(1) + (1-\lambda)\tilde{f}(0) \\ &= \lambda f(x_0) + (1-\lambda) f(x_1). \end{aligned} \tag{2.18}$$

Somit ist auch f strikt konvex. □

Es bleibt also zu zeigen, dass die zweite Gâteaux Ableitung bzgl. f positiv definit ist. Es gilt

$$\begin{aligned} Df(E)v &= \lim_{\alpha \to 0} \frac{f(E+\alpha v) - f(E)}{\alpha} \\ &= \frac{\partial}{\partial \alpha} f(E+\alpha v)|_{\alpha=0} \\ &= \int_0^T (p(l+\dot{E}) + \dot{E} \cdot p'(l+\dot{E})) \cdot \dot{v}\, dt \\ &=: b_E(v) \qquad \forall v \in W_0^{1,p}. \end{aligned} \tag{2.19}$$

Weiterhin gilt

$$\begin{aligned} D^2 f(E)(v,u) &= \lim_{\alpha \to 0} \frac{Df(E+\alpha u)v - Df(E)v}{\alpha} \\ &= \frac{\partial}{\partial \alpha} Df(E+\alpha u)v|_{\alpha=0} \\ &= \int_0^T \dot{u} \cdot (2p'(l+\dot{E}) + \dot{E} \cdot p''(l+\dot{E})) \cdot \dot{v}\, dt \\ &=: a_E(u,v) \qquad \forall u, v \in W_0^{1,p}. \end{aligned} \tag{2.20}$$

Die positive Definitheit der zweiten Gâteaux Ableitung am Punkt E ist nun äquivalent zur Bedingung

$$a_E(v,v) > 0 \qquad \forall v \in W_0^{1,p}, v \neq 0. \tag{2.21}$$

Dies gilt, wenn fast überall

$$(2p'(l+\dot{E}) + \dot{E} \cdot p''(l+\dot{E})) \cdot \dot{v}^2 > 0 \tag{2.22}$$

gilt. Um dies bezogen auf den Sachverhalt zu zeigen, reicht es aus, die bildliche Darstellung der Funktion

$$g_l(E) := 2p'(l+\dot{E}) + \dot{E} \cdot p''(l+\dot{E})) > 0 \qquad \forall E \in W_0^{1,p}. \tag{2.23}$$

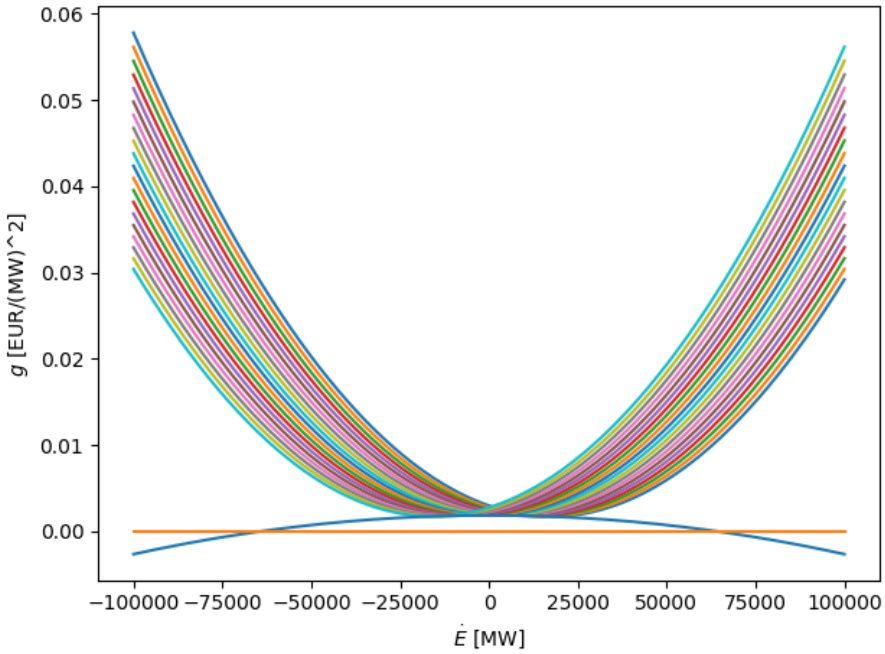


Abbildung 2.10: Funktionenschar $g_l(E)$ für verschiedene Werte der residualen Last l. Da alle Werte strikt größer als Null sind, folgt die strikte Konvexität des zu minimierenden Funktionals f

für die in den Daten auftretenden Werte zu untersuchen. In Abbildung 2.10 ist eine Schar von Funktionen $g_l(E)$ für verschiedene Werte der residualen Last, die in den gegebenen Daten vorkommen, dargestellt. Da die Werte für g_l alle über Null liegen, folgt dass das Funktional $f(E)$ für die zur Verfügung gestellten Daten tatsächlich konvex ist.

Um Satz 2.2.2 nun anwenden zu können, fehlt nur noch die Koerzitivitätsbedingung von f. Um diese zu zeigen, kann zunächst die folgende Hilfsfunktion definiert werden

$$\begin{aligned} h &: W_0^{1,p} \times L \to \mathbb{R} \\ h(x,y) &= \frac{x \cdot p(y+x)}{(1+x^2)^2}. \end{aligned} \tag{2.24}$$

Die Menge $L \subset \mathbb{R}$ beschreibt dabei die vorkommenden Werte der residualen Last und ist kompakt, da $l \in L^\infty$ angenommen wurde. Da p ein Polynom dritten Grades ist, folgt nun, dass jede Folge (x_n, y_n) mit $x_n \to \pm\infty$ gegen den Koeffizienten a_3 des Polynoms

p konvergiert. Zudem gilt laut (2.6) $a_3 > 0$. Dementsprechend ist auch die Menge $Z := \{(x,y) \in V \times L : h(x,y) \leq \frac{a_3}{2}\}$ kompakt und aufgrund von $h(0,\cdot) = 0$ nicht leer. Dies hat zur Folge, dass die Funktion

$$\begin{aligned} &k : W_0^{1,p} \times L \to \mathbb{R} \\ &k(x,y) = (h(x,y) - \frac{a_3}{2})(1+x^2)^2 \end{aligned} \tag{2.25}$$

auf der Menge Z ihr Minimum $C < 0$ annimmt. Aufgrund der Definition von Z gilt jedoch $k(x,y) \geq 0$ für $(x,y) \in (V \times L) \setminus Z$. Somit ist C auch das globale Minimum von k. Also gilt

$$\begin{aligned} k(x,y) &\geq C \\ \Leftrightarrow x \cdot p(y+x) &\geq \frac{a_3}{2}(1+x^2)^2 + C \\ &\geq \frac{a_3}{2}|x|^4 + C \\ \Rightarrow f(x) = \int_0^T \dot{x} \cdot p(y+\dot{x}) &\geq \frac{a_3}{2}\int_0^T |\dot{x}|^4 dt + \int_0^T C dt \\ &\geq C''||x||^4_{W_0^{1,4}} + C', \end{aligned} \tag{2.26}$$

wobei die letzte Ungleichung durch die Poincaré-Friedrichs-Ungleichung gilt, die aufgrund der Null-Randbedingungen benutzt werden kann. Nun sind alle Bedingungen von Satz 2.2.2 erfüllt, sodass das Minimierungsproblem (2.7) eine Lösung besitzt. Die Eindeutigkeit dieser Lösung folgt direkt aus der strikten Konvexität des Funktionals f. Denn angenommen, es gibt zwei Lösungen E_1 und E_2 mit $\min_{E \in V} f(E) = f(E_1) = f(E_2)$. Dann gilt:

$$\begin{aligned} f(\lambda E_1 + (1-\lambda)E_2) &< \lambda f(E_1) + (1-\lambda) f(E_2) \\ &= \min_{E \in V} f(E), \qquad \lambda \in [0,1] \end{aligned} \tag{2.27}$$

Dies ist jedoch ein Widerspruch zur Annahme, dass E_1 und E_2 das Funktional f minimieren.

2.2.2 Diskretisierung des Lösungsraumes

Um die Lösung nun numerisch berechnen zu können, muss das Problem zunächst in einen endlich dimensionalen Raum diskretisiert werden. Sei dazu

$$V^h := \{v \in \mathcal{C}([0,T]) : v \text{ linear in } [t,t+1]\ ; \forall t \in \{0,...,T\}; v(0) = v(T) = 0\} \tag{2.28}$$

Nun kann die diskrete Lösung E^h als Basisentwicklung mit Koeffizienten E_t wie folgt dargestellt werden

$$E^h = \sum_{t=0}^{T} E_t \Phi_t. \tag{2.29}$$

Dabei sind Φ_t die Basisfunktionen (Hutfunktionen) von V^h. Somit gilt auch:

$$\dot{E}^h|_{t\in[t,t+1]} = E_{t+1} - E_t. \tag{2.30}$$

Damit lässt sich das Minimierungsproblem (2.7) wie folgt diskretisieren:

$$\begin{aligned} &\min_{x\in V^h} f^h(x), \qquad \text{wobei} \\ f^h(x) &:= x_0 \cdot p(l_0 + x_0) + \sum_{t=1}^{T-1} (x_t - x_{t-1}) \cdot p(l_t + x_t - x_{t-1}). \end{aligned} \tag{2.31}$$

Man kann die Nebenbedingungen auch umschreiben, um diese in eine standardisierte Form zu bringen:

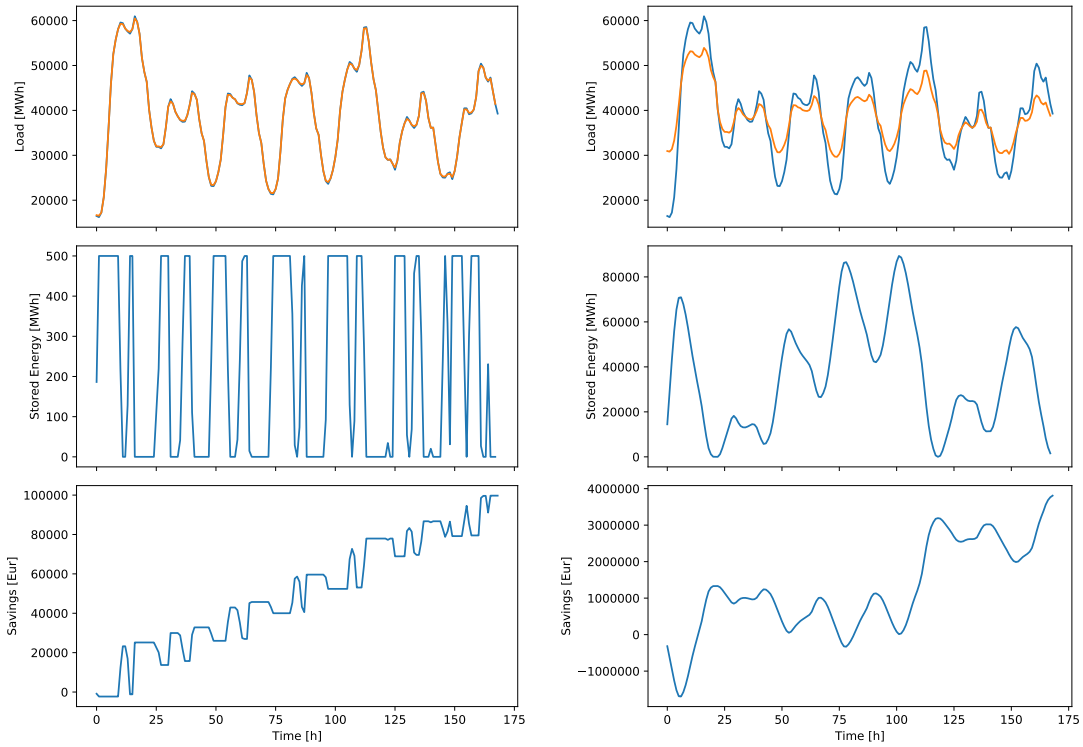
$$\begin{aligned} x_t &\geq 0 \\ E_{max} - x_t &\geq 0 \end{aligned} \tag{2.32}$$

Da V^h konform bzgl. $W_0^{1,p}$ ist, d.h. $V^h \subset W_0^{1,p}$, hat auch das Minimierungsproblem bezüglich f^h eine eindeutige Lösung in V^h. Somit kann das Problem nun mithilfe der Inneren-Punkte-Methode gelöst werden. Dabei wird die Zielfunktion $f^h(x)$ um einen Barriere Term erweitert:

$$\widetilde{f^h}(x) = f^h(x) - \mu \cdot \sum_{i=0}^{m} \ln(c_i(x)) \tag{2.33}$$

wobei $c_i(x)$ die Nebenbedingungen aus (2.32) sind. Dies führt dazu, dass $\widetilde{f^h}(x)$ für x Werte, die diese Nebenbedingungen verletzen, bestraft wird. Nun kann das Minimum der Funktion mithilfe des Newton-Verfahrens berechnet werden, wobei der Parameter μ nach jeder Newtoniteration verringert wird. Für weitere technische Details und Konvergenzanalysen wird auf [4, Kapitel 11] verwiesen. Das Programm zur Berechnung der Lösung basiert auf einem DLR internen Code, der für das Energiespeicherproblem angepasst wurde.

In Abbildung 2.11(a) ist die Lösung der Inneren-Punkte-Methode für eine Speicherkapazität von 500MWh über 7 Tage dargestellt. Im oberen Plot ist sowohl die wahre residuale Last als auch die durch das Verhalten des Speichers resultierende Last eingezeichnet. Wie man sieht, hat ein einzelner 500MWh Speicher keinen großen Einfluss auf den Verlauf der residualen Last. An dem Verlauf des Ladezustands im mittleren Bild lässt sich zudem erkennen, dass der Speicher an den lokalen Extremstellen der residualen Last innerhalb von 1-2 Stunden voll geladen, bzw. entleert wird. Erhöht man die Kapazität des Speichers auf $5 \cdot 10^9$MWh, so ist dessen Einfluss auf die residuale Last deutlich größer, wie man in Abbildung 2.11(b) sieht. Zudem wird der Verlauf der gespeicherten Energie fließender, da es sich lohnt die eingekaufte und verkaufte Energie auf einen größeren Zeitabschnitt zu verteilen, um den eigenen Einfluss auf die residuale Last zu minimieren. Im unteren Plot ist weiterhin der Gewinn der Strategie dargestellt. Bemerkenswerterweise unterscheidet sich der Gewinn dieser beiden Lösungen lediglich um einen Faktor 30, obwohl die Kapazität um einen Faktor 100 vergrößert wurde. Dies liegt wie oben beschrieben an der schwindenden Gewinnspanne, relativ zum gehandelten Energievolumen. In Abbildung 2.12 ist das Verhältnis zwischen der Kapazität des Speichers und dem damit resultierenden Gewinn dargestellt. Wie man sieht, tritt schon bei einer Kapazität von 50000 MWh ein Marktsättigungseffekt ein.



(a) $K = 500MWh$ (b) $K = 5 \cdot 10^9 MWh$

Abbildung 2.11: Optimale Lösung der Inneren-Punkte-Methode mit unterschiedlichen maximalen Speicherkapazitäten über sieben Tage. Oben ist die ursprüngliche residuale Last, sowie die durch den Speicher resultierende Last dargestellt. In der Mitte ist der Ladezustand des Speichers zu jeder Stunde t zu sehen. Unten ist der Gewinn dargestellt, den der Speicher über die Zeit erzielt.

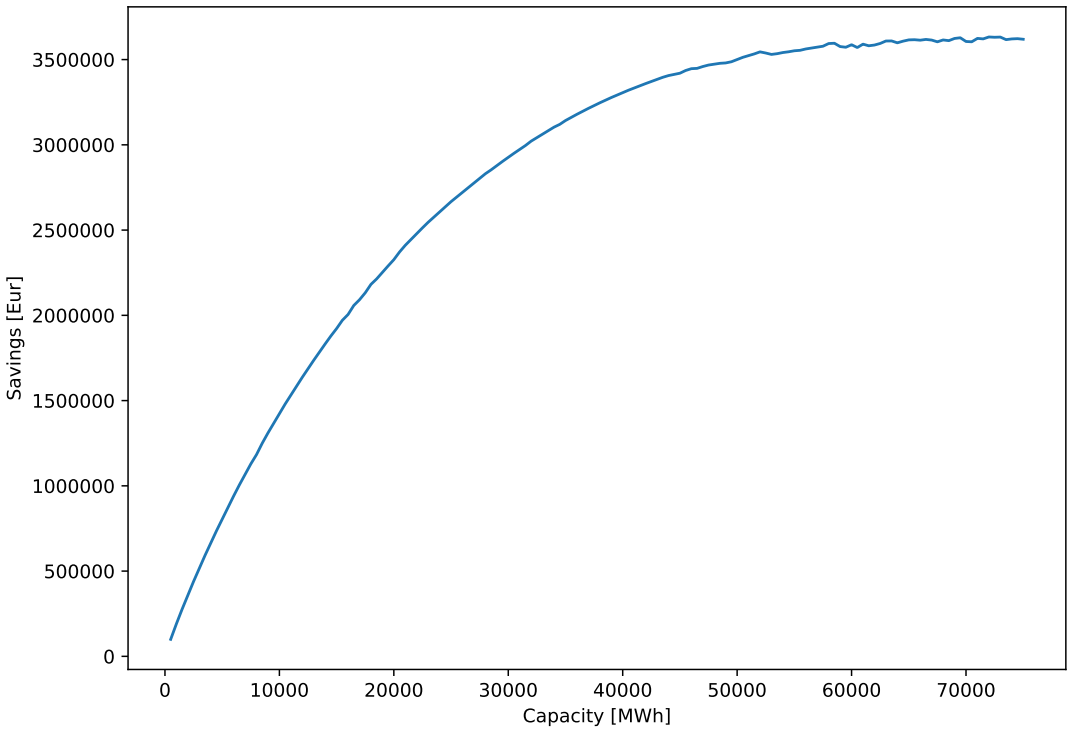


Abbildung 2.12: Verhältnis zwischen Kapazität und Gewinn

2.2.3 Markov Entscheidungsprozess der Energiespeicheragenten

Um das Problem nun mit einem Reinforcement Learning Ansatz zu lösen, muss zunächst das MDP definiert werden:

Ein Agent soll in der Lage sein Energie zu kaufen und zu verkaufen, sodass man den Aktionsraum durch drei diskrete Aktionen definieren kann:

$$\mathcal{A} := \{\text{verkaufen, nichts tun, kaufen}\} \tag{2.34}$$

Die Belohnungen, die der Agent für eine Aktion erhält, wird dabei wie folgt definiert:

$$r_{t+1} = \begin{cases} E_t \cdot p_t & \text{falls } a_t = \text{verkaufen}, \\ 0 & \text{falls } a_t = \text{nichts tun}, \\ -E_t \cdot p_t & \text{falls } a_t = \text{kaufen}, \end{cases} \tag{2.35}$$

wobei p_t der Marktpreis zur Stunde t ist und E_t die Menge der gehandelten Energie in MWh. Die Zustände, in denen sich der Agent befinden kann, sind jedoch etwas komplizierter. Zum einen muss ein Zustand vom vorherigen Zustand und der zuletzt ausgeführten Aktion des Agenten abhängen, damit die Beziehung

$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + V(s_{t+1}) | S_t = s_t, A_t = a_t] \tag{2.36}$$

ausgenutzt werden kann. Dies kann erreicht werden, indem der aktuelle Ladezustand des Speichers als Zustand gewählt wird. Dies reicht jedoch noch nicht aus, da ein Zustand auch genug Informationen beinhalten sollte, damit der Agent eine optimale Aktion in diesem Zustand ausführen kann. Damit der Agent weiß, ob er zu einer Stunde t seinen Speicher laden oder entladen soll, sollte er wissen, ob der Marktpreis demnächst steigt oder sinkt. Da der Marktpreis insbesondere von der residualen Last abhängt, sollte diese auch in den Zustand miteinfließen. Doch selbst wenn der Agent die aktuelle residuale Last als Zustand erhält, reicht ihm diese nicht aus um zu wissen, ob der Preis zur nächsten Stunde fällt oder steigt, sodass das MDP die Markov Eigenschaft nicht erfüllen würde. Das gleiche Problem tritt auch beim Lernen von Atari Spielen auf, bei dem ein einzelnes Bild nicht ausreicht um zu wissen ob ein Ball hoch oder runterfliegt. Dieses Problem wurde gelöst, indem stattdessen vier aufeinanderfolgende Bilder als ein Zustand gewählt wurden. Diese Technik kann auf das Speicherproblem übertragen werden, indem ein Agent nicht nur die aktuelle residuale Last sondern auch die der vorherigen oder darauffolgenden Stunden als Zustand erhält. In der Realität ist die zukünftige residuale Last zwar nicht bekannt, jedoch verfügen Akteure des Energiemarktes in der Regel über Daten, mit denen sie die residuale Last der Zukunft vorhersagen können. Außerdem ist zu beachten, dass es in der Regel nicht üblich ist, Informationen aus der Zukunft in den aktuellen Zustand zusammenzufassen, da ein Agent in der Lage ist die Zukunft mit

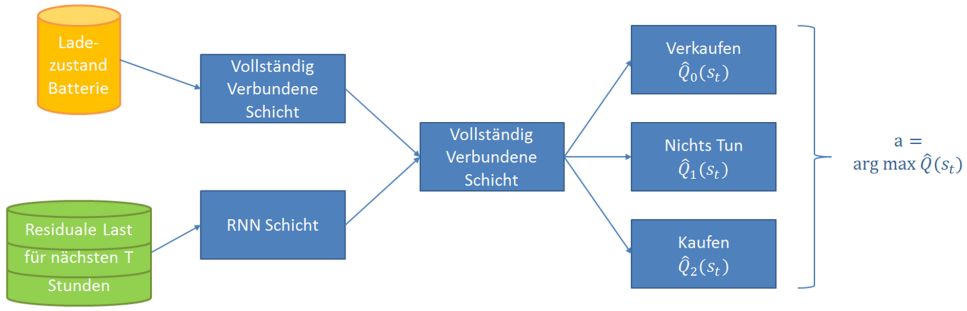


Abbildung 2.13: Deep Q-Network eines Speicheragenten

seinen eigenen Aktionen zu beeinflussen. Dies ist hier allerdings nicht der Fall, da der Zustand der residualen Last der Zukunft unabhängig von den Aktionen der Agenten ist. Somit können wir den Zustandsraum wie folgt definieren:

$$\mathcal{S} := B \times RL \tag{2.37}$$

Dabei stellt $B = \{0, 1\}$ den Ladezustand des Speichers dar und $RL = [0MW, 100000MW]^h$ die residuale Last in einem Bereich von h Stunden. Dabei kann sowohl die zukünftige als auch die vergangene residuale Last als Zustand gewählt werden.

Um das originale Q-Learning Verfahren anwenden zu können, müsste man den Zustandsraum zunächst diskretisieren. Allerdings würde der Zustandsraum dadurch sehr groß werden, zumal die Kardinalität der Diskretisierung exponentiell steigen würde je größer die Voraussicht h des Agenten gewählt wird. Somit bietet es sich an, die Aktionswertefunktion $Q(s_t, a_t)$ mithilfe eines Deep Q-Networks zu approximieren. Dies hat vor allem den Vorteil, dass die Daten der residualen Last durch ein rekurrentes neuronales Netzwerk verarbeitet werden können. In Abbildung 2.13 ist der Aufbau des Deep Q-Networks für einen Speicheragenten dargestellt. Der Ladezustand des Speichers und die residuale Last fließen zunächst getrennt in das Netzwerk ein und werden anschließend durch eine vollständig verbundene Schicht zusammengeführt. Anschließend wird für jede Aktion ein Output generiert, die den Aktionswerten in Abhängigkeit des Zustands entsprechen sollen.

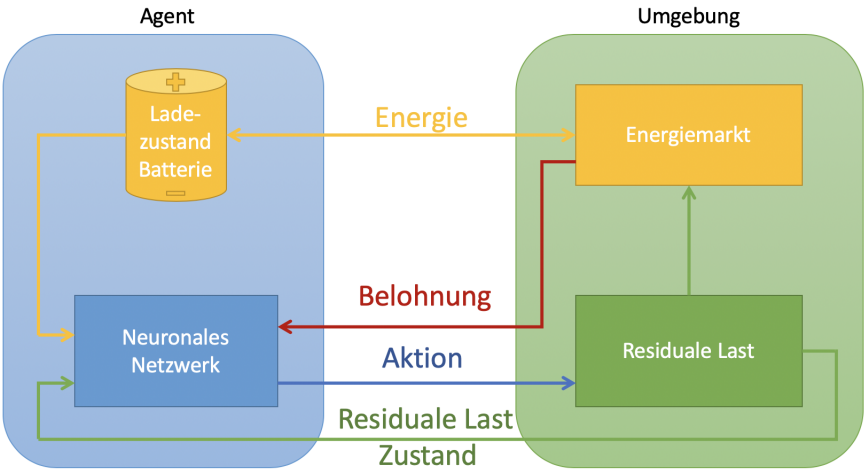
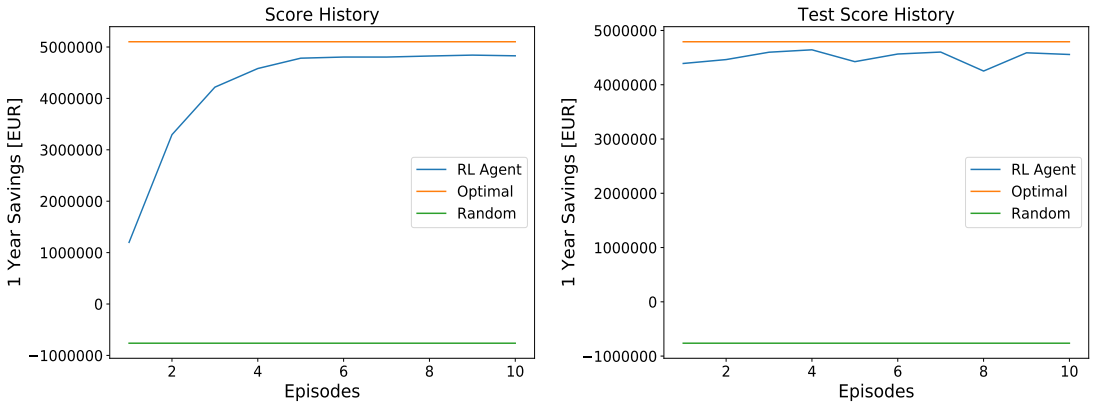


Abbildung 2.14: Wechselwirkung zwischen Agent und seiner Umgebung des Energiespeicher Modells

Zur Implementierung des neuronalen Netzwerks wurde das Python Paket „Keras“ [8] genutzt. Die Gewichte des Netzwerks werden dabei zunächst durch eine uniforme Wahrscheinlichkeitsverteilung initialisiert und werden während der Simulation des Modells durch die Verlustfunktion (1.39) optimiert. Dazu wurde der Optimierungsalgorithmus „Adam“ (Adaptive Moment Estimation) [21] mit einer Lernrate $\alpha = 0.001$ genutzt. Die hidden States des rekurrenten neuronalen Netzwerks werden dabei mit 0 initialisiert und werden nach jeder Vorhersage und jedem Optimierungsschritt zurückgesetzt. Um das Problem möglichst stationär zu halten, entspricht eine Episode einem simulierten Jahr, sodass die saisonale Varianz mitberücksichtigt werden kann. Da ein Agent jede Stunde seinen Speicher laden bzw. entladen kann, besteht eine Episode somit aus $24 \cdot 365 = 8760$ Schritten. In Abbildung 2.14 ist die gesamte Wechselwirkung zwischen dem Agenten und seiner Umgebung dargestellt. Der Energiemarkt liefert, bzw. kauft dabei die Elektrizität des Agenten und berechnet durch (2.6) die Belohnung, die der Agent für die ausgeführte Aktion erhält, womit sein Netzwerk anschließend trainiert wird.

Damit ein Agent genug Zeit hat um seine Strategie zu optimieren besteht das Training

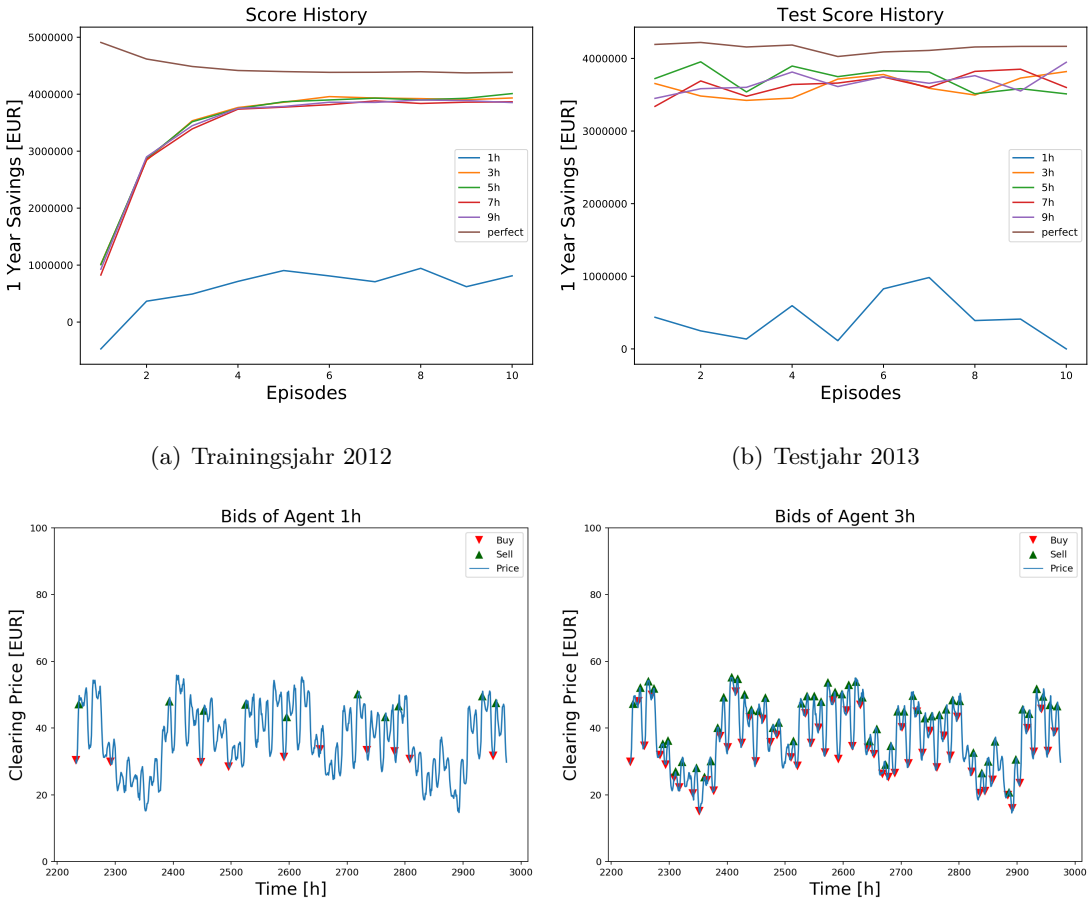


(a) Trainingsjahr 2012

(b) Testjahr 2013

Abbildung 2.15: Gewinn eines Reinforcement Learning Agenten im Vergleich zur optimalen Lösung

aus zehn Episoden, indem das Jahr 2012 simuliert wird. Um zu gewährleisten, dass der Agent eine generalisierbare Strategie gelernt hat und nicht zu sehr an den Trainigsdaten angepasst ist, werden die Gewichte des Netzwerks nach jeder Episode gespeichert und anschließend mit den Daten des Jahres 2013 getestet. In Abbildung 2.15 ist der Gewinn eines Reinforcement Learning Agenten mit einer Speicherkapazität von 500MWh im Vergleich zur optimalen Lösung des Inneren-Punkte-Verfahrens dargestellt. Der Agent erhält dabei zu jeder Stunde t sowohl die residuale Last der letzten fünf Stunden als auch der zukünftigen fünf Stunden als Zustand s_t. Wie man sieht, kommt der Reinforcement Learning Agent relativ nah an die optimale Lösung und liegt deutlich über einer vollständig zufälligen Strategie. Außerdem fällt auf, dass der Agent nach der ersten Episode einen deutlich höheren Gewinn auf dem Testjahr als auf dem Trainigsjahr erzielt hat. Dies ist auf die ε-Greedy Erkundungsstrategie zurückzuführen, da der Agent beim Training mit einer abklingenden Wahrscheinlichkeit eine zufällige Aktion ausführt, während beim Testen ε direkt auf null gesetzt wird. Es ist jedoch zu beachten, dass der Reinforcement Learning Agent niemals den Gewinn der optimalen Lösung erreichen kann, da er aufgrund des diskreten Zustandsraumes zu jeder Stunde die gesamte Speicherkapazität kaufen oder verkaufen muss. Die optimale Lösung ist hingegen in der Lage mit beliebigen Energiemen-



(a) Trainingsjahr 2012

(b) Testjahr 2013

(c) Handelsverhalten 1 Stunde Voraussicht

(d) Handelsverhalten 3 Stunden Voraussicht

Abbildung 2.16: Ergebnisse des DQN Algorithmus mit unterschiedlicher Voraussicht

gen zu handeln, solange die Speicherkapazität nicht überschritten wird. Zudem liegt die Stärke des Reinforcement Learning Ansatzes darin, in unsicheren Umgebungen angewandt werden zu können. Dies ist nämlich der Fall, wenn sich mehr als nur ein Agent an der Simulation beteiligt. Der Marktpreis hängt dann nämlich nicht nur von der residualen Last ab, sondern auch von den Entscheidungen der übrigen Agenten. In Abbildung 2.16(a) und 2.16(b) ist der Gewinn auf dem Trainings- und Testjahr dargestellt, wenn sowohl die optimale Lösung als auch fünf weitere Reinforcement Learning Agenten sich gleichzeitig am Markt beteiligen. Hierbei fällt besonders der sinkende Gewinn der optimalen Lösung

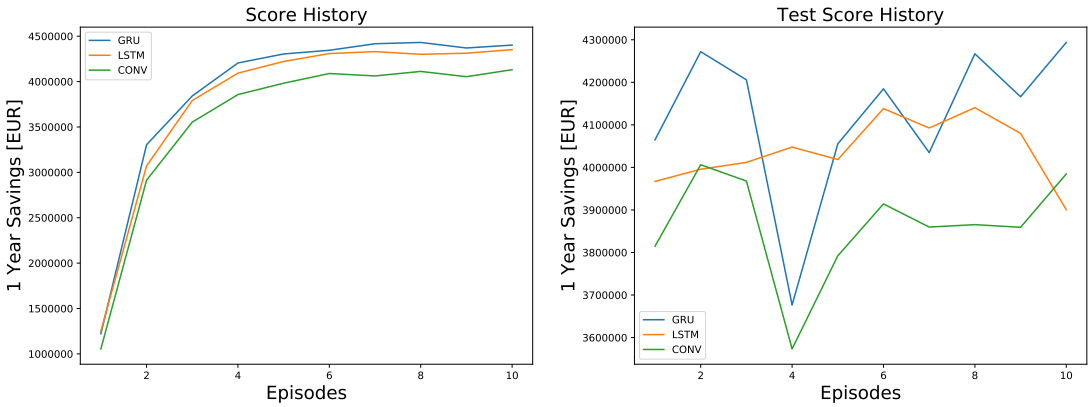
auf, je besser die Entscheidungen der Reinforcement Learning Agenten werden. Auch der Vorsprung der Lösung die mit dem Optimierungsverfahren berechnet wurde sinkt gegenüber den RL Agenten um ca. 100.000€ im Vergleich zur getrennten Auswertung aus Abbildung 2.15. Dies lässt vermuten, dass bei einer Simulation mit noch mehr Agenten die optimale Lösung sogar überholt werden könnte.

Gleichzeitig wurden in Abbildung 2.16 Agenten mit unterschiedlicher Voraussicht verglichen. Es ist zu erkennen, dass es keinen signifikanten Unterschied zwischen den Agenten mit mehr als einer Stunde Voraussicht gibt. Wenn ein Agent jedoch nur die residualen Last der aktuellen Stunde t als Zustand erhält, so schneidet er deutlich schlechter ab, wie oben bereits angekündigt. An dem Kaufverhalten dieses Agenten in Abbildung 2.16(c) ist zu erkennen, dass er sich jeweils ein konstantes Preisniveau zum Kaufen und Verkaufen aussucht, während ein Agent mit drei-stündiger Voraussicht in Abbildung 2.16(d) in der Lage ist, die meisten lokalen Extremstellen zu erkennen und auszunutzen. Obwohl es keinen signifikanten Unterschied zwischen den restlichen Einstellungen gibt, wurde für den weiteren Verlauf der Arbeit eine Voraussicht von drei Stunden gewählt, um Rechenzeit zu sparen.

2.2.4 Ergebnisse der Reinforcement Learning Agenten

Aufgrund der Verbindung von Reinforcement Learning, neuronalen Netzwerken und der Modellierung des elektrischen Marktes hängen die Ergebnisse jedoch noch von vielen weiteren Parametern ab. Diese lassen sich in drei unterschiedliche Kategorien aufteilen. Zum einen gibt es die Hyperparameter des Neuronalen Netzwerks wie der Art der Schichten und dessen Anzahl an Neuronen. Des Weiteren können auch die Parameter des MDPs variiert werden, z.B. der Diskontierungsfaktor γ oder die Größe des Zustandsraumes. Den größten Einfluss auf die Ergebnisse haben jedoch die physikalischen Eigenschaften des Energiespeichers wie die verfügbare Speicherkapazität oder auch dessen Wirkungsgrad.

Zunächst wurden die Hyperparameter des Neuronalen Netzwerks untersucht. Es hat sich

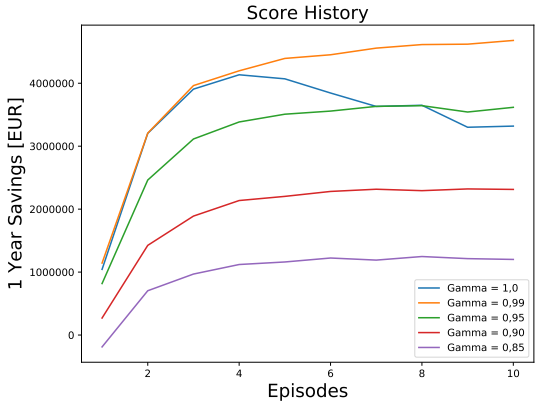


(a) Trainingsjahr 2012

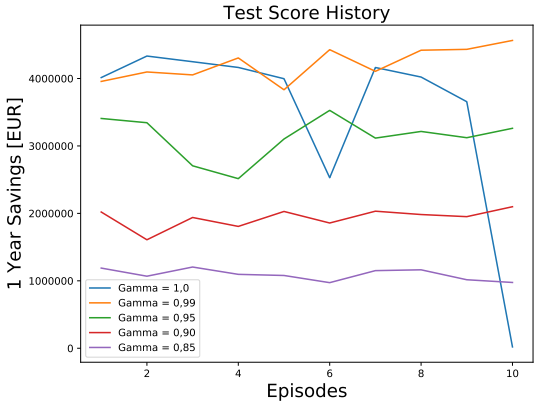
(b) Testjahr 2013

Abbildung 2.17: Vergleich des Gewinns zwischen verschiedenen Netzwerkarchitekturen

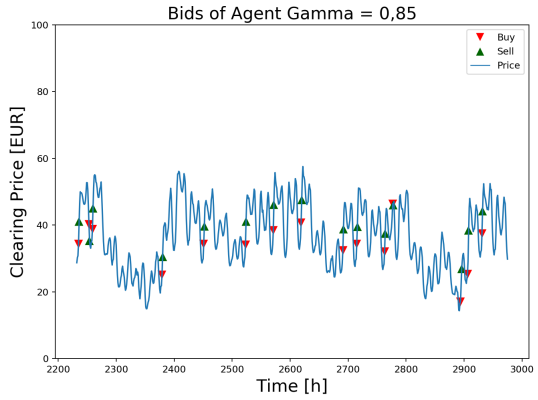
jedoch ergeben, dass es kaum Unterschiede bzgl. des Gewinns der Agenten zwischen den Einstellungen gab. Selbst wenn man unterschiedliche Arten an Netzwerkschichten zur Verarbeitung der zeitlichen Daten wählt, ist kein besonders großer Unterschied zwischen den gelernten Strategien zu erkennen. In Abbildung 2.17 ist der Vergleich zwischen den in Kapitel 1.5 vorgestellten Netzwerkschichten dargestellt. Es ist zu erkennen, dass ein Netzwerk mit einer LSTM bzw. GRU Schicht sowohl für das Trainings- als auch das Testjahr einen leichten Vorsprung gegenüber eines Netzwerks mit einer Convolutional Schicht hat. Da Convolutional Netzwerke insbesondere zur Verarbeitung von Bilddaten entwickelt wurden, ist es nicht verwunderlich, dass hier rekurrente Netzwerke besser funktionieren. Es ist jedoch nicht ganz klar, ob man nun eine GRU oder LSTM Schicht wählen sollte, da sie sich im Gewinn des Trainingsjahres kaum unterscheiden. An den Ergebnissen des Testjahres ist zu erkennen, dass die GRU Schicht meistens einen höheren Gewinn erzielt, jedoch weniger stabil gegenüber einer LSTM Schicht ist. Da eine GRU Schicht jedoch bezüglich der Laufzeit effizienter ist, wurde bei allen weiteren Ergebnissen eine GRU Schicht benutzt. Es ist jedoch zu beachten, dass das vollständige Potential der rekurrenten neuronalen Netzwerke nicht ausgenutzt werden kann. Die wahre Stärke dieser Netzwerkarchitekturen liegt im hidden state, in dem die Informationen der vergangenen Zustände gespeichert werden. Dies erfordert jedoch ein sequentielles Training des



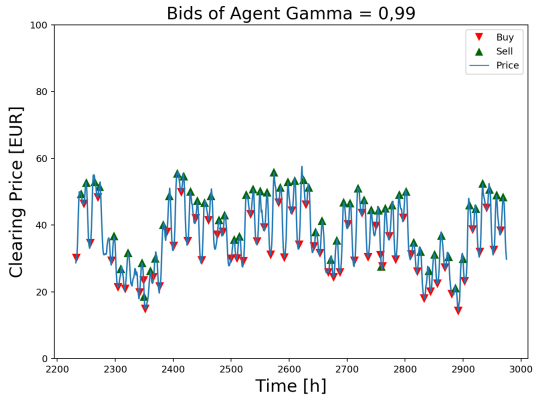
(a) Trainingsjahr 2012



(b) Testjahr 2013



(c) Handelsverhalten $\gamma = 0.85$

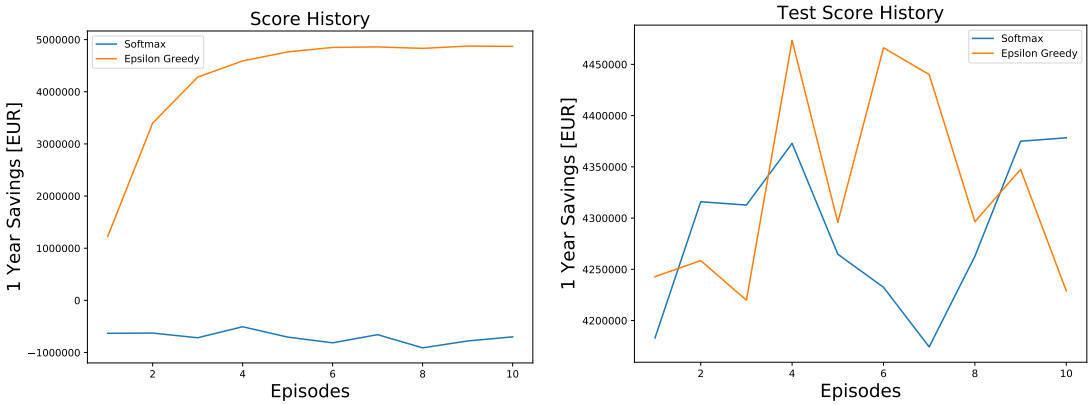


(d) Handelsverhalten $\gamma = 0.99$

Abbildung 2.18: Vergleich der Agenten mit unterschiedlichem Diskontierungsfaktor γ

Netzwerks. Dies ist im Deep Q-Network Algorithmus nicht der Fall ist, da das Netzwerk mit zufälligen Stichproben des Erfahrungsvektors trainiert wurde. Somit muss der hidden state nach jeder Beobachtung auf den Initialwert zurückgesetzt werden und spielt lediglich innerhalb eines Zustands eine Rolle.

Einen deutlich größeren Einfluss haben jedoch die Parameter des MDPs. Die Größe des Zustandsraumes wurde oben bereits durch den Vergleich der Voraussicht der residualen Last untersucht. Ein weiterer Parameter, der für die Voraussicht der Agenten zuständig



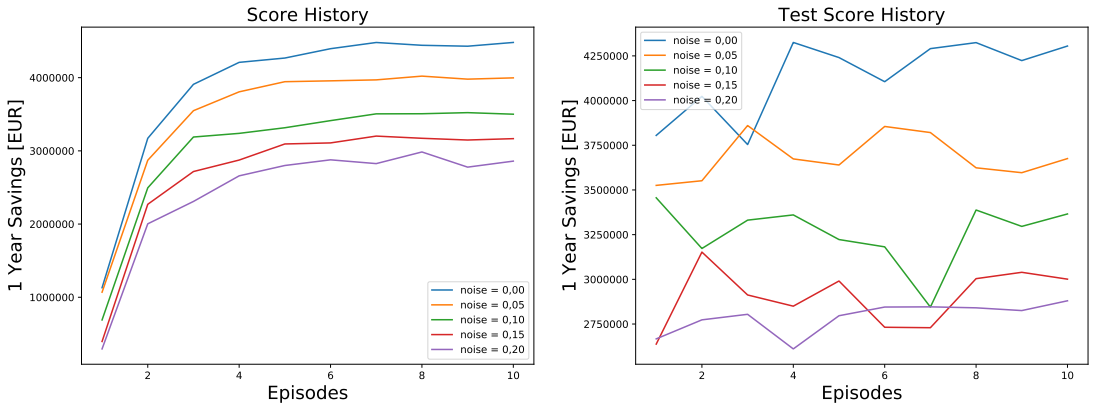
(a) Traingsjahr 2012

(b) Testjahr 2013

Abbildung 2.19: Vergleich zwischen Erkundungsstrategien

ist, ist der Diskontierungsfaktor γ. Dieser wurde in Abbildung 2.18 verglichen. Wie man sieht, hat eine leichte Verringerung von γ eine negative Wirkung auf den Gewinn des Agenten. Der Grund dafür lässt sich an dem Handelsverhalten des Agenten in Abbildung 2.18(c) erkennen. Für $\gamma = 0.85$ wird der Agent gierig und verkauft seine Energie früher als er es sollte. Gleichzeitig kauft er auch zu spät ein, was daran liegt, dass er den Verlust, den er beim Einkaufen macht, so schnell wie möglich wieder reinholen muss, da der Gewinn sonst zu weit in der Zukunft liegen würde und somit zu stark diskontiert wird. Für $\gamma = 1$ erreicht der Agent einen ähnlich hohen Gewinn wie für $\gamma = 0.99$, fällt jedoch nach einigen Trainingsepisoden wieder ab. Dieser Effekt liegt daran, dass die Q-Werte des MDPs unbeschränkt sind und vermutlich nach einiger Zeit explodieren. Somit ist $\gamma = 0.99$ die beste Wahl für dieses MDP.

Weiterhin wurde untersucht, wie sich die Erkundungsstrategie der Agenten auf deren Gewinn auswirkt. Der Vergleich zwischen der ε-Greedy mit Zerfallsparameter $\delta_\varepsilon = 0.9999$ und der Softmax-Aktionswahl mit Temperaturparameter $\tau = 1$ ist in Abbildung 2.19 dargestellt. Es fällt auf, dass der Agent mit Softmax-Aktionswahl während der Trainingsphase nicht in der Lage ist, einer guten Strategie zu folgen, während der ε-Greedy Agent sich nach jeder Episode verbessert. Dies liegt daran, dass sich die Q-Werte bzgl. der



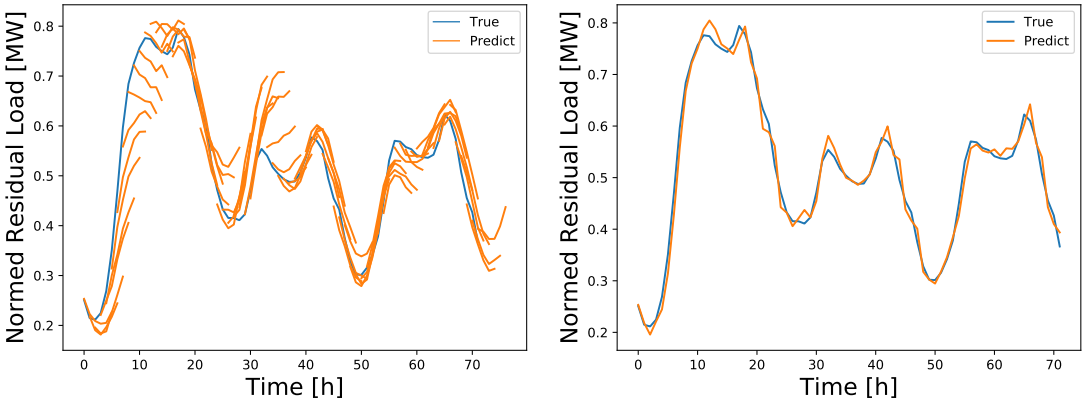
(a) Traingsjahr 2012

(b) Testjahr 2013

Abbildung 2.20: Vergleich zwischen der Genauigkeit der Vorhersagen

unterschiedlichen Aktionen zu wenig unterscheiden, sodass die Softmax Funktion eine uniforme Wahrscheinlichkeitsverteilung generiert. Somit verhält sich dessen Strategie wie eine vollständig zufällige Strategie. Durch das Reduzieren des Temperaturparameters τ ist es möglich kleine Unterschiede der Q-Werte stärker zu gewichten, jedoch ist nicht klar wie groß dieser gewählt werden sollte um zu gewährleisten, dass der Agent vor allem am Anfang des Trainings auch genügend erkundet. Doch wenn man sich die Ergebnisse der Testphase anschaut, in der die Agenten zu jeder Stunde die Aktionen mit den höchsten Q-Werten ausführen, stellt man fest, dass beide Erkundungsstrategien einen ähnlich hohen Gewinn erzielen, obwohl der Agent mit Softmax-Aktionswahl in der Trainingsphase nur Verluste gemacht hat.

Die oben beschriebenen Ergebnisse repräsentieren jedoch noch keinen realistischen Agenten, da deren gelernte Strategien auf der residualen Last der Zukunft basieren, die in der Realität nicht bekannt ist. Dennoch verfügen realistische Agenten zumindest Vorhersagen der residualen Last, worauf sie ihre Entscheidungen stützen. In dem Projekt „Assessing the Plurality of Actors and Policy Interactions: Agent-Based Modelling of Renewable Energy Market Integration“ [9] des Instituts für Technische Thermodynamik



(a) 6 vorhergesagte Stunden

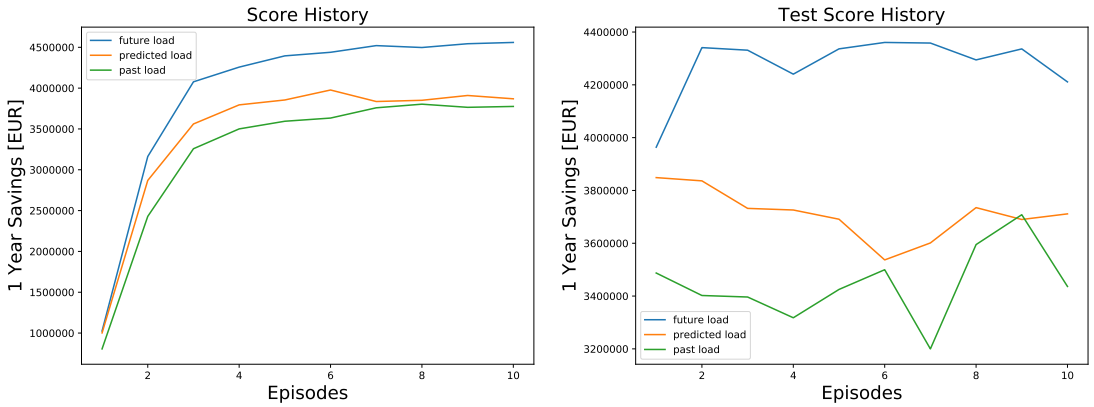
(b) 1 vorhergesagte Stunde

Abbildung 2.21: Vorhersage eines neuronalen Netzwerks der residualen Last

des Deutschen Zentrum für Luft- und Raumfahrt wurde die Voraussicht eines Agenten durch verrauschte exakte Daten modelliert. Diese Methodik kann auf das Energiespeicher Modell übertragen werden, indem der Zustand der aktuellen und zukünftigen residualen Last durch einen Fehler $\varepsilon \sim \mathcal{N}(0, \sigma \cdot \max l_t)$ modifiziert wird:

$$\hat{l}_t := l_t + \varepsilon \tag{2.38}$$

Dabei ist l_t die wahre residuale Last zur Stunde t und σ ein Parameter, der die Größenordnung des Fehlers bestimmt. Die Werte der vergangenen residualen Last, die der Agent erhält, werden jedoch nicht verändert, da diese bereits bekannt sind. In Abbildung 2.20 sind die Ergebnisse für unterschiedliche σ dargestellt. Wie zu erwarten war, sinkt der Gewinn der Agenten ab, je größer dessen Vorhersagefehler ist. Noch interessanter wäre es jedoch zu untersuchen, wie ein Agent mit tatsächlich vorhergesagten Werten umgeht. Diese können mithilfe eines supervised Machine Learning Ansatzes generiert werden. Auch für diese Aufgabe sind rekurrente neuronale Netzwerke besonders gut geeignet, da man eine zeitlich zusammenhängende Reihe von Werten vorhersagen möchte. In Abbildung 2.21 ist die 6-stündige Vorhersage eines einfachen neuronalen Netzwerks mit zwei GRU Schichten dargestellt. Als Input erhält das Netzwerk die residuale Last



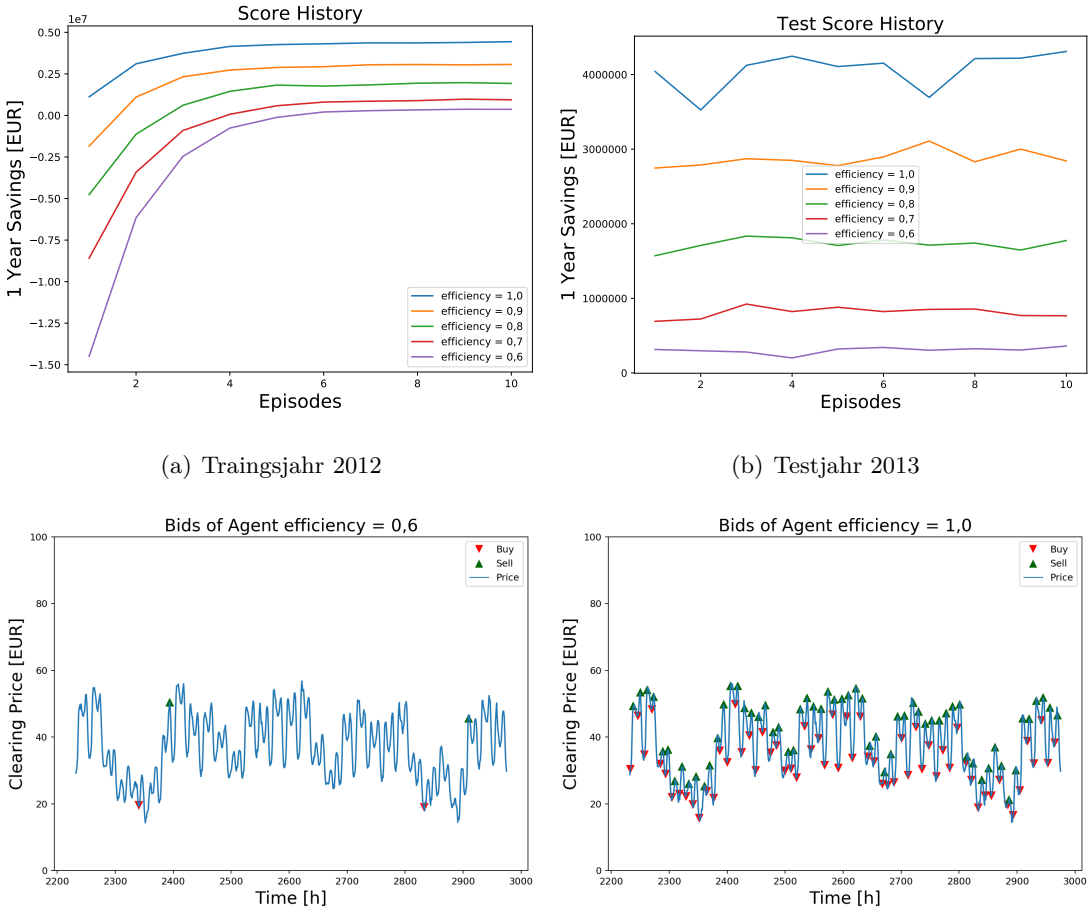
(a) Traingsjahr 2012

(b) Testjahr 2013

Abbildung 2.22: Vergleich zwischen perfekter Vorausicht und vorhergesagter Voraussicht

der vergangenen 24 Stunden und generiert als Output die der darauffolgenden 6 Stunden. Das Netzwerk wurde mithilfe des Adam Optimierers über 50 Epochen trainiert, wobei die Jahre 2012-2013 in jeweils einen Trainings- und Testdatensatz zufällig aufgeteilt wurden. Nun kann der Output dieses Netzwerks als Zustand der Reinforcement Learning Agenten gewählt werden. Die Ergebnisse eines solchen Agenten sind in Abbildung 2.22 zu sehen. Gleichzeitig wurde dieser mit perfekter Voraussicht und mit der bereits bekannten vergangenen Last verglichen. Wie zu erwarten war, liegt der Gewinn des Agenten mit vorhergesagten Daten zwischen den beiden Alternativen, mit denen er verglichen wurde. Insbesondere ist an dem Testjahr zu erkennen, dass der Agent mit vorhergesagten Daten deutlich schneller gegen eine gute Strategie konvergiert, als der Agent mit nur den vergangenen Daten. Dafür besteht jedoch auch eine größere Gefahr, dass sich dieser zu sehr an den Trainingsdaten anpasst. Es ist jedoch zu beachten, dass dies keineswegs eine optimale Vorhersage der residualen Last ist, sodass die Performanz dieses Agenten mit einem besseren Vorhersagemodell noch weiter gesteigert werden kann.

Bei den bisherigen Ergebnissen wurde allerdings eine wichtige Eigenschaft eines Energiespeichers vernachlässigt, nämlich der Energieverlust. Um diesen zu modellieren, muss ein Agent, der seinen Speicher um eine Energiemenge K aufladen will, stattdessen $K' = K \cdot \frac{1}{wg}$



(a) Traingsjahr 2012

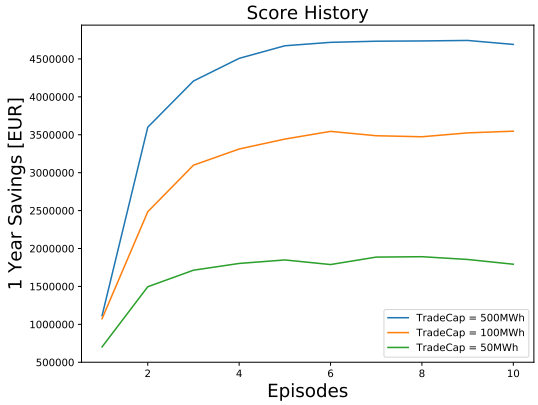
(b) Testjahr 2013

(c) Handelsverhalten $wg = 0.6$

(d) Handelsverhalten $wg = 1.0$

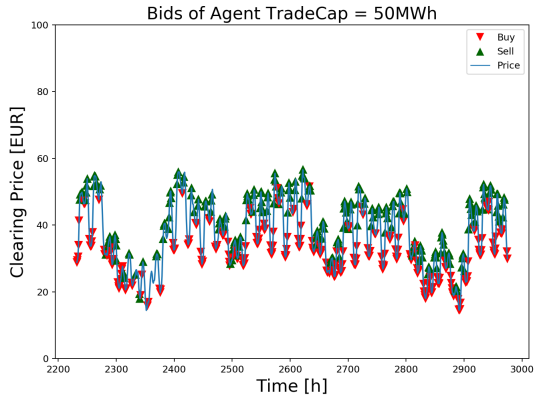
Abbildung 2.23: Vergleich der Agenten mit unterschiedlichen Wirkungsgrad

einkaufen. Dabei ist $wg \in [0, 1]$ der Wirkungsgradparameter, der beschreibt wie hoch der Energieverlust des Speichers ist. Offensichtlich sinkt der Gewinn des Agenten, je niedriger der Wirkungsgradparameter ist. Diese Tatsache wird durch Abbildung 2.23 bestätigt. Insbesondere an dem Handelsverhalten des Agenten mit $wg = 0.6$ kann man erkennen, dass der Gewinnverlust bzgl. eines perfekten Speichers nicht nur an dem Energieverlust selbst liegt, sondern auch daran, dass der Agent nicht mehr alle lokalen Extremstellen ausnutzen kann, da die Gewinnspanne nicht immer groß genug ist. Außerdem ist zu beachten, dass durch die Einführung von Energieverlust die gesamte Jahreslast erhöht

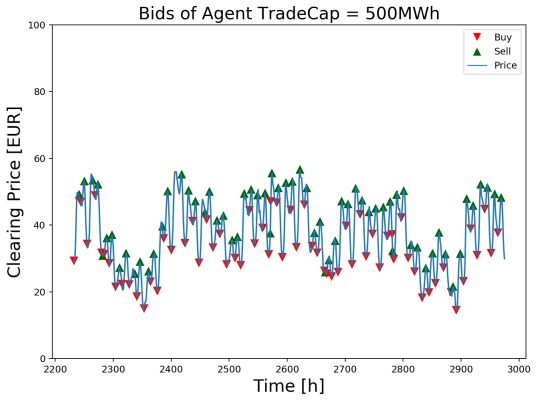


(a) Traingsjahr 2012

(b) Testjahr 2013



(c) Handelskapazität = $50MWh$



(d) Handelsverhalten = $500MWh$

Abbildung 2.24: Vergleich der Agenten mit unterschiedlichen Handelskapazitäten

werden muss um diesen Verlust auszugleichen.

Bisher waren die Agenten in der Lage ihren gesamten Speicher innerhalb einer Stunde zu laden, bzw. zu entladen. Allerdings kann es in der Realität Einschränkungen geben, wie viel Energie ein Speicher pro Stunde aufnehmen kann, bzw. in Leistung umwandeln kann. Aus diesem Grund wurde das Modell um eine maximale Handelskapazität erweitert, sodass der Speicher eines Agenten zwar nach wie vor eine Kapazität von 500MWh hat, jedoch nur 100 bzw. 50MWh pro Stunde aufnehmen oder entladen kann. Der Effekt von unterschiedlichen Handelskapazitäten ist in Abbildung 2.24 dargestellt. Wie man erkennen

kann, hat eine geringere Handelskapazität einen negativen Einfluss auf den Gewinn des Agenten. Dies liegt insbesondere daran, dass ein Agent, der nur mit 50MWh pro Stunde handeln kann, nicht mehr nur an den lokalen Extremstellen ein- bzw. verkaufen kann, sondern auch auf die drumherum liegenden Stunden ausweichen muss, um die gesamte Kapazität des Speichers ausnutzen zu können.

3 Fazit

In dieser Masterarbeit wurde die Anwendbarkeit von Reinforcement Learning Verfahren zur agentenbasierten Modellierung des Energiemarktes untersucht. Insbesondere wurde die Einführung von Energiespeicherkraftwerken durch Reinforcement Learning Agenten simuliert und dessen Wirtschaftlichkeit und Effekt auf den Energiemarkt analysiert, wozu es bisher noch keine vergleichbaren wissenschaftlichen Arbeiten gab. Die Strategien der Energiespeicheragenten, die durch einen „Deep Q-Learning“ Ansatz erlernt wurden, weisen ein rationales Handelsverhalten auf und sind sogar mit der optimalen Lösung, die mithilfe eines numerischen Verfahrens unter perfekten Bedingungen berechnet wurde, im Hinblick auf den erzielten Gewinn vergleichbar. Selbst für unterschiedliche physikalische Eigenschaften des Speichers und unter variablen Marktbedingungen durch konkurrierende Agenten konnten die Reinforcement Learning Agenten eine erfolgreiche Handelsstrategie erlernen, ohne dabei den zugrundeliegenden Algorithmus anpassen zu müssen. Somit eignet sich der Reinforcement Learning Ansatz sehr gut um den Energiemarkt unter Einführung von Energiespeichern realistisch zu modellieren. Dennoch lohnt es sich neben dem hier angewandten „deep Q-Network“ Algorithmus alternative Reinforcement Learning Verfahren zu untersuchen, da dieser lediglich mit diskreten Aktionsräumen umgehen kann. Dadurch kann ein Agent zwar über den Zeitpunkt eines Kaufs entscheiden, jedoch z.B. nicht über die gewünschte Energiemenge. Dazu könnte insbesondere die Anwendung des „Multi-Agent-Deep-Deterministic-Policy-Gradient“ [22] Verfahrens analysiert werden, das nicht nur in der Lage ist auf unendlichdimensionalen Aktionsräumen angewandt zu werden, sondern auch speziell für Multi-Agentenbasierte Probleme entworfen wurde.

Gleichzeitig wurde in dieser Arbeit auch gezeigt, dass Reinforcement Learning nicht für

alle Probleme geeignet ist. So kamen bei der Untersuchung des „Day-Ahead“ Marktes erhebliche Zweifel über die Konvergenz und Anwendbarkeit des Q-Learning Verfahrens auf. Es hat sich herausgestellt, dass der deutlich simplere „k-armed-Bandit“ Ansatz die gleichen Ergebnisse lieferte. Zudem konnte eine realistischere Marktsimulation insbesondere durch das Anpassen der Belohnungssignale erreicht werden. Dies zeigt, dass vor allem die Modellierung der Umgebung einen großen Einfluss auf die Performanz von Reinforcement Learning Algorithmen hat.

Literaturverzeichnis

[1] World Development Indicators: Electricity production, sources, and access. `http://wdi.worldbank.org/table/3.7`. abgerufen am 27. November 2019.

[2] Hans Alt. *Lineare Funktionanalysis*. 2002.

[3] Leemon C. Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. In *ICML*, 1995.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[5] Emma Brunskill. Lecture notes Reinforcement Learning. `http://web.stanford.edu/class/cs234/index.html`, January 2019.

[6] Umwelt Bundesamt. Energiebedingte Emissionen. `https://www.umweltbundesamt.de/daten/energie/energiebedingte-emissionen`. abgerufen am 27. November 2019.

[7] Lucian Busoniu, Robert Babuska, and Bart De Schutter. *Multi-agent Reinforcement Learning: An Overview*, volume 310, pages 183–221. 07 2010.

[8] François Chollet et al. Keras. `https://keras.io`, 2015.

[9] Marc Deissenroth, Martin Klein, Kristina Nienhaus, and Matthias Reeg. Assessing the Plurality of Actors and Policy Interactions: Agent-Based Modelling of Renewable Energy Market Integration. *Complexity*, 2017:1–24, 12 2017.

[10] François Deloche. RNN Bilder. https://commons.wikimedia.org/wiki/User:Ixnay. abgerufen am 11. März 2019.

[11] Wissenschaftlichen Dienst des Deutschen Bundestags. Vor- und Nachteile verschiedener Energiespeichersysteme. https://www.bundestag.de/resource/blob/412904/ca2dd030254284687a1763059f1f4c0c/wd-8-032-14-pdf-data.pdf. abgerufen am 28. September 2019.

[12] Kyunghyun Cho et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014.

[13] Arthur Flajolet and Patrick Jaillet. Real-Time Bidding with Side Information. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5162–5172. Curran Associates, Inc., 2017.

[14] Bundesministerium für Wirtschaft und Energie. EEG-Erfahrungsbericht. https://www.erneuerbare-energien.de/EE/Redaktion/DE/Downloads/bmwi_de/eeg-erfahrungsbericht.pdf?__blob=publicationFile&v=4. abgerufen am 27. November 2019.

[15] Bundesministerium für Wirtschaft und Energie. Förderung der erneuerbaren Energien. https://www.erneuerbare-energien.de/EE/Redaktion/DE/Standardartikel/gesetze.html. abgerufen am 28. September 2019.

[16] Andreas Hauer. Energiespeicher - Technologien und Anwendungen. https://www.energie-innovativ.de/fileadmin/user_upload/energie_innovativ/Energiedialog/Dokumente/2014-11-20-Energiedialog-Energiespeicher-Technologien-und-Anwendung.pdf.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, 12 1997.

[18] Jochen Homann. *Die Bundesnetzagentur*, pages 611–627. Springer Fachmedien Wiesbaden, Wiesbaden, 2017.

[19] Benjamin Van Roy John N. Tsitsiklis. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 05 1997.

[20] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/neural-networks-3/`. Abgerufen am 18. November 2019.

[21] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 12 2014.

[22] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *CoRR*, abs/1706.02275, 2017.

[23] Francisco S. Melo. Convergence of Q-learning: a simple proof. `http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf`.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[25] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications. *CoRR*, abs/1812.11794, 2018.

[26] Martin L. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. In *Wiley Series in Probability and Statistics*, 1994.

[27] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[28] Yves-Marie Saint-Drenan, Amany von Oehsen, Norman Gerhardt, Dr. Michael Sterner, Dr. Stefan Bofinger, and Dr. Kurt Rohrig. Dynamische Simulation der Stromversorgung in Deutschland nach dem Ausbauszenario der Erneuerbaren-Energien-Branche. `https://www.bee-ev.de/fileadmin/Publikationen/Studien/100119_BEE_IWES-Simulation_Stromversorgung2020_Endbericht.pdf`. abgerufen am 28. September 2019.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[30] Tesla. Tesla Powerpack to Enable Large Scale Sustainable Energy to South Australia. 2017.

[31] Anke Weidlich. *Engineering Interrelated Electricity Markets - An Agent-Based Computational Approach*. 10 2008.

[32] Dirk Werner. *Funktionalanalysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[33] Richard A. Zahoransky, Hans-Josef Allelein, Elmar Bollin, Helmut Oehler, Udo Schelling, and Harald Schwarz. *Liberalisierung der Energiemärkte*. Springer Fachmedien Wiesbaden, Wiesbaden, 2013.

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäßaus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Ort, Datum Unterschrift