Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

Masterprojekt

im Studiengang Elektrische Systeme

Reinforcement Learning zur Regelung eines Batteriespeichersystems unter Unsicherheit

Vorgelegt von

Chin-I Feng

Matrikel-Nummer: 300545

Durchgeführt an der

Hochschule Konstanz

Technik, Wirtschaft und Gestaltung

Betreuung:

Prof. Dr. Gunnar Schubert

M.Sc. Johannes Nicklaus

Konstanz, 23. Februar 2026

Kurzfassung

Dieses Masterprojekt untersucht den Einsatz von RL zur optimalen Regelung eines BESS unter Berücksichtigung von Strompreis- und Lastunsicherheiten. Hierfür wurde eine Simulationsumgebung in Python entwickelt, die Zustandsgrößen wie SoC und SoH modelliert sowie prognostizierte Preis- und Lastdaten mit Unsicherheiten einbezieht. Neben einem preisbasierenden Schwellenregel-Controller wurden die RL-Algorithmen DQN, TD3 und QR-DQN eingesetzt und evaluiert.

Die Ergebnisse zeigen, dass lernbasierte Ansätze insbesondere im Arbitrage-Szenario höhere Erlöse erzielen als der regelbasierte Ansatz. QR-DQN weist dabei die beste Performance unter den untersuchten RL-Agenten auf, da Unsicherheiten in prognostizierten Daten robuster berücksichtigt werden können. Im Peak-Shaving-Szenario gelingt es DQN und QR-DQN, Lastspitzen deutlich besser zu reduzieren als dem regelbasierten Ansatz und TD3, wenngleich keine vollständige Glättung erreicht wird.

Abstract

This master's project investigates the use of RL for the optimal control of a BESS under electricity price and demand uncertainty. A simulation environment was developed in Python to model state variables such as SoC and SoH, while incorporating forecasted price and demand data together with associated uncertainties. In addition to a price-based threshold controller, the RL algorithms DQN, TD3, and QR-DQN were applied and evaluated.

The results show that learning-based approaches achieve higher revenues in the arbitrage scenario compared to the rule-based controller. Among the evaluated RL agents, QR-DQN demonstrates the best performance, as it can handle uncertainties in forecasted data more robustly. In the peak-shaving scenario, DQN and QR-DQN reduce load peaks more effectively than the rule-based approach and TD3, although complete smoothing of longer load peaks is not achieved.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BESS	Batteriespeichersystem
DL	Deep Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EFC	Equivalent Full Cycles
KI	Künstliche Intelligenz
QR-DQN	Quantile Regression Deep Q-Network
RL	Reinforcement Learning
SB3	Stable-Baselines3
SoC	State of Charge
SoH	State of Health
TD3	Twin Delayed Deep Deterministic Policy Gradient

1 Einleitung

Stark volatile Strompreise und unsichere Marktentwicklungen schränken die Effizienz regelbasierter Steuerungsansätze für Batteriespeichersysteme (BESS) im Kontext von Energiearbitrage und Peak-Shaving ein [1]. Insbesondere erschwert die Unsicherheit des zukünftigen Last- und Nachfrageverlaufs eine zuverlässige Planung von Lade- und Entladestrategien für das Peak-Shaving [2]. Diese Lücke führt dazu, dass wirtschaftliche Potenziale zur Senkung der Stromkosten nicht vollständig ausgeschöpft werden [3].

Die optimale Planung von BESS erfordert eine dynamische Entscheidungsfindung, um mit unsicheren Strompreisen und variablen Lastverläufen umgehen zu können [4]. Methoden des Reinforcement Learning (RL) ermöglichen die Ableitung adaptiver Entscheidungsstrategien auf Basis der direkten Interaktion mit der Umgebung und eignen sich daher für den Betrieb von BESS ohne explizite Systemmodelle [5] [6].

Das Hauptziel dieses Masterprojekts ist es daher, eine RL-Umgebung für BESS in Python zu entwickeln und verschiedene RL-Regler bzw. RL-Agenten zur optimalen Steuerung eines BESS unter unsicheren Strompreisen und Nachfrageschwankungen zu analysieren und zu testen, um deren Verhalten im Hinblick auf Energiearbitrage und Peak-Shaving zu untersuchen. Ziel der Arbeit ist es zudem, die genannten RL-Agenten zu validieren, um deren Leistungsfähigkeit gegenüber Prognoseunsicherheiten bewerten zu können. Daraus ergeben sich die folgenden Forschungsfragen:

(1) Wie effektiv sind RL-Regler für die Steuerung von BESS unter Preis- und Lastunsicherheit im Vergleich zu regelbasierten Ansätzen?

(2) Wie robust sind RL-Agenten gegenüber Prognoseunsicherheiten?

(3) Welche RL-Agenten zeigen die beste Performance bei Energiearbitrage und Peak-Shaving?

Im weiteren Verlauf dieser Arbeit werden die Entwicklung einer Simulationsumgebung für ein BESS sowie der Vergleich der eingesetzten RL-Agenten beschrieben. In **Kapitel 2** werden die angewandten RL-Algorithmen erläutert. **Kapitel 3** beschreibt die Entwicklung der Simulationsumgebung mit BESS-Modellierung sowie Zustands-, Aktions- und Belohnungsdefinition.

In **Kapitel 4** wird das Entscheidungsverhalten der trainierten RL-Agenten im Arbitrage- und Peak-Shaving-Szenario mit einem selbst implementierten regelbasierten Controller verglichen. Darüber hinaus werden die regelbasierten Steuerungsansätze als Referenzmethoden vorgestellt. **Kapitel 5** stellt die Simulationsergebnisse dar und ordnet diese in einer kurzen Diskussion ein. Am Ende erfolgt in **Kapitel 6** das Fazit und der Ausblick auf Verbesserungen der Simulation.

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen dieser Arbeit vorgestellt. Zunächst werden die eingesetzten RL-Agenten erläutert, die als Regelungsstrategie für die Steuerung des BESS dienen. Anschließend werden die relevanten Grundlagen zu BESS beschrieben, einschließlich State of Charge (SoC), State of Health (SoH), Batteriedegradation sowie typischer Einsatzszenarien wie Energiearbitrage und Peak-Shaving. Abschließend werden die verwendeten Python-Pakete vorgestellt, die zur Umsetzung der Simulation und der RL-Methoden eingesetzt werden.

2.1 RL-Agenten

Zur Lösung der in dieser Arbeit betrachteten Steuerungsaufgabe wird ein RL-Ansatz verwendet, um geeignete Entscheidungen ohne feste vordefinierte Regeln treffen zu können. Basierend auf den jeweiligen Anforderungen der Anwendung werden angemessene RL-Agenten ausgewählt, die sich hinsichtlich algorithmischer Eigenschaften unterscheiden. Für den speziellen Kontext der Entscheidungslogik kommen folgende Agenten zum Einsatz, die in den nächsten Unterkapiteln konkret vorgestellt werden:

Tab. 2.1: Klassifizierung der Agenten nach Typ

Agentenart	**Agent**
Wertbasiert	Deep Q-Network (DQN)
Actor-Critic	Twin Delayed Deep Deterministic Policy Gradient (TD3)
Distributional	Quantile Regression Deep Q-Network (QR-DQN)

2.1.1 DQN

DQN ist ein wertbasierter Ansatz des Deep Reinforcement Learning (DRL), bei dem nicht direkt eine Policy gelernt wird, sondern die Aktions-Wert-Funktion, auch als Q-Funktion bezeichnet, $Q(s, a)$, approximiert wird [7]. Zur Approximation der Q-Funktion wird in DQN ein neuronales Netz eingesetzt. Durch den Einsatz von Deep Learning (DL) können komplexe und nichtlineare Zusammenhänge zwischen Zuständen s und Aktionen a abgebildet werden, wobei das Ziel des Agenten darin besteht, den Erwartungswert der kumulierten Belohnung $\mathbb{E}[R_t]$ zu maximieren [8].

Wie in Abb. 2.1 dargestellt, erhält das Q-Netzwerk den aktuellen Zustand s_t als Eingabe und berechnet für jede mögliche diskrete Aktion einen Q-Wert $Q_\theta(s_t, a_i)$. Der Output des Netzwerks besteht somit aus einem Vektor von Aktionswerten, wobei jedes

Ausgabeneuron einem spezifischen Element des diskreten Aktionsraums zugeordnet ist. Die Aktionswahl erfolgt durch Maximierung dieser geschätzten Aktionswerte:

$$a_t = \arg\max_a Q_\theta(s_t, a).$$

Die gewählte Aktion a_t wird anschließend an die Umgebung übergeben, welche eine Belohnung r_t sowie den nächsten Zustand s_{t+1} zurückliefert. Während des Trainings wird das Q-Netzwerk mithilfe des temporalen Differenzfehlers aktualisiert, sodass die approximierte Q-Funktion schrittweise an den wahren Erwartungswert des zukünftigen Returns oder der zukünftigen kumulierten Belohnung angepasst wird [7].

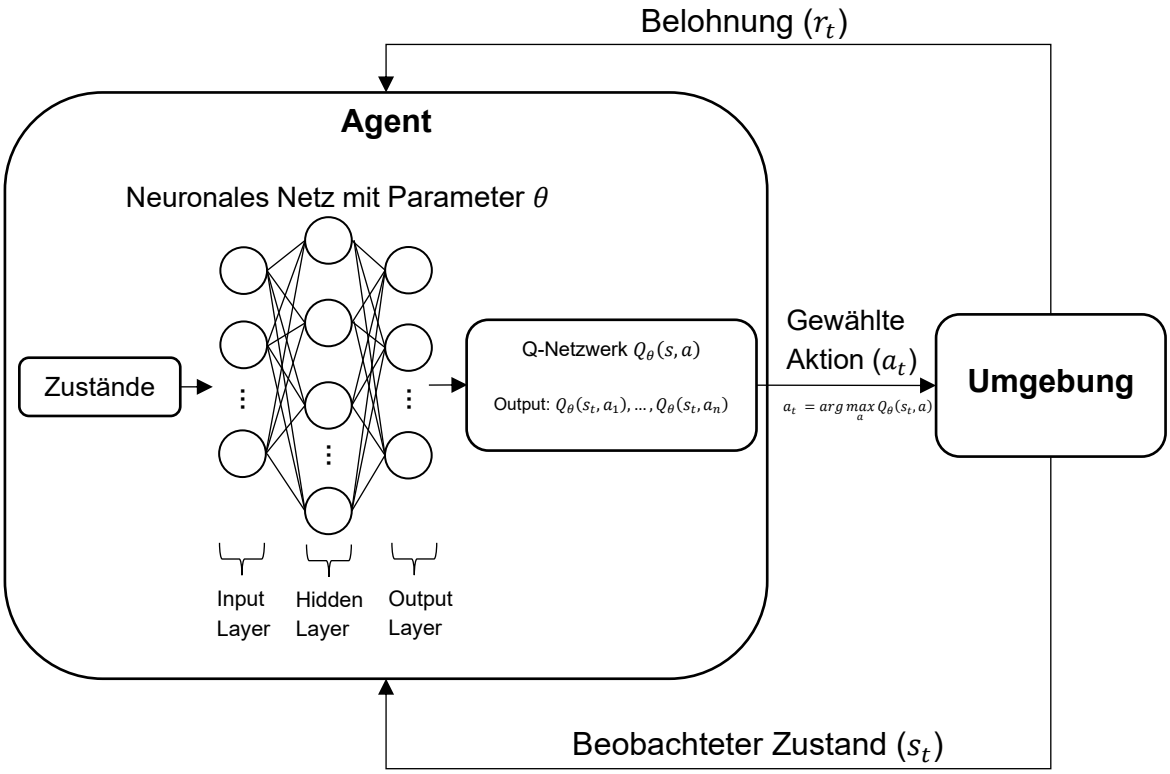

Abb. 2.1: DQN-Architektur

Im Rahmen dieser Arbeit wird DQN für die Steuerung des BESS im diskreten Aktionsraum eingesetzt. Aufgrund der diskreten Aktionsstruktur ist DQN für kontinuierliche Regelungsaufgaben nur bedingt geeignet, erlaubt jedoch eine Analyse des Entscheidungsverhaltens im Vergleich zu Agenten mit kontinuierlichem Aktionsraum.

2.1.2 TD3

TD3 ist ein weiterentwickelter Actor-Critic-Algorithmus für kontinuierliche Aktionsräume. Im Vergleich zu wertbasierten Verfahren wie DQN, die ausschließlich eine Aktions-Wert-Funktion approximieren, kombiniert TD3 eine deterministische Policy mit zwei Aktions-Wert-Funktionen zur stabileren Lernoptimierung [9].

Die Architektur von TD3 ist in Abb. 2.2 dargestellt. Der Algorithmus besteht aus einem Actor-Netzwerk, das eine deterministische Policy $\pi_\theta(s)$ repräsentiert. Für einen gegebenen Zustand s_t erzeugt der Actor direkt eine kontinuierliche Aktion

$$a_t = \pi_\theta(s_t).$$

Diese Aktion a_t wird an die Umgebung übergeben, welche eine Belohnung r_t sowie den nächsten Zustand s_{t+1} zurückliefert.

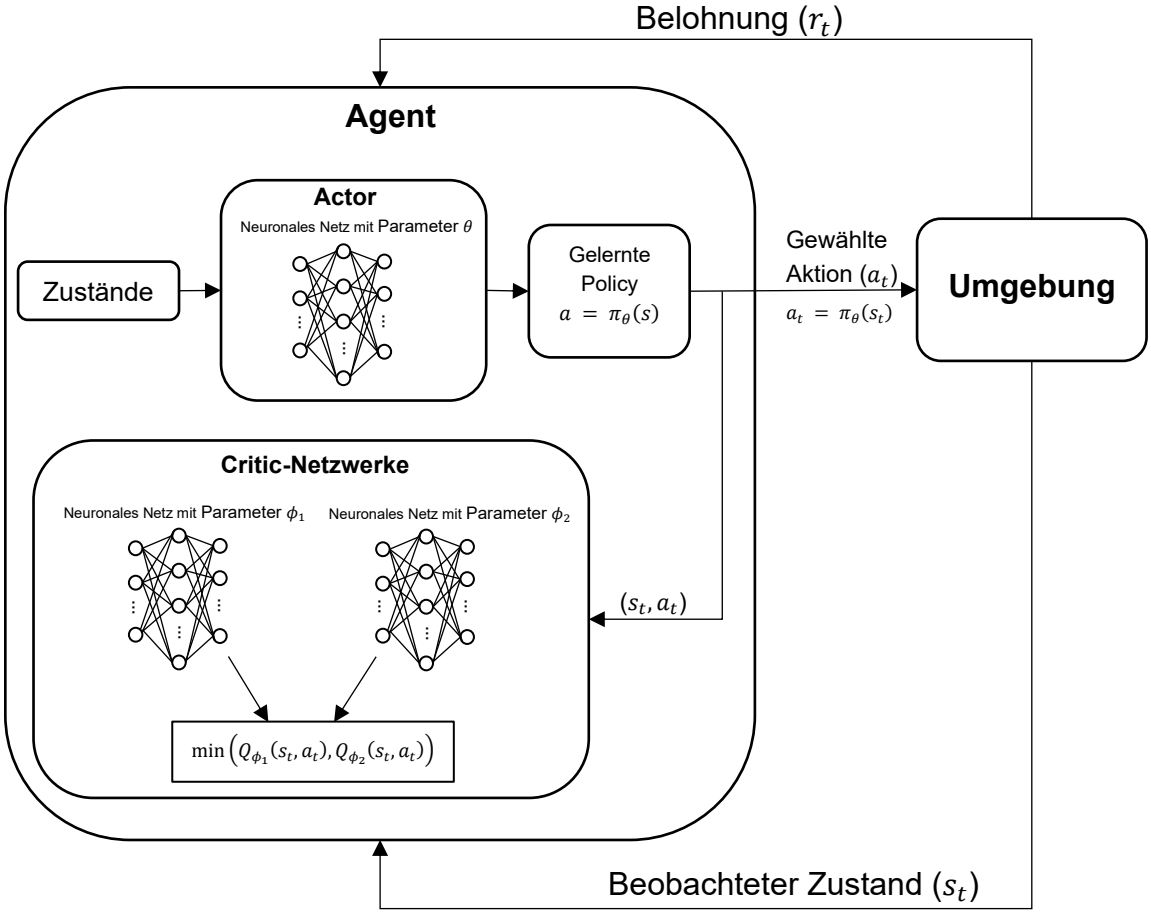

Abb. 2.2: TD3-Architektur

Zusätzlich verfügt TD3 über zwei unabhängige Critic-Netzwerke mit Parametern $\phi_1$und ϕ_2. Diese approximieren jeweils die Aktions-Wert-Funktion

$$Q_{\phi_i}(s_t, a_t), i \in \{1,2\},$$

und schätzen damit den Erwartungswert der kumulierten Belohnung $\mathbb{E}[R_t]$für die gewählte Aktion im aktuellen Zustand s_t. Zusätzlich wird zur Reduktion systematischer Überschätzungen nicht ein einzelner Q-Wert verwendet, sondern das Minimum der beiden Schätzungen:

$$\min\left(Q_{\phi_1}(s_t, a_t), Q_{\phi_2}(s_t, a_t)\right).$$

Diese konservative Bewertung stabilisiert das Training und verhindert systematische Überschätzungen der Aktionswerte. Während die Critic-Netzwerke durch Minimierung des temporalen Differenzfehlers trainiert werden, erfolgt die Aktualisierung des Actor-Netzwerks so, dass die von den Critic-Netzwerken bewerteten Q-Werte maximiert werden [9].

2.1.3 QR-DQN

QR-DQN erweitert das klassische Deep Q-Network um einen distributionalen Ansatz zur Modellierung der Return-Verteilung bzw. Verteilung der diskontierten kumulierten Belohnungssumme [10]. Im Gegensatz zu DQN, das ausschließlich den Erwartungswert der Aktions-Wert-Funktion $Q(s, a)$ approximiert, modelliert QR-DQN die gesamte Verteilung der zukünftigen kumulierten Belohnung [11]. Hierzu approximiert das neuronale Netz nicht direkt $Q(s, a)$, sondern eine Return-Verteilung

$$Z_\theta(s, a),$$

die durch eine diskrete Menge von N Quantilen beschrieben wird:

$$Z_\theta(s, a) = \{z_1(s, a), \dots, z_N(s, a)\}.$$

Die Größen $z_i(s, a)$ entsprechen diskreten Quantilen der Return-Verteilung. Die Quantilniveaus sind dabei gleichmäßig im Wahrscheinlichkeitsraum verteilt, während die Position der Quantile auf der Return-Achse von der zugrunde liegenden Verteilung abhängt.

Wie in Abb. 2.3 dargestellt, erhält das Quantile-Netzwerk den aktuellen Zustand s_t als Eingabe und gibt für jede mögliche diskrete Aktion eine Menge von Quantilen aus. Der Erwartungswert der approximierten Verteilung ergibt sich als Mittelwert der Quantile:

$$Q(s, a) = \mathbb{E}[Z_\theta(s, a)] = \frac{1}{N}\sum_{i=1}^{N} z_i\,(s, a).$$

Die Aktionswahl erfolgt analog zum klassischen DQN durch Maximierung dieses Erwartungswertes:

$$a_t = \arg\max_a Q(s_t, a).$$

Somit bleibt QR-DQN ein wertbasierter Ansatz, erweitert jedoch die Repräsentation des Returns von einem einzelnen Skalarwert zu einer diskreten Verteilung. Während die Aktionsentscheidung auf dem Erwartungswert basiert, wird das Training mithilfe der Quantile-Regression durchgeführt, wodurch die gesamte Verteilung berücksichtigt wird. Dies führt zu einer stabileren und informativeren Approximation der Return-Dynamik im Vergleich zu klassischen DQN-Verfahren [11].

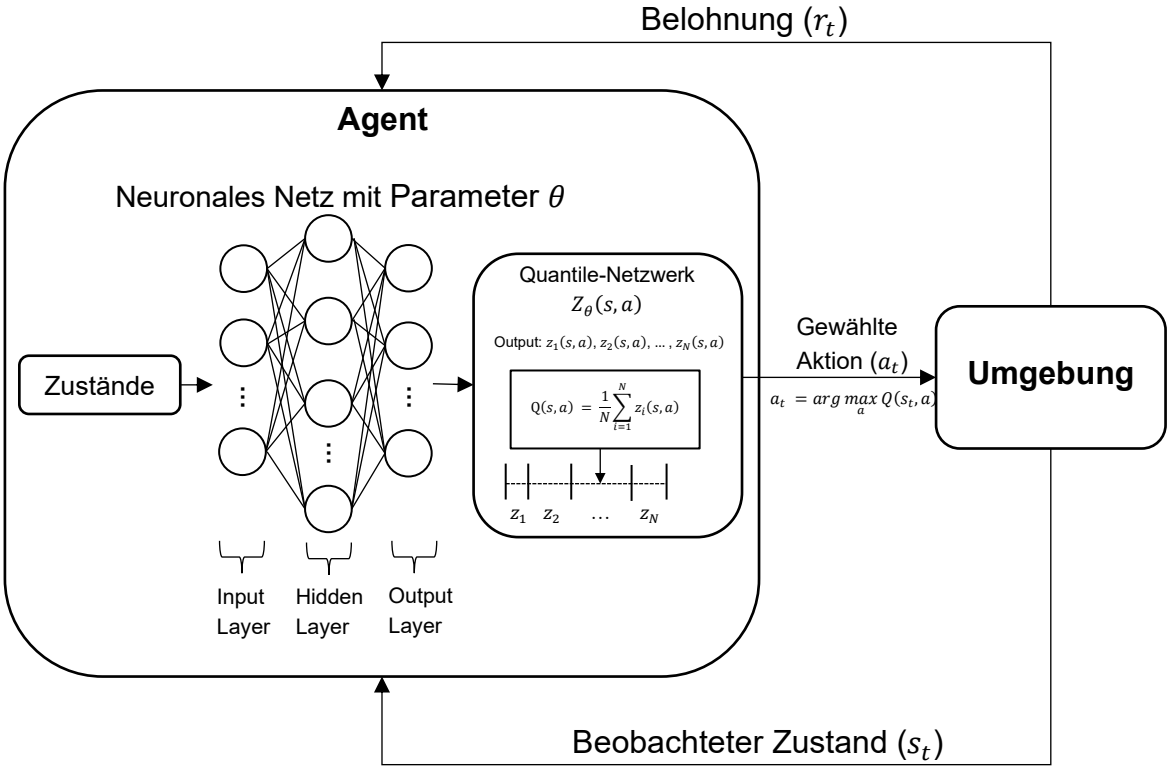

Abb. 2.3: QR-DQN-Architektur

Im Gegensatz zu DQN [7] und TD3 [9], die jeweils lediglich den Erwartungswert der zukünftigen Belohnung approximieren, modelliert QR-DQN [11] die gesamte Verteilung des Returns. Während DQN einen einzelnen Q-Wert pro Aktion schätzt und TD3 eine deterministische Policy auf Basis eines erwarteten Aktionswertes optimiert, approximiert QR-DQN mehrere Quantile der Return-Verteilung [10].

In Anwendungen mit hoher Unsicherheit, beispielsweise bei volatilen Strompreisen und variierenden Lastprofilen im Rahmen dieses Projekts, kann die Modellierung der gesamten Return-Verteilung zu einer robusteren Entscheidungsfindung beitragen [12]. Handlungsoptionen mit identischem Erwartungswert, jedoch unterschiedlichem Risikoprofil, lassen sich dadurch differenzierter bewerten. Der distributionale Ansatz bietet somit einen potenziellen Vorteil gegenüber rein erwartungswertbasierten Verfahren wie DQN und TD3 [10] [12].

2.2 BESS

BESS bilden in diesem Projekt die Simulationsumgebung zur Untersuchung und Bewertung des RL-Agenten und ermöglichen zugleich die Analyse wirtschaftlicher Optimierungspotenziale im Strommarkt durch flexible Energiespeicherung [4] [12]. Die Steuerung eines BESS erfordert die Berücksichtigung technischer Zustandsgrößen, der Batteriedegradation sowie betrieblicher Rahmenbedingungen [2].

In den nachfolgenden Unterkapiteln werden die relevanten Kenngrößen der RL-Umgebung, insbesondere SoC, SoH und Aspekte der Batteriedegradation, sowie die betrachteten Einsatzszenarien für die Simulation erläutert. Dazu zählen wesentliche Zustandsvariablen des BESS und typische Anwendungsfälle wie Energiearbitrage und Peak Shaving.

2.2.1 SoC

SoC beschreibt den aktuellen Ladezustand der Batterie bzw. des BESS und gibt das Verhältnis der gespeicherten Energie zur maximal verfügbaren Kapazität an [13]. Üblicherweise wird der SoC als normierter Wert im Bereich $[0,1]$ angegeben, was einem Bereich von $[0\%, 100\%]$ entspricht. Die Formel ist wie folgt gegeben:

$$SoC_t = \frac{E_t}{E_{max}}, \qquad SoC_t \in [0,1] \tag{2.1}$$

mit

$E_t =$ Aktuell gespeicherte Energie

$E_{max} =$ Maximal verfügbare Energiekapazität

Als Zustandsvariable begrenzt der SoC die physikalisch zulässigen Lade- und Entladeleistungen und bestimmt somit direkt den Aktionsraum des RL-Agenten [12]. Neben der Einhaltung technischer Grenzen ist der SoC entscheidend für die Bewertung wirtschaftlicher Strategien, da er bestimmt, ob Energie für Arbitrage oder Peak Shaving verfügbar ist.

Im Rahmen dieses Projekts wird die zeitliche Entwicklung des Ladezustands durch eine SoC-Update-Gleichung modelliert, die in der RL-Umgebung die Systemdynamik des BESS beschreibt [12]. Dabei wird der SoC im nächsten Zeitschritt aus dem aktuellen Ladezustand SoC_t sowie der gewählten Lade- bzw. Entladeleistung P_t unter Berücksichtigung der Lade- und Entladewirkungsgrade η berechnet. Formal ergibt sich die SoC-Dynamik wie folgt:

$$SoC_{t+1} = SoC_t + \frac{P_t \cdot \Delta t}{E_{max}} \eta(P_t), \qquad SoC_{t+1} \in [0,1] \tag{2.2}$$

$$P_t \in [-P_{max}, P_{max}]$$

$$\eta(P_t) = \begin{cases} \eta_{Laden} & , P_t > 0 \\ \dfrac{1}{\eta_{Entladen}} & , P_t < 0 \end{cases}$$

mit

SoC_t = Ladezustand der Batterie zum Zeitpunkt t
SoC_{t+1} = Ladezustand im nächsten Zeitschritt
P_t = Lade- bzw. Entladeleistung zum Zeitpunkt t
P_{max} = Maximal zulässige Lade- bzw. Entladeleistung
Δt = Länge eines Zeitschritts
E_{max} = Maximal verfügbare Energiekapazität
η_{Laden} = Wirkungsgrad beim Laden
$\eta_{Entladen}$ = Wirkungsgrad beim Entladen

2.2.2 SoH

SoH beschreibt den Gesundheitszustand der Batterie und charakterisiert den verbleibenden nutzbaren Kapazitätsanteil im Vergleich zur Nennkapazität zum Neuzustand [13]. Typischerweise wird der SoH ebenfalls normiert angegeben, wobei ein Wert von 1 (bzw. 100 %) einer neuen Batterie entspricht. Die Formel lautet:

$$SoH(t) = \frac{C_{Aktuell}(t)}{C_{Nominal}}, \qquad SoH(t) \in [0,1] \tag{2.3}$$

mit

$C_{Aktuell}(t)$ = Aktuell verfügbare Kapazität
$C_{Nominal}$ = Nennkapazität im Neuzustand

In der Simulationsumgebung dient der SoH als Maß für die langfristige Systemperformance und ermöglicht die explizite Berücksichtigung von Degradationskosten. Dadurch wird verhindert, dass kurzfristig gewinnmaximierende

Strategien zu übermäßiger Zyklisierung und langfristig erhöhten Alterungskosten führen [2].

Der SoH nimmt im Zeitverlauf infolge elektrochemischer Alterungsprozesse kontinuierlich ab [14]. Ein gängiges Maß zur Quantifizierung der zyklischen Degradation ist die Anzahl der Equivalent Full Cycles (EFC). Ein EFC entspricht einem vollständigen Lade- und Entladezyklus (z.B. von 0% auf 100% und anschließend zurück auf 0%) über die gesamte nutzbare Kapazität der Batterie [15]. In diskreter Zeit ergibt sich die inkrementelle Aktualisierung der EFC wie folgt:

$$EFC_{t+1} = EFC_t + \frac{1}{2} |SoC_{t+1} - SoC_t| \tag{2.4}$$

mit

$EFC_t =$ Anzahl der äquivalenten Vollzyklen zum Zeitpunkt t
$EFC_{t+1} =$ Anzahl der äquivalenten Vollzyklen im nächsten Zeitschritt
$SoC_t =$ Ladezustand zum Zeitpunkt t
$SoC_{t+1} =$ Ladezustand im nächsten Zeitschritt

Teilzyklen werden dabei anteilig berücksichtigt und zu äquivalenten Vollzyklen aggregiert. Mit zunehmender Anzahl an EFCs reduziert sich der SoH, da jede Zyklisierung zu einer inkrementellen Kapazitätsdegradation führt [15]. Die inkrementelle SoH-Änderung ergibt sich zu:

$$SoH_{t+1} = SoH_t - k_{Degradation} \cdot \Delta EFC_t, \qquad SoH_{t+1} \in [0,1] \tag{2.5}$$

mit

$SoH_t =$ Gesundheitszustand der Batterie zum Zeitpunkt t
$SoH_{t+1} =$ Gesundheitszustand der Batterie im nächsten Zeitschritt
$\Delta EFC_t =$ Zunahme der äquivalenten Vollzyklen im Zeitschritt t
$k_{Degradation} =$ Degradationskoeffizient pro EFC

Im Rahmen dieses Projekts wird die Batteriedegradation in monetarisierter Form in die Belohnungsfunktion integriert. Dadurch entsteht ein Zielkonflikt zwischen kurzfristiger Gewinnmaximierung und langfristiger Schonung des BESS.

2.2.3 Einsatzszenarien

In dieser Arbeit werden zwei Einsatzszenarien für das BESS betrachtet und simuliert, nämlich Energiearbitrage und Peak-Shaving. Beide Anwendungsfälle stellen unterschiedliche Anforderungen an die Betriebsstrategie des BESS und an die wirtschaftliche Performance [16] [17]. Die verwendeten Einsatzszenarien werden in den nächsten Absätzen beschrieben.

Energiearbitrage

Bei der Energiearbitrage wird das BESS genutzt, um Preisunterschiede am Strommarkt auszunutzen [16]. Energie wird in Zeiträumen mit niedrigen Strompreisen geladen und in Phasen hoher Preise wieder entladen, um den wirtschaftlichen Gewinn durch zeitliche Verschiebung des Energiebezugs zu maximieren. Wie in Abb. 2.4 dargestellt, wird bei niedrigen Preisen geladen und bei hohen Preisen entladen.

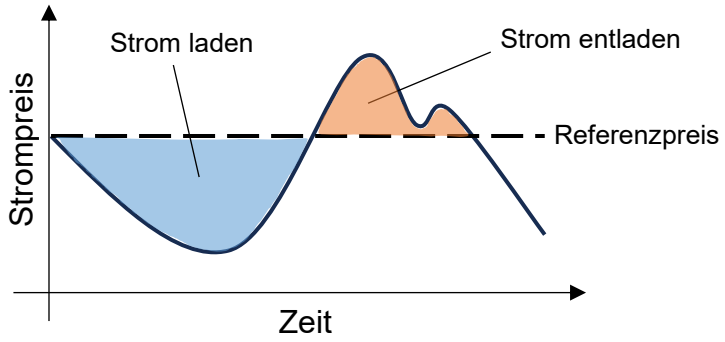

Abb. 2.4: Prinzip der Energiearbitrage anhand eines Strompreisverlaufs

Peak-Shaving

Beim Peak-Shaving dient das BESS der Reduktion von Lastspitzen. Ziel ist es, die maximale Netzanschlussleistung zu begrenzen und dadurch leistungsabhängige Netzentgelte zu reduzieren [17]. In Zeiten hoher Last wird Energie aus dem BESS bereitgestellt, um die Netzlast zu glätten, während in lastarmen Phasen geladen wird. Das grundlegende Prinzip ist in Abb. 2.5 dargestellt, in der die Reduktion der Lastspitzen durch den Einsatz des BESS schematisch veranschaulicht wird.

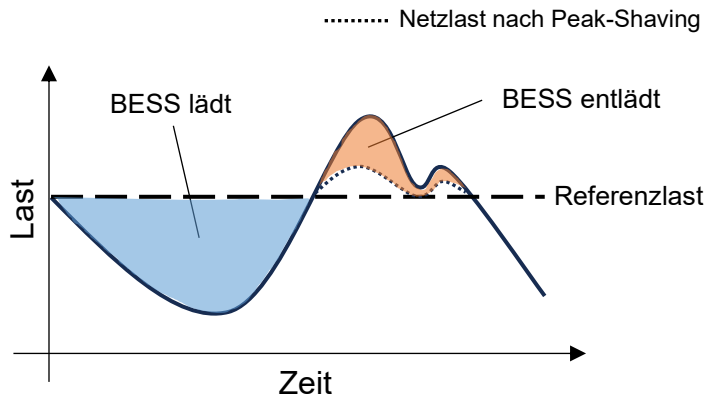

Abb. 2.5: Prinzip des Peak-Shaving anhand eines Lastverlaufs

Im Gegensatz zur Arbitrage steht hier nicht primär die Ausnutzung von Preisdifferenzen im Vordergrund, sondern die Minimierung von Spitzenlasten. Die Betriebsstrategie ist daher stärker an der Lastkurve als am Strompreis orientiert.

2.3 Python-Pakete

Python ist eine interpretierte Hochsprache, die besonders im Bereich der künstlichen Intelligenz (KI) weltweit angewendet wird [18]. Durch den Einsatz etablierter Frameworks können RL-Agenten effizient implementiert und trainiert werden, sodass der Fokus dieser Arbeit auf der Modellierung des BESS sowie der Gestaltung von Zustands- und Belohnungsfunktionen liegt. Die in dieser Arbeit verwendeten Python-Pakete werden in den nächsten Absätzen beschrieben.

NumPy

NumPy stellt mehrdimensionale Arrays bereit, die durch eine umfangreiche Sammlung numerischer Funktionen unterstützt werden. Diese Arrays bilden strukturierte Datencontainer, die entlang mehrerer Dimensionen organisiert sind und als Grundlage für die effiziente Implementierung mathematischer Operationen und komplexer Berechnungen dienen [19].

Pandas

Pandas ermöglicht die effiziente Verarbeitung komplexer Datentabellen unterschiedlicher Datentypen sowie von Zeitreihendaten, was über die Funktionalität reiner NumPy-Arrays hinausgeht. Insbesondere die in dieser Arbeit verwendeten DataFrames stellen flexible Datenstrukturen dar, mit denen Daten aus verschiedenen Quellen importiert, strukturiert und weiterverarbeitet werden können. Anschließend können die importierten Daten frei zugeschnitten, fehlende Elemente behandelt, hinzugefügt, umgeformt, umbenannt und ähnlich wie in Excel-Tabellen visualisiert werden [19].

Matplotlib

Matplotlib ist eine Python-Bibliothek für MATLAB-ähnliche 2D-Diagramme, die umfangreiche Funktionen enthält, um qualitativ hochwertige Grafiken aus Daten zu erstellen und diese interaktiv zu visualisieren. Darüber hinaus ermöglicht sie ausführliche Optionen durch Plotarten, Achsenformate, Schrifteigenschaften und mehr, mit denen man die Visualisierung präziser und flexibler gestalten kann [19].

Gymnasium

Gymnasium ist eine Python-Bibliothek zur Definition und Verwaltung von RL-Umgebungen. Diese Bibliothek stellt eine einheitliche Schnittstelle zur Verfügung, über die Zustandsräume, Aktionsräume sowie Übergangs- und Belohnungsfunktionen definiert werden können [20].

Stable-Baselines3 (SB3)

SB3 ist eine Open-Source-Bibliothek zur Implementierung und zum Training von RL-Algorithmen. Sie stellt standardisierte und getestete Implementierungen moderner RL-Algorithmen bereit [21]. In dieser Arbeit werden insbesondere DQN, QR-DQN und TD3 eingesetzt, um unterschiedliche Lernansätze für die Steuerung des BESS zu untersuchen.

3 Simulationsumgebung

Das nachfolgende Kapitel erläutert die entwickelte Simulationsumgebung für ein netzgekoppeltes BESS, welche als Trainings- und Evaluationsbasis für den RL-Regler dient. Diese Umgebung modelliert die Batteriedynamik (SoC und SoH), Strommarktpreise sowie Prognoseunsicherheiten und monetarisiert die entstehende Degradation.

Das Gesamtsystem wird in einer Gymnasium-kompatiblen Struktur implementiert. Dadurch können verschiedene RL-Ansätze unter realitätsnahen und reproduzierbaren Bedingungen systematisch untersucht werden.

3.1 Systemarchitektur der Simulationsumgebung

Die entwickelte Simulationsumgebung umfasst zwei unterschiedliche Betriebsmodi des BESS, nämlich die Energiearbitrage im Strommarkt sowie das Peak-Shaving zur Lastspitzenreduktion. Beide Szenarien basieren auf einer gemeinsamen physikalischen Batteriemodellierung, unterscheiden sich jedoch hinsichtlich Zielsetzung, Belohnungsfunktion und Betriebskontext.

3.1.1 Arbitrage-Szenario

Im Arbitrage-Szenario operiert das BESS marktgetrieben in Abhängigkeit vom Strompreis und tauscht Energie bidirektional mit dem Stromnetz aus, wie in Abb. 3.1 dargestellt. Der dargestellte Energiefluss bildet die physikalische Grundlage des Systems, während die eigentliche Entscheidungslogik durch einen RL-Regler implementiert wird.

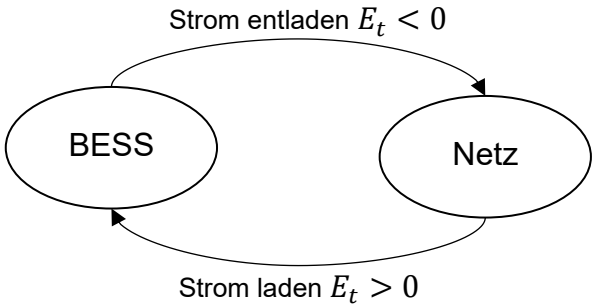

Abb. 3.1: Darstellung des Energieflusses im Arbitrage-Szenario

Der RL-Regler beobachtet zu jedem Zeitpunkt den aktuellen Systemzustand, bestehend aus dem SoC, dem SoH, dem aktuellen Strompreis sowie gegebenenfalls Preisprognosen einschließlich deren Unsicherheit (Prognosefehler). Auf Basis dieses Zustands entscheidet der Regler über die Lade- oder Entladeleistung des Speichers innerhalb der zulässigen Leistungsgrenzen.

Ziel des Reglers ist es, durch zeitlich optimierte Lade- und Entladeentscheidungen Preisunterschiede auszunutzen. Dabei soll Energie bevorzugt in Niedrigpreisphasen aufgenommen und in Hochpreisphasen wieder abgegeben werden. Gleichzeitig berücksichtigt der Agent technische Restriktionen sowie die durch Zyklisierung entstehende Batteriealterung, die als Degradationskosten in die Belohnungsfunktion eingeht.

3.1.2 Peak-Shaving-Szenario

Im Peak-Shaving-Szenario dient das BESS der Glättung von Lastspitzen und ist direkt mit der Last sowie dem Stromnetz verbunden, wie in Abb. 3.2 dargestellt. Der dargestellte Energiefluss beschreibt die physikalische Struktur des Systems, während die eigentliche Entscheidungslogik durch einen RL-Regler realisiert wird.

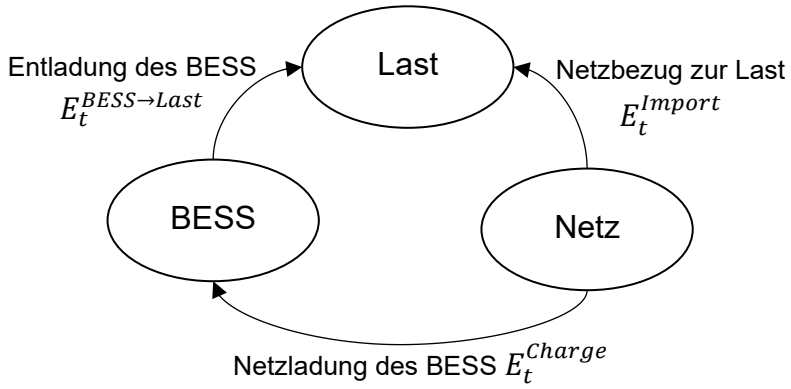

Abb. 3.2: Darstellung des Energieflusses im Peak-Shaving-Szenario

Der RL-Regler beobachtet zu jedem Zeitpunkt den aktuellen Systemzustand, bestehend aus dem SoC, dem SoH, der aktuellen Last, dem aktuellen Strompreis sowie gegebenenfalls Preisprognosen einschließlich deren Unsicherheit (Prognosefehler). Auf Basis dieses Zustands entscheidet der Regler über die Entlade- oder Ladeleistung des Speichers innerhalb der zulässigen Leistungsgrenzen, um Lastspitzen gezielt zu reduzieren.

In dieser Arbeit wird Peak-Shaving so implementiert, dass das BESS während einer Entladung zur Lastversorgung nicht gleichzeitig aus dem Stromnetz geladen werden kann. Eine gleichzeitige Be- und Entladung des BESS ist somit ausgeschlossen. In lastarmen Phasen ist es hingegen zulässig, dass sowohl die Last als auch das BESS gleichzeitig Strom aus dem Netz beziehen. Während das BESS zur Glättung einer Lastspitze entlädt, deckt das Netz ausschließlich die verbleibende Restlast.

Ziel des Reglers ist es, Lastspitzen möglichst effektiv zu reduzieren und damit netz- oder tarifbedingte Leistungskosten zu minimieren, wobei technische Restriktionen sowie batteriebedingte Alterungseffekte berücksichtigt werden.

3.2 Marktdaten und Unsicherheitsmodell

Dieser Abschnitt beschreibt die verwendeten Strompreis- und Lastdaten sowie das implementierte Unsicherheitsmodell. Die zeitlich aufgelösten Strompreise und Lastprofile bilden die Grundlage für die Bewertung der Lade- und Entladeentscheidungen im Arbitrage- und Peak-Shaving-Szenario.

Da sowohl Strompreise als auch Lastverläufe realen Prognoseunsicherheiten unterliegen, werden neben den tatsächlichen Zeitreihen auch Prognosewerte einschließlich modellierter Prognosefehler berücksichtigt. Dadurch kann analysiert werden, inwiefern Unsicherheitsinformationen die Entscheidungsfindung des RL-Reglers beeinflussen.

3.2.1 Strompreisdaten

Für die Simulation werden Day-Ahead-Strompreise verwendet, die über die Plattform Energy Charts des Fraunhofer ISE bezogen wurden [22]. Die Daten liegen in einer zeitlichen Auflösung von 15 Minuten vor. In Abb. 3.3 ist erkennbar, dass die Preise eine ausgeprägte Volatilität aufweisen, einschließlich kurzfristiger Preisspitzen sowie negativer Preisphasen.

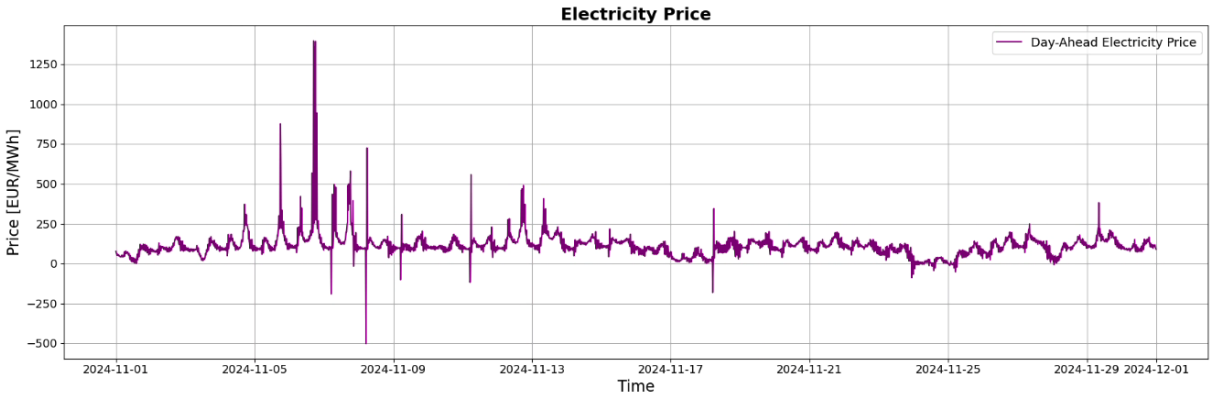

Abb. 3.3: Day-Ahead-Strompreise in 15-minütiger Auflösung

Kurzfristige Preisspitzen eröffnen potenzielle Arbitragemöglichkeiten, während negative Preise gezielte Ladeentscheidungen begünstigen können. Gleichzeitig erschwert die starke Dynamik eine rein deterministische Planung, wodurch lernbasierte Verfahren mit expliziter Berücksichtigung von Unsicherheiten motiviert werden.

Die Strompreise werden in der Simulationsumgebung als exogene Eingangsdaten behandelt und dienen sowohl zur Berechnung der ökonomischen Erlöse als auch als Bestandteil des Zustandsraums des RL-Reglers.

3.2.2 Lastsdaten

Für die Simulation werden elektrische Lastdaten verwendet, die über die Plattform SMARD der Bundesnetzagentur bezogen wurden [23]. Die Originaldaten repräsentieren die aggregierte Gesamtlast Deutschlands und liegen in einer zeitlichen Auflösung von 15 Minuten vor. In Abb. 3.4 ist eine ausgeprägte tagesperiodische Struktur erkennbar, die sich in wiederkehrenden Lastmaxima und -minima im Tagesverlauf äußert. Diese periodischen Muster sind charakteristisch für industrielle, gewerbliche sowie private Verbrauchsstrukturen.

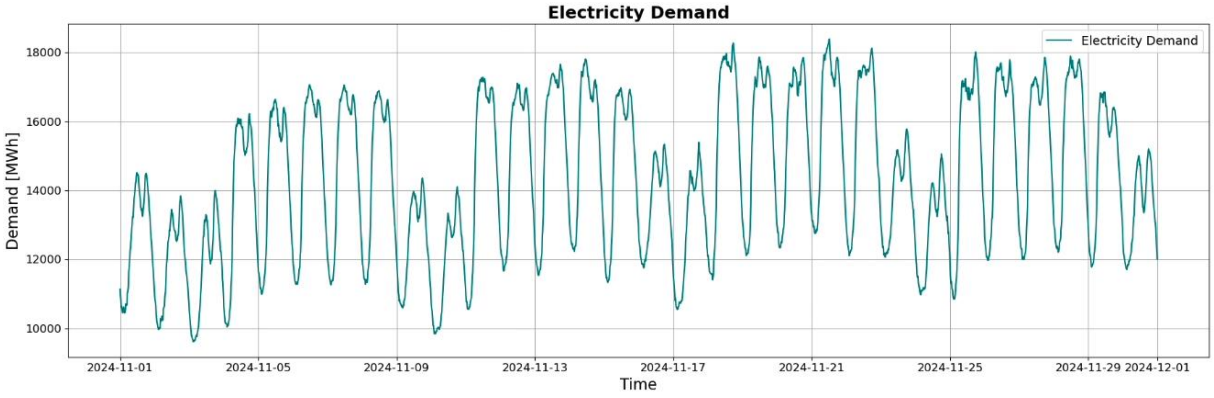

Abb. 3.4: Elektrische Gesamtlast Deutschlands in 15-minütiger Auflösung

Da die SMARD-Daten die Gesamtlast des deutschen Stromsystems abbilden und somit Größenordnungen im Bereich mehrerer Gigawatt aufweisen, werden die Lastwerte für die Simulation skaliert. Hierzu wird ein konstanter Skalierungsfaktor von 10^{-7}angewendet (siehe Abb. 3.5).

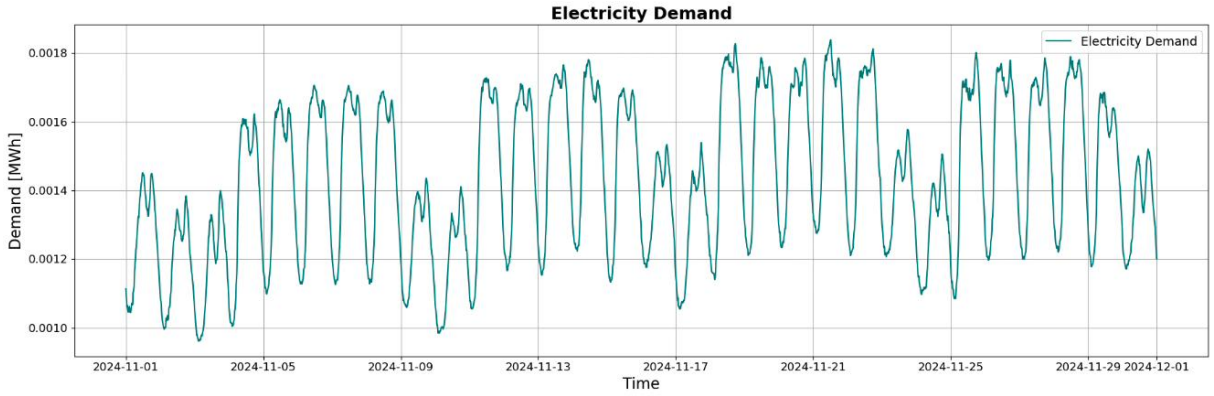

Abb. 3.5: Skalierte Lastdaten für die Simulation in 15-minütiger Auflösung

Die Skalierung ist erforderlich, um die Last auf eine für das modellierte BESS realistische Größenordnung zu übertragen. Ziel ist es, ein repräsentatives Lastprofil beizubehalten, während die absolute Leistungsgröße an die Dimensionierung des simulierten Speichersystems angepasst wird. Dadurch bleibt die charakteristische

zeitliche Struktur der Last erhalten, ohne unrealistische Leistungsanforderungen im Verhältnis zur Speicherkapazität zu erzeugen.

Die skalierten Lastdaten werden in der Simulationsumgebung als exogene Eingangsdaten verwendet und dienen im Peak-Shaving-Szenario als Grundlage für die Bestimmung der Restlast sowie der zu reduzierenden Lastspitzen.

3.2.3 Modellierung der Prognoseunsicherheit

In der Simulationsumgebung werden die historischen Markt- und Lastdaten als reale Zeitreihen verwendet. Der RL-Regler erhält zum aktuellen Zeitpunkt t den tatsächlichen, fehlerfreien Wert x_t (Strompreis oder Last). Zukünftige Werte x_{t+h} sind dem Regler jedoch nicht direkt zugänglich.

Stattdessen werden für zukünftige Zeitpunkte prognostizierte Werte $\tilde{x}_{t+h}$ bereitgestellt, die mit einem stochastischen Prognosefehler behaftet sind. Dadurch wird verhindert, dass der Agent implizit perfekte Kenntnis über den zukünftigen Verlauf der Marktdaten erhält, was in realen Anwendungen nicht möglich ist (siehe Abb. 3.6).

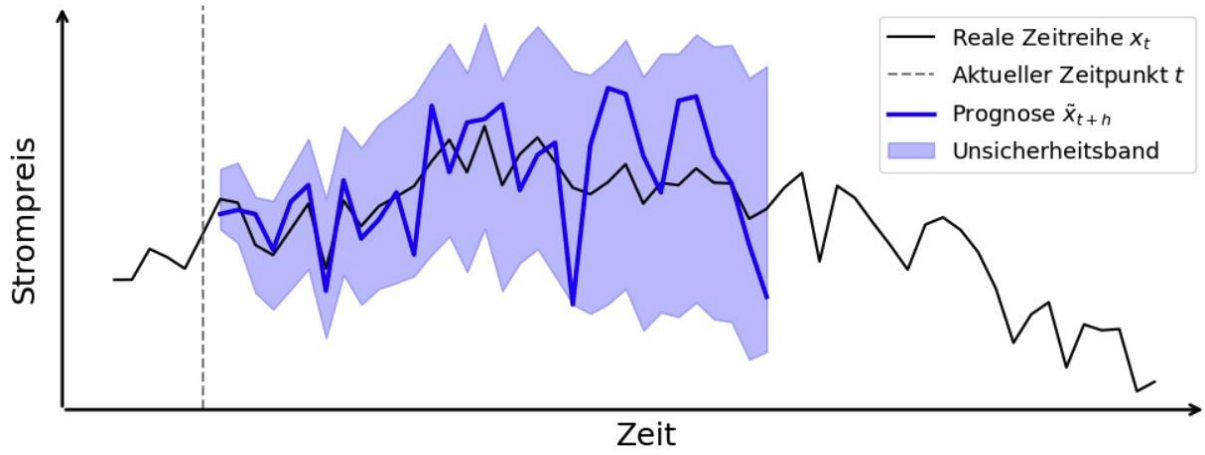

Abb. 3.6: Horizontabhängige Prognoseunsicherheit

Die Breite des Unsicherheitsbandes ist proportional zur horizontabhängigen Standardabweichung σ_h und nimmt mit wachsendem Prognosehorizont zu. Dadurch wird die realitätsnahe Zunahme der Prognoseunsicherheit bei längerfristigen Vorhersagen abgebildet. Während kurzfristige Prognosen eine höhere Genauigkeit aufweisen, steigt die Unsicherheit mit zunehmender zeitlicher Entfernung vom aktuellen Zeitpunkt signifikant an.

Zur Abbildung realistischer Prognosefehler wird ein stochastisches Unsicherheitsmodell für Preis- und Lastprognosen implementiert. Die Prognoseunsicherheit wird als horizontabhängiger, gaußverteilter Fehler $\sigma_h \varepsilon_{t,h}$ modelliert, wobei $\varepsilon_{t,h} \sim \mathcal{N}(0,1)$ ist. Die Varianz des Fehlers wächst mit dem

Prognosehorizont, sodass weiter in der Zukunft liegende Prognosewerte eine größere Streuung aufweisen als kurzfristige Vorhersagen. Auf diese Weise wird eine konsistente und kontrollierbare Abbildung unterschiedlicher Prognosegüten innerhalb eines Forecast-Horizonts ermöglicht. Für eine gegebene Größe x_t (Strompreis oder Last) wird der prognostizierte Wert $\tilde{x}_{t+h}$ für den Horizont $h \in \{1, \dots, H\}$ wie folgt definiert:

$$\tilde{x}_{t+h} = x_{t+h} + \sigma_h \, \varepsilon_{t,h} \tag{3.1}$$

mit

$x_{t+h} =$ Tatsächlicher Wert (Strompreis oder Last) zum Zeitpunkt $t + h$
$\tilde{x}_{t+h} =$ Prognostizierter Wert zum Zeitpunkt $t + h$
$\varepsilon_{t,h} =$ Standardnormalverteilter Fehlerterm
$\sigma_h =$ Horizon abhängige Standardabweichung des Prognosefehlers
$h =$ Prognosehorizont

Die Standardabweichung σ_h steigt mit zunehmendem Prognosehorizont an, um die empirisch beobachtete Zunahme der Prognoseunsicherheit bei längerfristigen Vorhersagen zu berücksichtigen. In dieser Arbeit wird eine monotone Anstiegsfunktion verwendet, beispielsweise eine lineare oder eine wurzelbasierte Skalierung:

$$\sigma_h = \sigma_0 + (\sigma_H - \sigma_0) \cdot f \cdot \left(\frac{h}{H}\right) \tag{3.2}$$

mit

$\sigma_0 =$ Standardabweichung für $h = 1$ (kurzer Horizont)
$\sigma_H =$ Standardabweichung für $h = H$ (längster Horizont)
$f(\cdot) =$ Monotone Skalierungsfunktion (z.B. linear oder $\sqrt{\cdot}$)
$H =$ Maximaler Prognosehorizont
$h =$ Aktueller Prognosehorizont

Die horizontabhängige Unsicherheitsmodellierung ermöglicht eine realitätsnahe Simulation unterschiedlicher Prognosegüten über verschiedene Zeitskalen hinweg. Kurzfristige Vorhersagen weisen dabei eine geringere Streuung auf, während die Unsicherheit mit zunehmendem zeitlichem Abstand vom aktuellen Zeitpunkt systematisch ansteigt. Auf diese Weise wird die empirisch beobachtete Abnahme der Prognosegenauigkeit bei längerfristigen Vorhersagen konsistent abgebildet.

Darüber hinaus schafft dieses Modell die Voraussetzung dafür, dass der RL-Regler Entscheidungen unter expliziter Berücksichtigung variierender Unsicherheitsniveaus treffen kann. Die Entscheidungsfindung erfolgt somit nicht auf Basis deterministischer Zukunftsinformationen, sondern unter realitätsnahen Annahmen begrenzter Prognosequalität. Die vollständige Implementierung des Prognosegenerators ist in Anhang A1 dokumentiert.

3.3 Modellierung des BESS

Für die simulationsbasierte Untersuchung wird ein physikalisch konsistentes Batteriemodell implementiert, das die wesentlichen technischen Eigenschaften eines netzgekoppelten BESS abbildet. Ziel ist es, die energetischen Zustandsgrößen sowie alterungsbedingte Effekte realitätsnah zu modellieren und gleichzeitig eine numerisch stabile Einbindung in die RL-Umgebung zu gewährleisten.

Das Modell berücksichtigt insbesondere die Dynamik des SoC, den SoH sowie technische Leistungs- und Kapazitätsgrenzen. Dadurch wird sichergestellt, dass alle vom RL-Regler getroffenen Entscheidungen physikalisch zulässig sind und realistischen Betriebsbedingungen entsprechen.

3.3.1 Technische Parameter

Die Modellierung des BESS basiert auf fest definierten technischen Kenngrößen, die die physikalischen Eigenschaften und energetischen Grenzen des Systems beschreiben. Die Nennkapazität C_{nom} bestimmt die maximale Speicherkapazität des Systems und legt damit die insgesamt verfügbare Energiemenge fest. Die maximale Leistung P_{max} begrenzt die zulässige Lade- und Entladeleistung pro Zeitschritt und definiert somit die dynamischen Leistungsgrenzen des Speichers.

Die Wirkungsgrade η_{Laden} und $\eta_{Entladen}$ berücksichtigen Umwandlungsverluste während der Energieübertragung und führen dazu, dass nicht die gesamte zu- oder abgeführte elektrische Energie im Speicher wirksam wird. Dadurch wird ein realitätsnahes Betriebsverhalten des Systems abgebildet. Die in Tab. 3.1 aufgeführten Parameter legen somit die physikalischen Rahmenbedingungen fest, innerhalb derer der RL-Regler Entscheidungen treffen kann, und stellen sicher, dass alle Aktionen technisch zulässig bleiben.

Tab. 3.1: Technische Parameter des BESS

Parameter	**Symbol**	**Wert**	**Einheit**	**Beschreibung**
Nennkapazität	C_{nom}	50	kWh	Maximale speicherbare Energie
Max. Leistung	P_{max}	10	kW	Maximale Lade- und Entladeleistung
Lade-Wirkungsgrad	η_{Laden}	0.95	-	Wirkungsgrad beim Laden
Entlade-Wirkungsgrad	$\eta_{Entladen}$	0.95	-	Wirkungsgrad beim Entladen
SoC-Minimum	SoC_{min}	0	-	Untere Betriebsgrenze
SoC-Maximum	SoC_{max}	1	-	Obere Betriebsgrenze

3.3.2 SoC-Dynamik

Die SoC-Dynamik beschreibt die zeitliche Entwicklung des Ladezustands in Abhängigkeit von der gewählten Lade- oder Entladeleistung. Der SoC wird diskret über die Simulationszeitschritte aktualisiert und berücksichtigt dabei Lade- und Entladeverluste über entsprechende Wirkungsgrade.

$$SoC_{t+1} = \begin{cases} SoC_t + \dfrac{P_t \cdot \Delta t \cdot \eta_{Laden}}{C_{nom}} & , P_t > 0 \\ SoC_t + \dfrac{P_t \cdot \Delta t}{C_{nom} \cdot \eta_{Entladen}} & , P_t < 0 \end{cases} \tag{3.3}$$

mit

SoC_t = Ladezustand der Batterie zum Zeitpunkt t
SoC_{t+1} = Ladezustand im nächsten Zeitschritt
P_t = Lade- bzw. Entladeleistung zum Zeitpunkt t
P_{max} = Maximal zulässige Lade- bzw. Entladeleistung
Δt = Länge eines Zeitschritts
C_{nom} = Energiekapazität
η_{Laden} = Wirkungsgrad beim Laden
$\eta_{Entladen}$ = Wirkungsgrad beim Entladen

Der SoC wird diskret über die Energiebilanz aktualisiert. Beim Laden wird die zugeführte Energie mit dem Lade-Wirkungsgrad multipliziert, während beim Entladen die abgegebene Energie unter Berücksichtigung des Entlade-Wirkungsgrads reduziert wird. Zusätzlich wird der SoC auf die definierten Hard-Grenzen SoC_{min} und SoC_{max} begrenzt. Die vollständige Implementierung der SoC-Update-Logik ist in Anhang A2 dokumentiert.

3.3.3 SoH-Dynamik

Neben der kurzfristigen Energiedynamik wird auch die langfristige Alterung des Speichers modelliert. Die SoH-Dynamik basiert auf der Annahme, dass Batteriealterung primär durch Lade- und Entladezyklen verursacht wird. Hierzu wird die Zunahme von EFC erfasst und über einen Degradationskoeffizienten in eine Reduktion des SoH überführt. Zur Vereinfachung der Simulation wird ein lineares Alterungsmodell verwendet, bei dem der SoH proportional zur akkumulierten Zyklenbelastung abnimmt.

Die Degradationskosten c_{deg} beschreiben die monetären Kosten pro EFC und ermöglichen die Berücksichtigung von Alterungseffekten in der Belohnungsfunktion.

Der Parameter k_{deg} gibt den relativen Verlust des SoH pro EFC an und modelliert somit die zyklusbedingte Kapazitätsabnahme des BESS. Beide Parameter bestimmen maßgeblich den langfristigen Einfluss der Nutzung auf die wirtschaftliche und technische Lebensdauer des Speichers. Die verwendeten alterungsrelevanten Parameter sind in Tab. 3.2 zusammengefasst.

Tab. 3.2: Alterungsparameter

Parameter	**Symbol**	**Wert**	**Einheit**	**Beschreibung**
Degradationskosten	c_{deg}	0.05	EUR/EFC	Kosten pro EFC
SoH-Verlust pro EFC	k_{deg}	0.005	-	Relative SoH-Reduktion pro EFC

Der SoH wird diskret über die Simulationszeitschritte aktualisiert und berücksichtigt dabei alterungsbedingte Effekte infolge von Lade- und Entladezyklen. Die Reduktion des Gesundheitszustands erfolgt proportional zur im jeweiligen Zeitschritt akkumulierten EFC (vgl. Gleichung 2.4).

$$SoH_{t+1} = max\left(0, SoH_t - k_{deg} \cdot \overbrace{\frac{|SoC_{t+1} - SoC_t|}{2}}^{\text{EFC}}\right) \tag{3.4}$$

mit

SoH_t = Gesundheitszustand der Batterie zum Zeitpunkt t
SoH_{t+1} = Gesundheitszustand der Batterie im nächsten Zeitschritt
SoC_t = Ladezustand der Batterie zum Zeitpunkt t
SoC_{t+1} = Ladezustand im nächsten Zeitschritt
k_{deg} = Degradationskoeffizient pro EFC

Durch die Verwendung der Max-Funktion wird sichergestellt, dass der SoH nicht unter eine definierte Mindestgrenze fällt. Das Modell stellt eine vereinfachte lineare Alterungsapproximation dar und dient der numerisch stabilen Integration degradationsbedingter Effekte in die Simulationsumgebung. Die vollständige Implementierung der SoH-Aktualisierung ist in Anhang A3 dokumentiert.

3.3.4 Simulationsparameter

Die Simulationsparameter definieren die zeitlichen und szenariospezifischen Rahmenbedingungen der RL-Umgebung. Der Zeitschritt Δt beträgt 0.25 Stunden, was einer zeitlichen Auflösung von 15 Minuten entspricht. Die Episodenlänge beträgt 7 Tage. Jede Trainingsepisode repräsentiert somit eine einwöchige Betriebsphase des BESS.

Der Prognosehorizont ist auf 3 Stunden festgelegt. Bei einer Zeitschrittweite von 0.25 Stunden ergibt sich daraus ein Prognosefenster von 12 zukünftigen Zeitschritten. Zur Vereinfachung der Simulation und zur Reduzierung der Trainingszeit wurde bewusst ein moderater Prognosehorizont gewählt.

Der Parameter Netzexport definiert das betrachtete Betriebsszenario. Ist der Netzexport zugelassen, kann der Speicher Energie in das Netz einspeisen (Arbitrage-Modus). Ist der Netzexport deaktiviert, erfolgt ausschließlich eine Versorgung der Last ohne Netzeinspeisung (Peak-Shaving-Modus). Die verwendeten alterungsrelevanten Parameter sind in Tab. 3.3 zusammengefasst.

Tab. 3.3: Simulationsparameter

Parameter	**Wert**	**Einheit**	**Beschreibung**
Zeitschritt Δt	0.25	h	Simulationsauflösung (15 min)
Episodenlänge	7	Tage	Dauer einer Trainings-Episode
Prognosehorizont	3	h	Länge des Forecast-Fensters
Netzexport	True/False	-	Arbitrage- oder Peak-Shaving-Modus

Für die Simulation und Bewertung der Regelungsansätze werden die RL-Agenten szenariospezifisch mit historischen Zeitreihendaten trainiert. Im Arbitrage-Szenario erfolgt das Training mit den Strompreisdaten des vollständigen Monats November 2024 (siehe Abb. 3.3). Als Testdatensatz dienen die Strompreisdaten der ersten Woche im November 2025, um die Fähigkeit zur Generalisierung auf unbekannte Marktbedingungen zu evaluieren.

Im Peak-Shaving-Szenario werden für das Training sowohl Strompreis- als auch Lastdaten des November 2024 verwendet (siehe Abb. 3.3 und Abb. 3.5). Die Testdaten bestehen aus Strompreis- und Lastprofilen der ersten Woche im Jahr 2025.

3.4 Implementierung der RL-Umgebung

Zur Abbildung der Entscheidungslogik wird das BESS in eine RL-Umgebung integriert. Die Umgebung beschreibt die Interaktion zwischen Agent und Energiesystem als diskreten Markov-Entscheidungsprozess. In jedem Zeitschritt beobachtet der Agent den aktuellen Systemzustand, wählt eine Lade- oder Entladeaktion und erhält eine Belohnung in Abhängigkeit vom ökonomischen Ergebnis sowie möglichen Degradations- oder Strafkosten.

3.4.1 Zustandsraum

Im RL beschreibt der Zustand s_t die zum Entscheidungszeitpunkt t verfügbare Informationsmenge über System und Umgebung [4]. Der Zustandsraum umfasst die Größen, die der RL-Regler zur Bewertung der aktuellen Situation und zur Auswahl einer geeigneten Aktion benötigt.

Aufgrund eines Prognosehorizonts von 3 h und einer zeitlichen Auflösung von 15 min (0.25 h) ergeben sich pro Prognosefenster

$$H = \frac{3}{0.25} = 12$$

Zeitschritte.

Damit werden jeweils 12 zukünftige Werte für Strompreis sowie, sofern verfügbar, für die Last in den Zustandsvektor integriert. Im Peak-Shaving-Szenario umfasst der Zustandsraum somit:

- 1 SoC
- 1 SoH
- 1 letzte Aktion
- 1 aktueller Strompreis
- 1 aktuelle Last
- 4 zeitliche Merkmale (Sinus/Kosinus für Tages- und Jahresperiode)
- 12 Preisprognosewerte
- 12 Lastprognosewerte

Insgesamt ergibt sich damit eine Zustandsdimension von

$$1 + 1 + 1 + 1 + 1 + 4 + 12 + 12 = 33$$

Zustandsgrößen.

Im Arbitrage-Szenario stehen hingegen keine Lastdaten und keine Lastprognosen zur Verfügung. Der Zustandsraum reduziert sich daher auf

$$1 + 1 + 1 + 1 + 4 + 12 = 20$$

Dimensionen.

Durch die Einbindung von Zeitmerkmalen und Prognosedaten wird dem Agenten ermöglicht, periodische Muster sowie zukünftige Entwicklungen in die Entscheidungsfindung einzubeziehen. Die Zustandsgrößen werden normiert, um numerische Stabilität während des Trainings sicherzustellen. Ein exemplarischer Codeausschnitt zur Implementierung des Zustandsraums ist in Anhang A4 dokumentiert.

3.4.2 Aktionsraum

Der Aktionsraum beschreibt die vom Agenten steuerbare Lade- bzw. Entladeleistung des BESS. Je nach verwendetem Algorithmus wird zwischen einem kontinuierlichen und einem diskreten Aktionsraum unterschieden.

Kontinuierliche Aktionen

Im kontinuierlichen Modus, der für den Algorithmus TD3 verwendet wird, kann die Leistung innerhalb der physikalischen Grenzen

$$a_t \in [-P_{max}, +P_{max}] \text{ mit } P_{max} = 10 \text{ kW}$$

frei gewählt werden. Der Agent bestimmt somit eine reellwertige Lade- oder Entladeleistung, wodurch eine fein abgestufte Regelung des BESS ermöglicht wird.

Diskrete Aktionen

Im diskreten Modus, der für DQN und QR-DQN eingesetzt wird, steht eine endliche Anzahl vordefinierter Leistungsstufen zur Verfügung. In dieser Arbeit umfasst der diskrete Aktionsraum insgesamt 21 äquidistante Aktionen im Bereich

$$a_t \in \{-10, -9, \dots, 0, \dots, +9, +10\} \text{ kW}.$$

Negative Werte entsprechen einer Entladeleistung, positive Werte einer Ladeleistung. Die Diskretisierung vereinfacht die Q-Wert-Schätzung bei wertbasierten Verfahren, reduziert jedoch die Regelauflösung im Vergleich zum kontinuierlichen Ansatz. Die Umsetzung der hier beschriebenen Aktionsraumdefinition ist im Codeausschnitt in Anhang A5 dargestellt.

3.4.3 Belohnungsunktion

Die Belohnungsfunktion basiert primär auf dem ökonomischen Ergebnis des jeweiligen Zeitschritts. Ziel ist es, wirtschaftlich vorteilhafte Betriebsstrategien zu fördern und gleichzeitig degradationsbedingte Kosten sowie gegebenenfalls Netzrestriktionen zu berücksichtigen. Die allgemeine Struktur der Belohnungsfunktion lautet:

$$r_t = R_t - C_t^{deg} - S_t^{cap} \tag{3.5}$$

mit

r_t = Belohnung zum Zeitpunkt t
R_t = Erlös bzw. Kosten aus Netzbezug oder Netzeinspeisung
C_t^{deg} = Degradationskosten infolge der Batterienutzung
S_t^{cap} = Peak-Shaving-Strafterm

Die konkrete Ausgestaltung der Belohnungsfunktion ist szenarioabhängig und wird in den folgenden Absätzen getrennt für das Arbitrage- und das Peak-Shaving-Szenario beschrieben.

Arbitrage-Szenario

Im Arbitrage-Szenario besteht die Belohnung ausschließlich aus dem wirtschaftlichen Energieaustausch mit dem Netz sowie den Degradationskosten. Ein zusätzlicher Strafterm ist hier nicht enthalten, da keine Importbegrenzung berücksichtigt wird. Für einen Zeitschritt gilt damit:

$$r_t = \overbrace{-p_t \cdot E_t}^{R_t} \overbrace{- c_{deg} \cdot \frac{|SoC_{t+1} - SoC_t|}{2}}^{C_t^{deg}} \tag{3.6}$$
$$E_t = a_t \cdot \Delta t$$

mit

r_t = Belohnung zum Zeitpunkt t
p_t = Strompreis zum Zeitpunkt t
E_t = Tatsächlich umgesetzte Energie mit dem Netz
a_t = Aktion des RL-Agenten
Δt = Zeitschritt (hier: 0.25 h)
c_{deg} = Degradationskosten pro EFC
SoC_t = Ladezustand zum Zeitpunkt t
SoC_{t+1} = Ladezustand im nächsten Zeitschritt

Der Agent entscheidet über eine Lade- bzw. Entladeleistung, wobei positive Aktionen einem Netzbezug (Kaufen) und negative Aktionen einer Netzeinspeisung (Verkauf) entsprechen. Aus der tatsächlich umgesetzten Energie E_t(unter Berücksichtigung von Leistungs- und SoC-Grenzen sowie Wirkungsgraden) wird der monetäre Beitrag berechnet. Dabei führt Laden ($E_t > 0$) zu Kosten und Entladen ($E_t < 0$) zu Erlösen. Die physikalischen Randbedingungen entsprechen der in Unterkapitel 3.1.1 dargestellten Systemstruktur.

Peak-Shaving-Szenario

Im Peak-Shaving-Szenario wird die allgemeine Belohnungsfunktion um einen quadratischen Strafterm erweitert, der eine Überschreitung einer definierten Netzimportgrenze sanktioniert. Dadurch wird der Agent gezielt dazu angehalten, Lastspitzen durch den Einsatz des Speichers zu reduzieren. Da im Peak-Shaving-Szenario keine Netzeinspeisung zulässig ist, dient eine Entladung ausschließlich der Lastversorgung, während der verbleibende Anteil der Last als Restlast vom Netz bezogen wird. Zusätzlich kann das BESS in lastarmen Phasen aus dem Netz geladen werden.

Die Belohnung setzt sich aus den Energiekosten für den Netzbezug, den Degradationskosten sowie einer Zusatzstrafe zur gezielten Lastspitzenreduktion zusammen:

$$r_t = \overbrace{-p_t \cdot E_t^{Grid}}^{R_t} \overbrace{- c_{deg} \cdot \frac{|SoC_{t+1} - SoC_t|}{2}}^{C_t^{deg}} \overbrace{- k \cdot \left(max\left(0, E_t^{Import} - E_t^{cap}\right)\right)^2}^{S_t^{cap}} \tag{3.7}$$

$$E_t^{Grid} = E_t^{Import} + E_t^{Charge}$$

mit

r_t = Belohnung zum Zeitpunkt t
p_t = Strompreis zum Zeitpunkt t
E_t^{Grid} = Gesamter Netzbezug
E_t^{Import} = Netzbezug zur Deckung der Last
E_t^{Charge} = Netzenergie zum BESS-Laden
E_t^{cap} = Importgrenze
c_{deg} = Degradationskosten pro EFC
SoC_t = Ladezustand zum Zeitpunkt t
SoC_{t+1} = Ladezustand im nächsten Zeitschritt
k = Strafkoeffizient

Hierbei beschreibt E_t^{Grid} die gesamte im Zeitschritt aus dem Netz bezogene Energie, bestehend aus dem Restlastbezug E_t^{Import} sowie der zum Laden des BESS verwendeten Energie E_t^{Charge}.

Die Aktionsvariable a_t bestimmt dabei die Energierichtung des BESS. Für $a_t > 0$ wird Energie aus dem Netz bezogen und zum Laden des BESS verwendet, sodass $E_t^{Charge} > 0$ gilt. Für $a_t < 0$ wird Energie aus dem BESS zur Deckung der Last bereitgestellt. In diesem Fall reduziert sich der Netzimport E_t^{Import}, da ein Teil der Last direkt durch das BESS gedeckt wird. Die physikalischen Randbedingungen entsprechen der in Unterkapitel 3.1.2 dargestellten Systemstruktur.

Im Modell wird keine explizite Leistungsgrenze für das Stromnetz angenommen. Die Leistungsbeschränkungen gelten ausschließlich für das BESS, insbesondere hinsichtlich maximaler Lade- und Entladeleistung sowie der SoC-Grenzen. Das Netz kann somit beliebige Energiemengen liefern, während das BESS physikalisch limitiert ist.

In lastarmen Phasen oder bei niedrigen Strompreisen kann der Agent entscheiden, den Speicher aus dem Netz zu laden, während die Last weiterhin vollständig aus dem Netz gedeckt wird. In diesem Fall treten E_t^{Charge} und E_t^{Import} parallel auf.

In Lastspitzenphasen oder bei hohen Strompreisen kann der Agent hingegen gespeicherte Energie zur Lastdeckung einsetzen. Dadurch wird E_t^{Import} reduziert. Während einer aktiven Entladung zur Lastversorgung erfolgt kein gleichzeitiges Laden des Speichers, sodass $E_t^{Charge} = 0$ gilt. Ziel ist es, den Netzimport möglichst unter die definierte Importgrenze E_t^{cap} zu senken, um Strafkosten zu vermeiden.

Die Netzimportgrenze E_t^{cap} wird im Peak-Shaving-Szenario auf Basis historischer Lastdaten bestimmt. Dazu werden Quantile der Lastverteilung verwendet. In lastarmen Zeiträumen entspricht die Grenze dem 60%-Quantil, während in lastintensiven Zeiträumen eine strengere Grenze auf Basis des 30%-Quantils definiert wird. Dadurch wird das BESS gezielt zur Reduktion von Lastspitzen eingesetzt.

Der quadratische Strafterm wird aktiviert, sobald der tatsächliche Netzimport E_t^{Import} die definierte Grenze E_t^{cap} überschreitet. In Lastspitzenphasen entsteht somit ein starker Anreiz für den Agenten, gespeicherte Energie (sofern verfügbar) zur Deckung der Last einzusetzen. Durch eine geeignete Entladeaktion kann der Netzimport reduziert und idealerweise unter die Importgrenze abgesenkt werden. Ziel des Agenten ist es daher, die Differenz

$$E_t^{Import} - E_t^{cap}$$

möglichst klein zu halten, um den Strafterm zu minimieren und die Gesamtbelohnung zu maximieren. Die Implementierung der Belohnungsfunktion ist in Anhang A6 dokumentiert.

4 Regelungsansätze

In diesem Kapitel werden vier verschiedene Regelungsansätze zur Steuerung des BESS vorgestellt und miteinander verglichen. Neben einem regelbasierten Referenzcontroller werden drei in Kapitel 2.1 vorgestellte RL-Agenten untersucht. Der Fokus liegt auf der Analyse des Entscheidungsverhaltens im Arbitrage- und im Peak-Shaving-Modus. Dabei wird untersucht, wie die jeweiligen Ansätze Lade- und Entladeentscheidungen treffen und wie sich die Strategien in den beiden Betriebsmodi unterscheiden.

4.1 Rule-Based Controller

Der regelbasierte Controller dient in dieser Arbeit als Referenzstrategie und ermöglicht einen Vergleich mit den eingesetzten RL-Agenten. Somit bildet dieser Ansatz eine Grundlage zur Bewertung der lernbasierten Verfahren. Im Gegensatz zu den RL-Ansätzen basiert die Steuerungslogik auf fest definierten Entscheidungsregeln.

4.1.1 Entscheidungsregel

Der regelbasierte Controller basiert auf einer preisorientierten Entscheidungslogik. Zur Definition von günstigen und teuren Preisbereichen werden aus historischen Preisdaten zwei Perzentilschwellen bestimmt, typischerweise das 20%-Perzentil p_{low}und das 80%-Perzentil p_{high}. Der aktuelle Strompreis im Zeitschritt t, bezeichnet als p_t, wird relativ zu diesen beiden Schwellen eingeordnet und kontinuierlich auf einen Steuerfaktor abgebildet. Die resultierende Lade- bzw. Entladeleistung ergibt sich aus:

$$a_t = \left(1 - 2 \cdot clip\left(\frac{p_t - p_{low}}{p_{high} - p_{low}}, 0{,}1\right)\right) \cdot P_{max} \tag{4.1}$$

mit

$a_t =$ vom Controller gewählte Lade- bzw. Entladeleistung im Zeitschritt t

$p_t =$ aktueller Strompreis im Zeitschritt t

$p_{low} =$ unteres Perzentil der historischen Preisverteilung

$p_{high} =$ oberes Perzentil der historischen Preisverteilung

$clip(x, 0, 1) =$ Begrenzung von x auf $[0{,}1]$

Im Gegensatz zu einer binären Schwellenregel (Laden oder Entladen mit fixer Leistung) erfolgt die Leistungsanpassung proportional zur Preisabweichung. Liegt der aktuelle Strompreis p_t nahe oder unterhalb von p_{low} (untere 20% der historischen Preisverteilung), wird eine hohe Ladeleistung angesteuert. Befindet sich $p_t$hingegen im Bereich von p_{high} (obere 20% der historischen Preisverteilung), wird die Entladeleistung entsprechend erhöht. Für Preiswerte zwischen p_{low}und p_{high}ergibt sich eine lineare Skalierung der Leistung. Dadurch entsteht ein stetiges

Steuerverhalten zwischen maximaler Lade- und maximaler Entladeleistung. Theoretisch ist die Aktion a_t genau im Mittelpunkt des Intervalls $[p_{low}, p_{high}]$ gleich null. In der praktischen Simulation kann Inaktivität jedoch zusätzlich durch Leistungsbegrenzungen an den SoC-Grenzen oder durch sehr kleine Leistungswerte entstehen. In Anhang A.7 ist ein exemplarischer Auszug der Implementierung des Rule-Based Controllers enthalten.

Aus Gründen der Fokussierung auf die Entwicklung der Simulationsumgebung und die Analyse der RL-Regler wird der regelbasierte Controller in dieser Arbeit ausschließlich preisbasiert implementiert. Ziel ist es, eine nachvollziehbare und deterministische Referenzstrategie bereitzustellen, die als Baseline für den Vergleich mit den lernbasierten Verfahren dient.

4.1.2 Arbitrage-Verhalten

Der regelbasierte Controller folgt einer festen Schwellenwertstrategie (siehe Unterkapitel 4.1.1). Unterschreitet der Strompreis einen definierten unteren Grenzwert, wird der Speicher geladen. Überschreitet der Strompreis hingegen einen oberen Schwellenwert, erfolgt eine Entladung. Für Preiswerte im Intervall p_{low} und p_{high} wird die Leistung proportional zur normierten Preisabweichung linear skaliert. Abb. 4.1 zeigt das Arbitrage-Verhalten des Rule-Based Controllers in der Evaluierungsphase über einen Zeitraum von einer Woche.

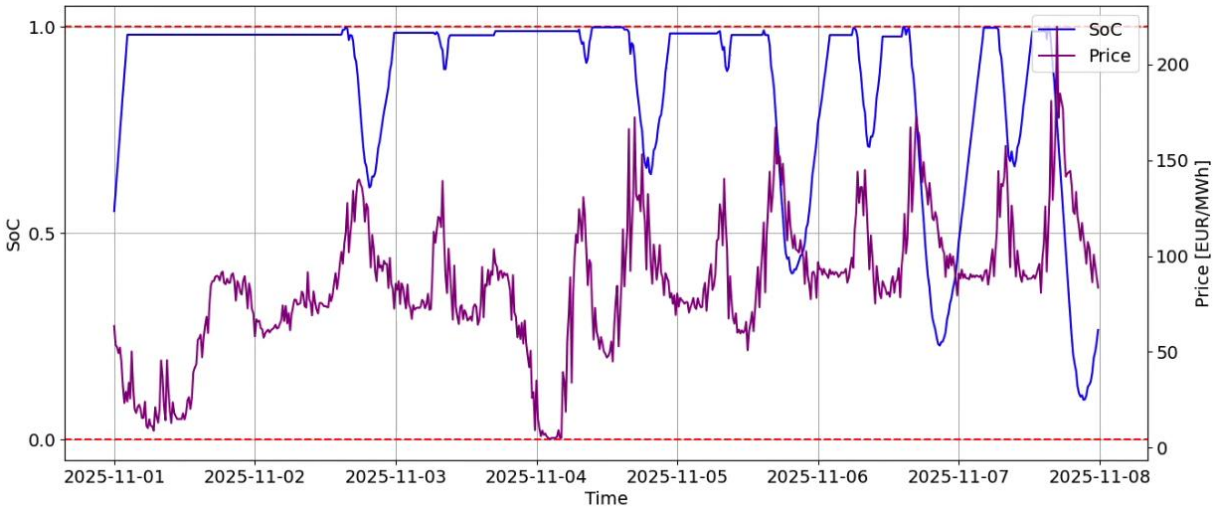

Abb. 4.1: Arbitrage-Verhalten des Rule-Based Controllers im Testzeitraum

Während längerer Niedrigpreisphasen steigt der SoC kontinuierlich an, wohingegen in Hochpreisphasen eine gezielte Entladung erfolgt. Im Gegensatz zu lernbasierten Ansätzen basiert die Entscheidungsfindung ausschließlich auf dem aktuellen Preisniveau. Zukünftige Preisentwicklungen oder Unsicherheiten im Markt werden nicht berücksichtigt. Kurzfristige oder moderate Preisschwankungen werden nur dann genutzt, wenn sie die definierten Grenzwerte erreichen. Preisbewegungen innerhalb

des Schwellenbands bleiben hingegen ungenutzt, auch wenn sie potenziell wirtschaftlich vorteilhaft wären.

4.1.3 Peak-Shaving-Verhalten

Da in dieser Arbeit keine separate Peak-Shaving-Strategie implementiert wird, basiert die Lade- und Entladeentscheidung weiterhin auf der in Abschnitt 4.1.1 beschriebenen preisorientierten Regel. Dies bedeutet, dass der BESS bei niedrigen Strompreisen geladen und bei hohen Preisen entladen wird. Im Gegensatz zum Arbitrage-Betrieb erfolgt die Entladung nicht zur Einspeisung in das Netz, sondern zur direkten Deckung der lokalen Last. Eine explizite Berücksichtigung von Lastspitzen findet dabei nicht statt. Aufgrund der identischen preisgetriebenen Entscheidungslogik entspricht die SoC-Dynamik weitgehend dem in Abb. 4.1 dargestellten Arbitrage-Verhalten.

Hohe Strompreise korrelieren in der Praxis häufig mit erhöhten Lastsituationen [3]. Dadurch kann die preisbasierte Entladestrategie indirekt zu einer Reduktion von Lastspitzen führen. Das beobachtbare Peak-Shaving-Verhalten stellt somit kein gezielt optimiertes Steuerungsziel dar, sondern ergibt sich als Nebeneffekt der preisorientierten Regel. Dieses Verhalten ist in Abb. 4.2 dargestellt.

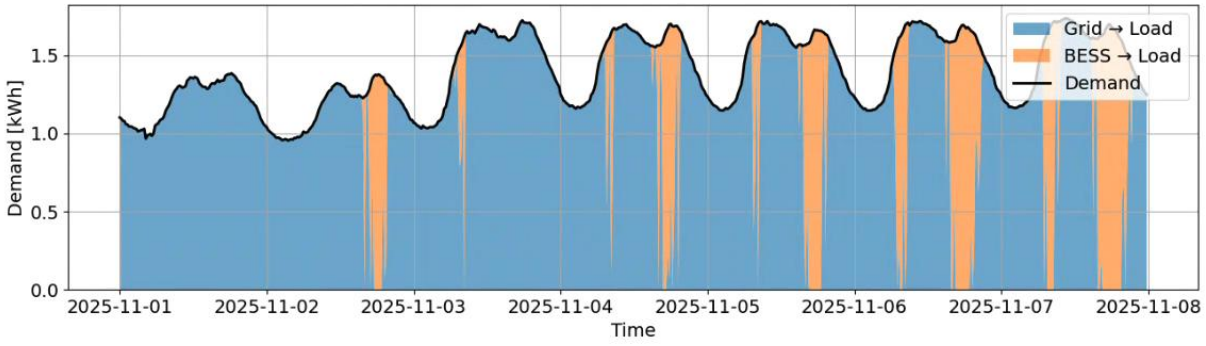

Abb. 4.2: Peak-Shaving-Verhalten des Rule-Based Controllers im Testzeitraum

Die oben dargestellte Lastaufteilung zeigt, dass eine Reduktion einzelner Lastspitzen erfolgt, jedoch keine vollständige oder systematische Glättung der Lastkurve erreicht wird. Der Zusammenhang zwischen Strompreis und Last ist in der Praxis nicht vollständig deterministisch. Zwar treten hohe Preise häufig in Zeiten erhöhter Nachfrage auf, jedoch existieren auch Lastspitzen ohne entsprechend hohe Preisniveaus sowie Preispeaks ohne außergewöhnliche Lastsituationen.

Da keine explizite Peak-Shaving-Strategie implementiert ist, erfolgt die Entladung des BESS ausschließlich preisgetrieben. Eine direkte Optimierung des maximalen Netzbezugs oder eine gezielte Begrenzung von Lastspitzen ist nicht Bestandteil der Regel. Die beobachtete Reduktion einzelner Peaks ergibt sich daher lediglich dann, wenn hohe Preisphasen zeitlich mit hohen Lastanforderungen zusammenfallen. In Fällen fehlender Preis-Last-Korrelation bleibt eine Glättung der Lastspitze aus.

4.2 DQN

Nach der Analyse des regelbasierten Controllers wird nun das Verhalten des DQN-Agenten im Testzeitraum untersucht. Die Auswertung erfolgt getrennt nach Arbitrage- und Peak Shaving, um die Auswirkungen der lernbasierten Entscheidungsstrategie systematisch zu analysieren.

4.2.1 Arbitrage-Verhalten

Im Gegensatz zum regelbasierten Controller basiert die Entscheidungslogik des DQN-Agenten nicht auf festen Preisschwellen, sondern auf einer durch Training erlernten Aktionspolitik (siehe Kapitel 2.1.1). Es ist erkennbar, dass der Agent bei niedrigen Preisphasen überwiegend lädt und bei hohen Preisniveaus entlädt. Dabei folgt der SoC-Verlauf deutlich der Preisstruktur, was auf ein erfolgreich erlerntes Arbitrage-Verhalten hinweist. Abb. 4.3 zeigt das Arbitrage-Verhalten des DQN-Agenten in der Evaluierungsphase über einen Zeitraum von einer Woche.

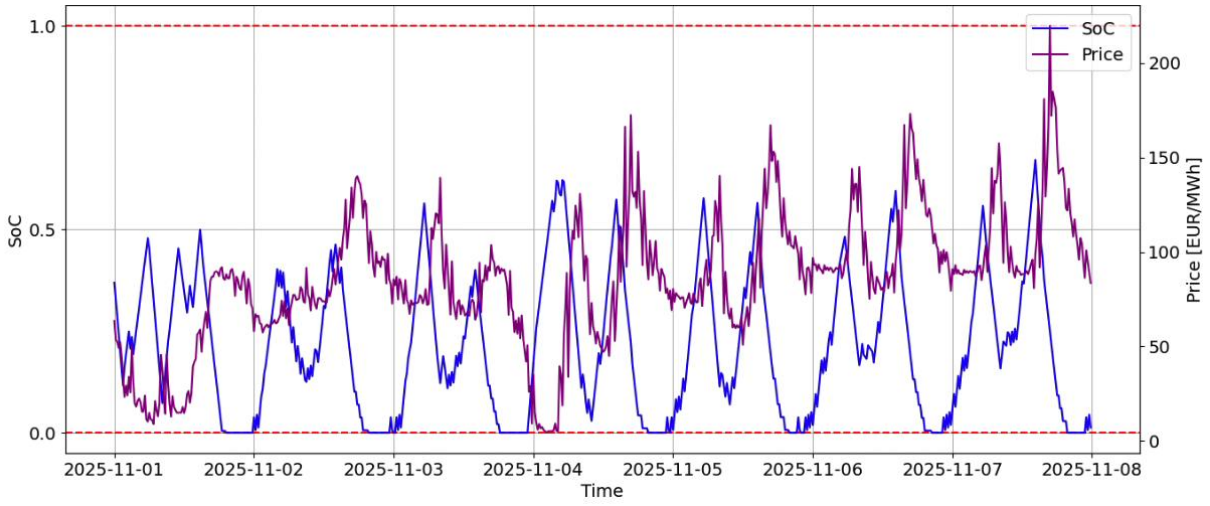

Abb. 4.3: Arbitrage-Verhalten des DQN-Agenten im Testzeitraum

Auffällig ist, dass der Agent nicht ausschließlich bei extremen Preiswerten aktiv wird, sondern auch innerhalb moderater Preisbereiche differenzierte Lade- und Entladeentscheidungen trifft. Dies deutet darauf hin, dass der DQN-Algorithmus Preisdynamiken und zeitliche Muster berücksichtigt, anstatt nur starre Schwellenwerte zu verwenden. Im Vergleich zum regelbasierten Ansatz erscheint das Verhalten flexibler an die tatsächliche Preisentwicklung angepasst.

4.2.2 Peak-Shaving-Verhalten

Wie bereits im Arbitrage-Betrieb basiert auch hier die Lade- und Entladeentscheidung auf der durch Training erlernten Aktionspolitik und nicht auf einer expliziten Last- oder Peak-Shaving-Regel. Abb. 4.4 zeigt den SoC- und Preisverlauf des DQN-Agenten im Peak-Shaving-Betrieb.

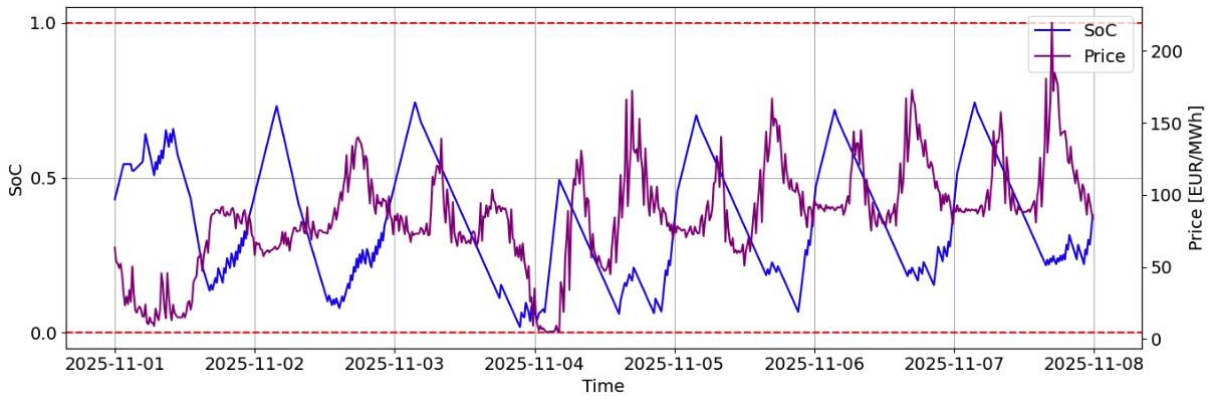

Abb. 4.4: SoC- und Preisverlauf des DQN-Agenten im Peak-Shaving

Aus Abb. 4.4 ist ersichtlich, dass der DQN-Agent grundsätzlich bei hohen Preisphasen überwiegend entlädt und bei niedrigen Preisen lädt. Im Gegensatz zum regelbasierten Controller wirkt das Lade- und Entladeverhalten des DQN-Agenten deutlich flexibler. In Abb. 4.5 wird die resultierende Lastaufteilung zwischen Netzbezug und BESS-Versorgung dargestellt.

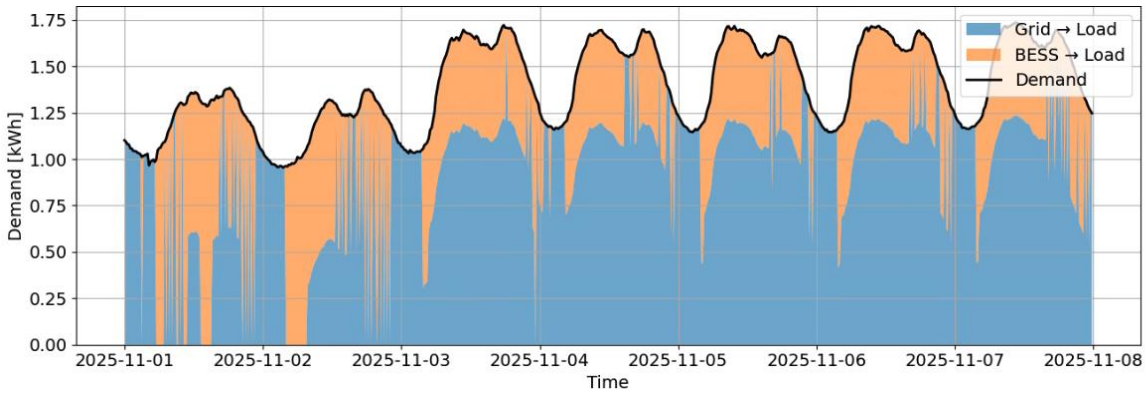

Abb. 4.5: Lastaufteilung des DQN-Agenten im Peak-Shaving

Aus Abb. 4.5 ist ersichtlich, dass der DQN-Agent gespeicherte Energie gezielt zur Deckung der Last einsetzt. Die Ladung des Speichers erfolgt überwiegend in Phasen niedriger Strompreise sowie in lastarmen Zeiträumen, wodurch günstige Marktbedingungen mit geringer Netzbelastung kombiniert werden. In darauffolgenden Hochpreisphasen oder bei auftretenden Lastspitzen wird die gespeicherte Energie genutzt, um einen Teil der Last zu decken und den Netzbezug zu reduzieren. Dieses Verhalten verdeutlicht, dass der DQN-Agent sowohl Preis- als auch Lastinformationen in die Entscheidungsfindung einbezieht. Die Entladung konzentriert sich insbesondere auf Zeiträume erhöhter Nachfrage, wodurch eine partielle Glättung der Netzlast erreicht wird.

4.3 TD3

Im Folgenden wird das Verhalten des TD3-Agenten im Testzeitraum analysiert. Die Untersuchung erfolgt getrennt nach Arbitrage- und Peak-Shaving-Betrieb, um die Charakteristika der kontinuierlichen, lernbasierten Strategie bewerten zu können.

4.3.1 Arbitrage-Verhalten

Der TD3-Agent basiert auf einem kontinuierlichen Aktionsraum und kann somit stufenlose Lade- und Entladeleistungen wählen. Aus der Abb. 4.6 ist ersichtlich, dass der Agent stark auf kurzfristige Preisänderungen reagiert und Lade- sowie Entladeentscheidungen teilweise sehr aggressiv ausführt. Zwar erfolgt eine grundsätzliche Orientierung am Preisniveau, jedoch wird der Unterschied zwischen ausgeprägten Hoch- und Niedrigpreisphasen nicht konsequent ausgenutzt.

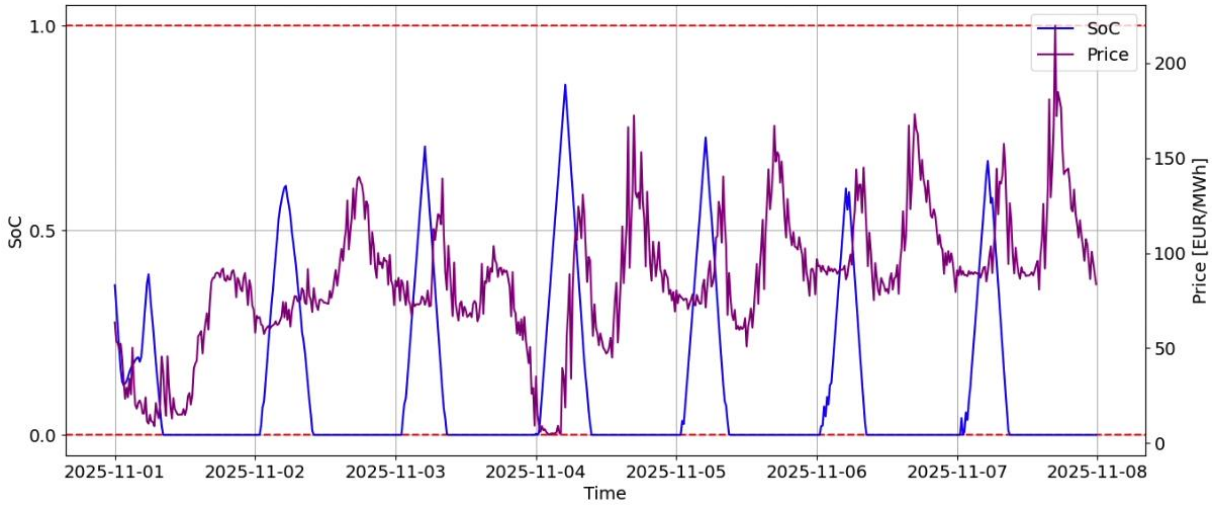

Abb. 4.6: Arbitrage-Verhalten des TD3-Agenten im Testzeitraum

Im Vergleich zum DQN-Ansatz, der lediglich diskrete Leistungsstufen auswählen kann, wirkt das Lade- und Entladeverhalten des TD3-Agenten deutlich glatter und kontinuierlicher. Übergänge zwischen Lade- und Entladephasen erfolgen weniger abrupt, da keine festen Leistungsstufen erzwungen werden. Dadurch erscheint die SoC-Dynamik insgesamt gleichmäßiger und weniger sprunghaft als beim diskreten DQN-Agenten.

4.3.2 Peak-Shaving-Verhalten

Durch den kontinuierlichen Aktionsraum ist es dem TD3-Agenten möglich, die Lade- und Entladeleistung stufenlos zu variieren. In mehreren Zeitabschnitten führt dies dazu, dass innerhalb kurzer Zeit vergleichsweise große Energiemengen aus dem Speicher abgegeben werden. Abb. 4.7 verdeutlicht, dass der TD3-Agent in Phasen hoher Strompreise häufig mit einer ausgeprägten Entladeleistung reagiert.

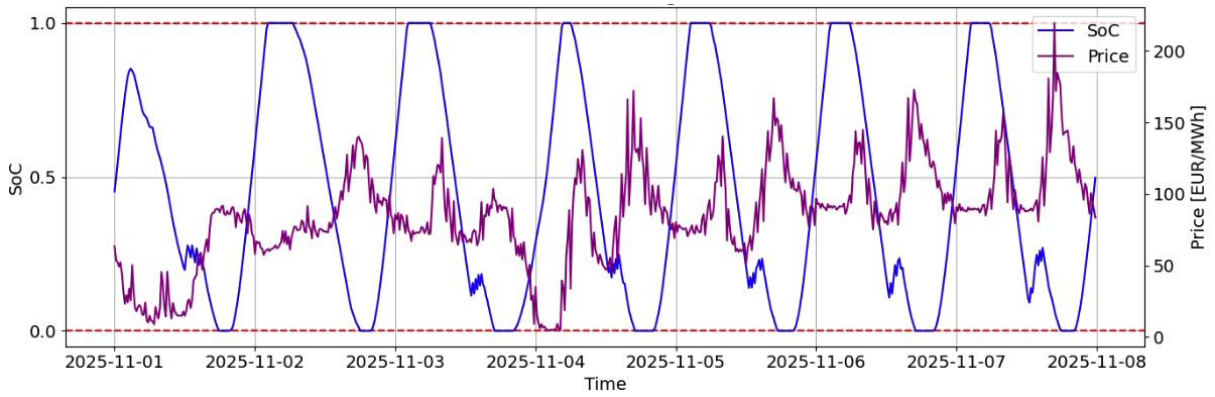

Abb. 4.7: SoC- und Preisverlauf des TD3-Agenten im Peak-Shaving

Wie in Abb. 4.8 erkennbar ist, wird bei einzelnen Lastspitzen ein sehr hoher Anteil der Last durch den Speicher übernommen. Dadurch entlädt sich der Speicher jedoch rasch, sodass gegen Ende der Lastspitze nicht mehr ausreichend Energie zur Verfügung steht. In diesen Phasen steigt der Netzbezug wieder deutlich an, wodurch die Lastspitze nicht über den gesamten Zeitraum hinweg effektiv geglättet wird.

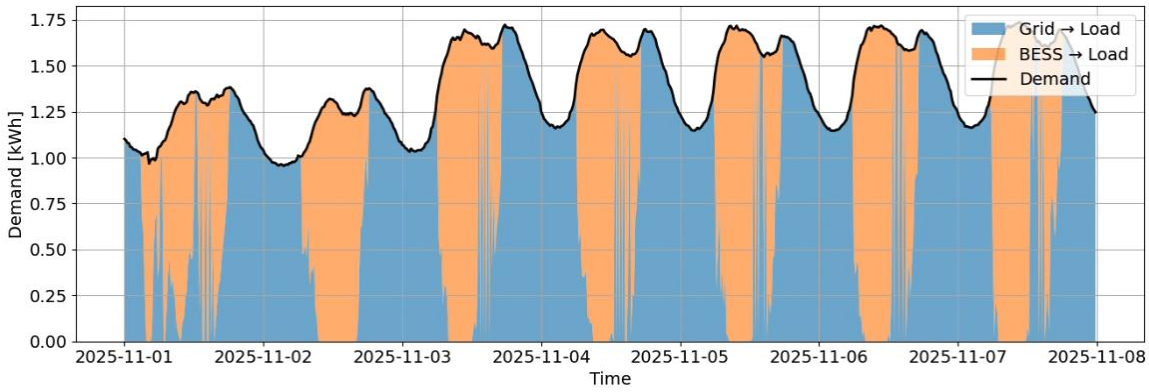

Abb. 4.8: Lastaufteilung des TD3-Agenten im Peak-Shaving

Das Peak-Shaving-Verhalten des TD3-Agenten entspricht daher nicht vollständig den Erwartungen. Zwar wird zu Beginn von Hochlastphasen eine deutliche Reduktion des Netzbezugs erreicht, jedoch erfolgt die Entladung teilweise zu aggressiv, sodass keine nachhaltige Abdeckung der gesamten Lastspitze gewährleistet ist. Dieses Verhalten deutet darauf hin, dass die lernbasierte Strategie primär auf kurzfristige Preisreize reagiert und die zeitliche Struktur der Lastspitzen nicht ausreichend berücksichtigt.

Im Vergleich dazu nähert sich das Peak-Shaving-Verhalten des DQN-Agenten stärker der idealisierten Lastglättung an, wie sie in Abb. 2.5 konzeptionell dargestellt ist. Der DQN-Agent zeigt somit eine gleichmäßigere Energieabgabe über die Dauer der Lastspitze, wodurch eine stabilere und kontinuierlichere Reduktion des Netzbezugs erreicht wird.

4.4 QR-DQN

Im Folgenden wird das Verhalten des QR-DQN-Agenten im Testzeitraum analysiert. Die Untersuchung erfolgt getrennt nach Arbitrage- und Peak-Shaving-Betrieb, um die Eigenschaften der distributionalen Entscheidungsstrategie zu bewerten.

4.4.1 Arbitrage-Verhalten

Der QR-DQN-Agent basiert auf einem distributionalen Lernansatz, bei dem nicht lediglich ein einzelner Erwartungswert der zukünftigen Belohnung approximiert wird, sondern die gesamte Return-Verteilung modelliert wird (siehe Kapitel 2.1.3). Abb. 4.3 zeigt das Arbitrage-Verhalten des DQN-Agenten in der Evaluierungsphase.

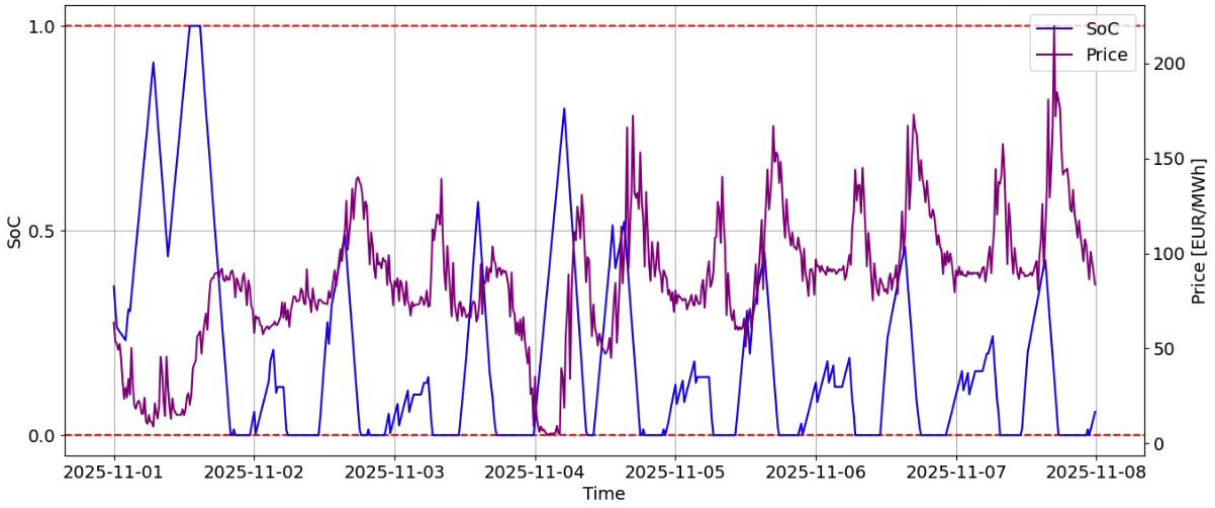

Abb. 4.9: Arbitrage-Verhalten des QR-DQN-Agenten im Testzeitraum

Wie bereits bei DQN und TD3 erkennbar, ist auch beim QR-DQN-Agenten eine deutliche Anpassung des SoC an die Preisstruktur zu beobachten. Der Agent speichert Energie vorwiegend in Phasen niedriger Strompreise und gibt sie bei hohen Preisniveaus wieder ab. Dieses Verhalten verdeutlicht, dass der distributionale Ansatz neben der Berücksichtigung von Unsicherheitsinformationen auch die grundlegende Struktur der Preisentwicklung abbilden kann.

Im Vergleich zum klassischen DQN-Ansatz zeigt der QR-DQN-Agent ein grundsätzlich ähnliches preisgetriebenes Verhalten (vgl. Abb. 4.3). Auch beim QR-DQN-Agenten werden die Lade- und Entladeentscheidungen maßgeblich durch das aktuelle Preisniveau bestimmt. Im Vergleich zum TD3-Agenten reagiert QR-DQN jedoch sensibler auf moderate Preisänderungen (vgl. Abb. 4.6). Insbesondere bei mittleren Preisniveaus sind Anpassungen des SoC erkennbar, was darauf hindeutet, dass nicht ausschließlich ausgeprägte Preisextreme berücksichtigt werden.

4.4.2 Peak-Shaving-Verhalten

Im Gegensatz zum reinen Arbitrage-Betrieb basieren die Lade- und Entladeentscheidungen des QR-DQN-Agenten im Peak-Shaving-Szenario nicht ausschließlich auf dem Preisniveau, sondern berücksichtigen zusätzlich die aktuelle Lastsituation. Da dem Agenten sowohl Preis- als auch Last-Informationen zur Verfügung stehen, erfolgt die Steuerung in Abhängigkeit beider Größen. In Phasen hoher Strompreise und gleichzeitig erhöhter Nachfrage wird verstärkt entladen, um die Lastspitze zu reduzieren, während in preisgünstigen und lastarmen Zeiträumen Energie gespeichert wird. Der SoC-Verlauf zeigt entsprechend eine Anpassung sowohl an die Preisstruktur als auch an die zeitliche Entwicklung der Last (siehe Abb. 4.10).

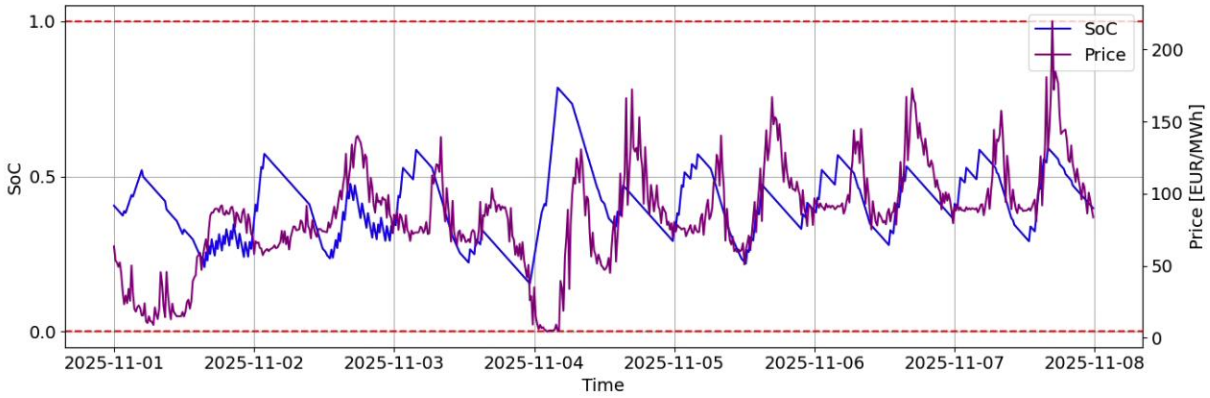

Abb. 4.10: SoC- und Preisverlauf des QR-DQN-Agenten im Peak-Shaving

Im Hinblick auf das Peak-Shaving zeigt sich, dass der QR-DQN-Agent Lastspitzen teilweise reduziert, indem gespeicherte Energie zur Deckung der Last eingesetzt wird. Im Vergleich zum TD3-Agenten erfolgt die Entladung weniger abrupt und gleichmäßiger über die Dauer der Lastspitze verteilt. Gleichzeitig ähnelt das Verhalten in der Grundstruktur dem klassischen DQN-Ansatz, wobei insbesondere bei moderaten Preisänderungen zusätzliche Anpassungen erkennbar sind (siehe Abb. 4.11).

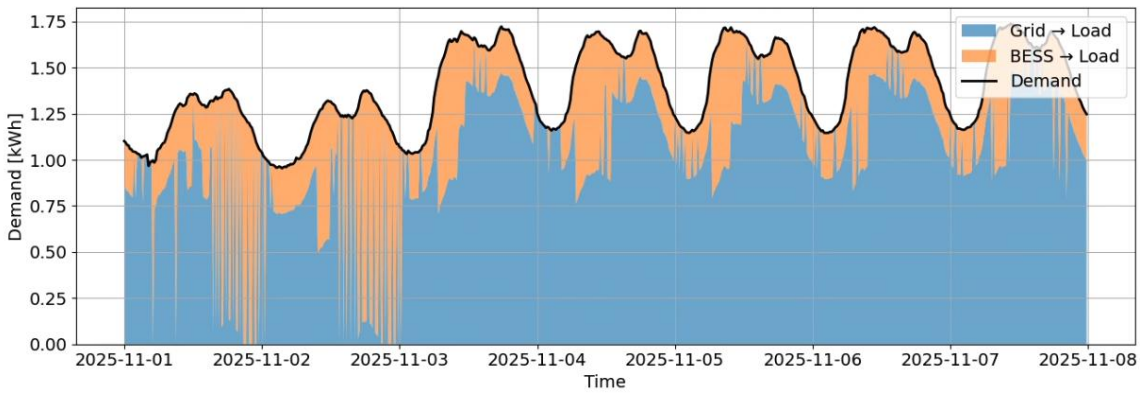

Abb. 4.11: Lastaufteilung des QR-DQN-Agenten im Peak-Shaving

Die Lastaufteilung in Abb. 4.11 verdeutlicht, dass der Speicher nicht ausschließlich zu Beginn einer Lastspitze entlädt, sondern die Energieabgabe über mehrere Zeitpunkte verteilt. Dadurch wird eine stabilere, wenn auch nicht vollständig idealisierte, Glättung der Netzlast erreicht. Auffällig sind jedoch die zahlreichen einzelnen Entladeimpulse, bei denen der Speicher in kurzen Zeitintervallen Energie bereitstellt. Dieses diskontinuierliche Verhalten führt dazu, dass die Lastkurve nicht gleichmäßig, sondern in mehreren kleineren Stufen geglättet wird.

Eine mögliche Ursache hierfür liegt im diskreten Aktionsraum des QR-DQN-Ansatzes. Da nur vordefinierte Leistungsstufen gewählt werden können, erfolgt die Leistungsanpassung nicht vollständig kontinuierlich. Im Gegensatz zu TD3, der stufenlose Entladeleistungen ermöglicht, entstehen hier einzelne, klar abgegrenzte Energieabgaben. Insgesamt zeigt der QR-DQN-Agent ein ausgewogenes Peak-Shaving-Verhalten, das zwischen der eher diskreten Reaktion des DQN und der teilweise aggressiven Entladung des TD3-Agenten einzuordnen ist, jedoch keine vollständig gleichmäßige Lastglättung erreicht.

5 Simulationsergebnisse

In diesem Kapitel werden die Regelungsansätze unter realistischen Markt- und Lastbedingungen evaluiert. Ziel ist es, die Leistungsfähigkeit der verschiedenen RL-Agenten sowie der regelbasierten Baseline hinsichtlich Wirtschaftlichkeit, Robustheit und Batteriedegradation zu vergleichen.

5.1 Vergleich Arbitrage

Im Folgenden werden die verschiedenen Regelungsansätze im Arbitrage-Szenario hinsichtlich wirtschaftlicher Performance und Auswirkungen auf den Batteriezustand systematisch gegenübergestellt. Bewertet werden der erzielte Markterlös, die verursachten Degradationskosten, der finale SoH sowie die kumulierte Belohnung über den gesamten Simulationszeitraum. Die entsprechenden Kennzahlen sind in Tab. 5.1 zusammengefasst.

Tab. 5.1: Vergleich der Regelungsansätze im Arbitrage-Szenario

Agent	**Erlös [€]**	**Degradationskosten [€]**	**Letzter SoH [%]**	**Belohnung [€]**
Rule-Based	9.124	**0.196**	**98.0**	8.927
DQN	11.012	0.459	95.4	10.554
TD3	6.980	0.241	97.6	6.739
QR-DQN	**13.179**	0.355	96.4	**12.824**

Der QR-DQN-Agent erzielt mit 13.179 € den höchsten Erlös aller betrachteten Ansätze. Gleichzeitig bleiben die Degradationskosten mit 0.355 € auf einem moderaten Niveau, sodass sich auch die höchste Gesamtbelohnung (12.824 €) ergibt. Dies deutet darauf hin, dass der distributionale Ansatz in der Lage ist, Preisvolatilitäten effizient auszunutzen, ohne die Batterie übermäßig zu belasten.

Der DQN-Agent erzielt ebenfalls einen hohen Erlös (11.012 €), verursacht jedoch die höchsten Degradationskosten (0.459 €) und weist mit 95.4% den niedrigsten finalen SoH auf. Dieses Ergebnis deutet darauf hin, dass neben ausgeprägten Hoch- und Niedrigpreisphasen auch moderate Preisbereiche zur Energieverschiebung genutzt werden. Die erhöhte Anzahl an Lade- und Entladezyklen geht mit einer stärkeren Beanspruchung der Batterie und entsprechend höheren Degradationskosten einher, obwohl der Agent den zweithöchsten Erlös innerhalb dieser Arbeit erzielt.

Der TD3-Agent erzielt hingegen den geringsten Erlös (6.980 €) und eine entsprechend niedrigere Gesamtbelohnung. Der vergleichsweise hohe finale SoH von 97.6% deutet darauf hin, dass moderate Preisbereiche seltener zur Energieverschiebung genutzt werden und insgesamt weniger Lade- und Entladezyklen auftreten. Die geringere Zyklisierung führt zu einer reduzierten Batteriebeanspruchung, geht jedoch zulasten der erzielten Arbitragegewinne.

Die regelbasierte Strategie erzielt stabile Ergebnisse mit einem moderaten Erlös (9.124 €) und sehr geringen Degradationskosten (0.196 €), sodass der SoH mit 98.0% nahezu erhalten bleibt. Da die Entscheidungslogik ausschließlich auf festen Schwellenwerten basiert und moderate Preisbereiche nicht berücksichtigt werden, tritt insgesamt eine geringere Zyklisierung des Speichersystems auf. Dies führt zu einer reduzierten Batteriebeanspruchung, begrenzt jedoch zugleich die Realisierung zusätzlicher Arbitragepotenziale im Vergleich zu lernbasierten Ansätzen.

Insgesamt zeigt der Vergleich, dass lernbasierte Verfahren insbesondere im volatilen Arbitrage-Betrieb deutliche wirtschaftliche Vorteile bieten können. Gleichzeitig wird sichtbar, dass unterschiedliche Agenten verschiedene Strategien im Spannungsfeld zwischen Gewinnmaximierung und Batteriedegradation entwickeln.

5.2 Vergleich Peak-Shaving

Im Peak-Shaving-Szenario werden die Regelungsansätze hinsichtlich der resultierenden Netzbezugskosten, der verursachten Degradationskosten sowie des finalen SoH verglichen. In der RL-Umgebung enthält die Belohnungsfunktion für das Peak-Shaving-Szenario zusätzlich einen Straffterm (vgl. Gleichung 3.7), der die Glättung von Lastspitzen incentivieren soll. Da dieser Straffterm jedoch keine physikalische Einheit besitzt und keinen unmittelbar monetarisierbaren wirtschaftlichen Verlust repräsentiert, wird dieser Straffterm für die in Tab. 5.2 dargestellte Kostenbewertung nicht berücksichtigt. Auf diese Weise wird eine konsistente Vergleichbarkeit der tatsächlichen ökonomischen Effekte der verschiedenen Regelungsansätze gewährleistet.

Tab. 5.2: Vergleich der Regelungsansätze im Peak-Shaving-Szenario

Agent	**Kosten [€]**	**Degradationskosten [€]**	**Letzter SoH [%]**	**Belohnung [€]**
Rule-Based	**76.395**	**0.160**	**98.4**	**-76.555**
DQN	83.602	0.302	97.0	-83.904
TD3	80.678	0.404	96.0	-81.082
QR-DQN	80.896	0.246	97.5	-81.142

Die niedrigsten Gesamtkosten werden durch den regelbasierten Ansatz erzielt (76.395 €), gefolgt vom TD3-Agenten (80.678 €). Auch hinsichtlich der Degradationskosten zeigt die regelbasierte Strategie mit 0.160 € die geringste Batteriebeanspruchung. Auf den ersten Blick könnte daraus eine Überlegenheit dieser Ansätze abgeleitet werden.

Eine detaillierte Betrachtung der Lastverläufe zeigt, dass die reine Kostenbetrachtung im Peak-Shaving-Betrieb nicht ausreicht, um die Zielerfüllung angemessen zu bewerten. Trotz geringer Kosten erfolgt beim regelbasierten Ansatz keine effektive Glättung von Lastspitzen. Die Speicherleistung wird nicht gezielt in Phasen hoher Netzlast eingesetzt, sodass die zentrale Zielsetzung des Peak-Shaving-Betriebs, nämlich die Reduktion von Lastspitzen, nicht erreicht wird.

Im Gegensatz zum regelbasierten Ansatz gelingt es den lernbasierten Agenten grundsätzlich besser, Lastspitzen zu reduzieren. DQN und QR-DQN zeigen dabei die deutlichste Glättung der Lastverläufe und sind in der Lage, auch länger andauernde Lastspitzen weitgehend zu kompensieren (siehe Abb. 4.5 und Abb. 4.11).

TD3 weist zwar eine leicht verbesserte Glättung gegenüber dem regelbasierten Ansatz auf, kann jedoch nur Teile der Lastspitzen abdecken (siehe Abb. 4.8). Insbesondere bei längeren Lastspitzen reicht die Entladeleistung nicht aus, um eine vergleichbare Reduktion wie bei DQN und QR-DQN zu erzielen. Die resultierende Glättung bleibt daher unvollständig und liegt qualitativ zwischen dem regelbasierten Ansatz und den diskreten DQN-Varianten.

Im Rahmen dieser Arbeit wurde keine explizite quantitative Metrik zur Bewertung der Peak-Shaving-Qualität implementiert, beispielsweise in Form der maximalen Restlast, der Lastvarianz oder einer definierten Peak-Reduktionsrate. Die Bewertung der Zielerfüllung erfolgt daher anhand der grafischen Analyse der Lastverläufe. Eine systematische quantitative Charakterisierung der Peak-Shaving-Performance stellt einen geeigneten Ansatz für weiterführende Untersuchungen dar.

5.3 Diskussion

Die Ergebnisse zeigen deutliche Unterschiede zwischen dem preisbasierenden regelbasierten Ansatz und den lernbasierten Verfahren sowohl im Arbitrage- als auch im Peak-Shaving-Szenario. Der in dieser Arbeit verwendete preisbasierte Rule-Based-Controller nutzt Perzentilschwellen zur Identifikation günstiger und teurer Marktphasen. Dadurch können Preisunterschiede grundsätzlich zur Generierung von Erlösen im Arbitrage-Betrieb genutzt werden. Aufgrund der fehlenden Fähigkeit, zukünftige Preisentwicklungen explizit zu bewerten oder Prognoseunsicherheiten systematisch zu berücksichtigen, erfolgt jedoch keine dynamische Anpassung der Lade- und Entladestrategie. Die Entscheidungslogik bleibt statisch und reagiert ausschließlich auf aktuelle Schwellenwerte. Dies führt zu vergleichsweise wenigen Lade- und

Entladezyklen, wodurch die Batteriebeanspruchung gering bleibt und der finale SoH den höchsten Wert unter allen untersuchten Ansätzen aufweist.

Im Vergleich dazu kann DQN Preisunterschiede deutlich effizienter ausnutzen. Durch den lernbasierten Ansatz werden prognostizierte Preis- und Lastdaten implizit in die Entscheidungsfindung integriert. Auch bei Vorliegen von Prognoseunsicherheiten gelingt es DQN, wirtschaftlich vorteilhafte Strategien zu erlernen. Dies führt im Arbitrage-Szenario zu höheren Erlösen im Vergleich zum regelbasierten Ansatz.

TD3 zeigt hingegen in der vorliegenden Simulation eine geringere Robustheit gegenüber Unsicherheiten in den Prognosedaten. Die resultierende Arbitrage-Performance bleibt hinter DQN zurück. Auch im Peak-Shaving-Szenario wird die Zielsetzung nur teilweise erfüllt. Insbesondere bei länger andauernden Lastspitzen gelingt keine ausreichende Reduktion, sodass die funktionale Zielerfüllung eingeschränkt bleibt.

QR-DQN erzielt im Arbitrage-Szenario die besten Ergebnisse unter den untersuchten RL-Agenten. Der distributionale Ansatz erlaubt eine verbesserte Berücksichtigung von Unsicherheiten in den prognostizierten Preis- und Lastdaten, was zu den höchsten Erlösen beziehungsweise der höchsten kumulierten Belohnung führt. Dies deutet darauf hin, dass die Modellierung der Return-Verteilung in volatilen und unsicheren Marktsituationen vorteilhaft ist.

Im Peak-Shaving-Betrieb zeigen DQN und QR-DQN eine deutlich bessere Reduktion von Lastspitzen im Vergleich zum regelbasierten Ansatz und TD3. Zwar erfolgt keine vollständige Glättung aller Lastspitzen, insbesondere bei länger andauernden Hochlastphasen, dennoch wird eine signifikante Reduktion erreicht. Zwischen DQN und QR-DQN zeigt sich zudem eine höhere Effizienz hinsichtlich der eingesetzten Kosten und Degradationsaufwendungen, da eine vergleichbare oder bessere Glättungswirkung mit geringerer Batteriebeanspruchung erzielt wird.

Insgesamt verdeutlichen die Ergebnisse, dass lernbasierte Verfahren insbesondere unter Prognoseunsicherheiten deutliche Vorteile gegenüber statischen Schwellenregeln aufweisen. Die Fähigkeit zur impliziten Berücksichtigung von Unsicherheiten und zur dynamischen Anpassung der Strategie stellt einen entscheidenden Faktor für die wirtschaftliche und funktionale Performance dar.

6 Fazit und Ausblick

Ziel dieser Arbeit war die Entwicklung und Evaluation von RL-basierten Regelungsansätzen für ein netzgekoppeltes BESS unter Berücksichtigung von Preis- und Nachfrageunsicherheiten. Dabei wurden sowohl ein Arbitrage- als auch ein Peak-Shaving-Szenario untersucht und mit einem regelbasierten Referenzansatz (Rule-Based-Controller) verglichen.

Die Ergebnisse zeigen, dass lernbasierte Verfahren im Arbitrage-Betrieb deutliche wirtschaftliche Vorteile gegenüber einem statischen Schwellenregel-Ansatz erzielen können. Während der preisbasierte Rule-Based-Controller durch die verwendeten Perzentilschwellen Preisunterschiede grundsätzlich nutzen kann, fehlt die Fähigkeit zur dynamischen Anpassung an prognostizierte Marktverläufe und Unsicherheiten. Dies führt zu einer konservativen Betriebsweise mit geringer Batteriebeanspruchung, jedoch begrenztem Erlöspotenzial.

Die untersuchten RL-Agenten zeigen eine höhere Adaptivität gegenüber Preis- und Lastverläufen. Insbesondere QR-DQN erzielt im Arbitrage-Szenario die höchste Erlös- und Gesamtbelohnung. Die Modellierung der Return-Verteilung ermöglicht eine verbesserte Berücksichtigung von Unsicherheiten und führt zu robusteren Entscheidungsstrategien unter volatilen Marktbedingungen. DQN zeigt ebenfalls eine deutliche Leistungssteigerung gegenüber dem regelbasierten Ansatz. TD3 hingegen weist in der vorliegenden Simulation eine geringere Robustheit gegenüber Prognosefehlern auf und erzielt entsprechend schwächere Ergebnisse.

Im Peak-Shaving-Szenario wird deutlich, dass eine reine Kostenbetrachtung nicht ausreicht, um die Zielerfüllung zu bewerten. Während der regelbasierte Ansatz geringe Kosten verursacht, wird die funktionale Zielsetzung der Lastspitzenreduktion nicht erreicht. DQN und QR-DQN zeigen eine signifikant bessere Glättung der Lastverläufe, wenngleich keine vollständige Eliminierung länger andauernder Lastspitzen erfolgt. TD3 erreicht lediglich eine partielle Verbesserung gegenüber dem regelbasierten Ansatz.

Allerdings sind Verbesserungen und Erweiterungen der in dieser Arbeit entwickelten Ansätze möglich und sinnvoll. Eine erste Erweiterung betrifft den regelbasierten Controller. Der in dieser Arbeit eingesetzte preisbasierte Schwellenregel-Ansatz berücksichtigt ausschließlich aktuelle Preisniveaus. Eine Integration expliziter Prognoseinformationen sowie eine zusätzliche Peak-Shaving-Regel könnte die Entscheidungslogik erweitern und eine dynamischere Reaktion auf erwartete Markt- und Lastverläufe ermöglichen. Dadurch ließe sich die Abhängigkeit von statischen Perzentilschwellen reduzieren und die funktionale Zielerfüllung insbesondere im Peak-Shaving-Szenario verbessern.

Weiterhin wurde im Rahmen dieser Arbeit keine explizite quantitative Metrik zur Bewertung des Peak-Shaving-Erfolgs implementiert. Zukünftige Arbeiten könnten eine systematische Kennzahl entwickeln, beispielsweise auf Basis der maximalen Restlast, der Varianz der Netzlast oder einer definierten Peak-Reduktionsrate. Eine solche Metrik würde eine objektive und reproduzierbare Bewertung der Zielerreichung ermöglichen.

Auch das verwendete Degradationsmodell stellt einen Ansatzpunkt für Erweiterungen dar. In dieser Arbeit wurde ein lineares Modell zur Approximation des SoC-Verlusts verwendet. Realistische Alterungsprozesse von Lithium-Ionen-Batterien weisen jedoch nichtlineare Charakteristika auf und hängen beispielsweise vom Ladezustand oder von der Zyklentiefe ab. Die Integration einer nichtlinearen Alterungsgleichung oder eines physikalisch fundierten Degradationsmodells könnte die Aussagekraft der Simulationsergebnisse weiter erhöhen.

Im Gegensatz zu regelbasierten Ansätzen bieten lernbasierte Regelungsarchitekturen Vorteile hinsichtlich wirtschaftlicher Performance und funktionaler Zielerfüllung im Peak-Shaving-Betrieb, ohne auf fest definierte Entscheidungsregeln angewiesen zu sein. Dabei zeigt QR-DQN als distributionaler RL-Agent eine höhere Fähigkeit, Entscheidungen unter Unsicherheitsbedingungen zu treffen, als DQN und TD3.

Literaturverzeichnis

[1] D. Cao, W. Hu, J. Zhao, u.a., “Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review,” *Journal of Modern Power Systems and Clean Energy*, Bd. 8, Nr. 6, S. 1029-1042, 2020.

[2] M. Mussi, L. Pellegrino, O. F. Pindaro, u.a., “A Reinforcement Learning Controller Optimizing Costs and Battery State of Health in Smart Grids,” *Journal of Energy Storage*, Bd. 82, Nr. 110572, 2024.

[3] J. Li, C. Wang und H. Wang, “Deep Reinforcement Learning for Wind and Energy Storage Coordination in Wholesale Energy and Ancillary Service Markets,” *Energy and AI*, Bd. 14, Nr. 100280, 2023.

[4] X. Chen, G. Qu, Y. Tang, S. Low und N. Li, “Reinforcement Learning for Selective Key Applications in Power Systems: Recent Advances and Future Challenges,” *IEEE Transactions on Smart Grid*, Bd. 13, Nr. 4, S. 2935-2958, 2022.

[5] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare und J. Pineau, “An Introduction to Deep Reinforcement Learning,” *Foundations and Trends in Machine Learning*, Bd. 11, Nr. 3-4, S. 219-354, 2018.

[6] J. R. Vázquez-Canteli und Z. Nagy, “Reinforcement Learning for Demand Response: A Review of Algorithms and Modeling Techniques,” *Applied Energy*, Bd. 235, S. 1072–1089, 2019.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, u.a., “Human-Level Control Through Deep Reinforcement Learning,” *Nature*, Bd. 518, Nr. 7540, S. 529-533, 2015.

[8] R. S. Sutton und A. G. Barto, *Reinforcement Learning: An Introduction*, 2. Aufl. Cambridge, MA, USA: MIT Press, 2018.

[9] S. Fujimoto, H. van Hoof, und D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholm, Sweden, 2018, S. 1587-1596.

[10] M. G. Bellemare, W. Dabney, und R. Munos, “A Distributional Perspective on Reinforcement Learning,” in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017, S. 449-458.

[11] W. Dabney, M. Rowland, M. G. Bellemare, und R. Munos, “Distributional Reinforcement Learning with Quantile Regression,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, New Orleans, USA, 2018, S. 2892-2901.

[12] Y. Ding, X. Chen, und J. Wang, “Deep Reinforcement Learning-Based Method for Joint Optimization of Mobile Energy Storage Systems and Power Grid with High Renewable Energy Sources,” *Batteries*, Bd. 9, Nr. 4, Art.-Nr. 219, 2023.

[13] G. L. Plett, *Battery Management Systems, Volume I: Battery Modeling*, Norwood, MA, USA: Artech House, 2015.

[14] J. S. Edge, S. O’Kane, R. Prosser und N. Kirkaldy, “Lithium Ion Battery Degradation: What You Need to Know,” *Physical Chemistry Chemical Physics*, Bd. 23, Nr. 14, S. 8200-8221, 2021.

[15] B. Tepe, S. Jablonski, H. Hesse und A. Jossen, “Lithium-Ion Battery Utilization in Various Modes of E-Transportation,” *eTransportation*, Bd. 18, Art. Nr. 100274, 2023.

[16] J. Cao, D. Harrold, Z. Fan, T. Morstyn, D. Healey und K. Li, “Deep Reinforcement Learning-Based Energy Storage Arbitrage with Accurate Lithium-Ion Battery Degradation Model,” in *IEEE Transactions on Smart Grid*, Bd. 11, Nr. 5, S. 4513-4521, 2020.

[17] H. T. Nguyen und D. -H. Choi, “Three-Stage Inverter-Based Peak Shaving and Volt-VAR Control in Active Distribution Networks Using Online Safe Deep Reinforcement Learning,” in *IEEE Transactions on Smart Grid*, Bd. 13, Nr. 4, S. 3266-3277, 2022.

[18] J. Unpingco, *Python for Probability, Statistics, and Machine Learning*, 2. Aufl. Cham: Springer Verlag, 2019.

[19] J. Vanderplas, *Data Science mit Python: das Handbuch für den Einsatz von IPython, Jupyter, NumPy, Pandas, Matplotlib, Scikit-Learn*, übers. K. Lorenzen, Frechen: Mitp Verlag, 2018.

[20] M. Towers, A. Kwiatkowski, J. Terry, u.a., “Gymnasium: A Standard Interface for Reinforcement Learning Environments,” arXiv:2407.17032, 2024.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus und N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, Bd. 22, Nr. 268, S. 1-8, 2021.

[22] Energie-Charts, *Stromproduktion und Börsenstrompreise in Deutschland im November 2024*. [Online]. Verfügbar unter: https://energy-charts.info/charts/price_spot_market/chart.htm?l=de&c=DE&interval=month&year=2024&month=11 (Zugriff am: 14. Februar 2025).

[23] SMARD, *Marktdaten*. [Online]. Verfügbar unter: https://www.smard.de/home/downloadcenter/download-marktdaten/ (Zugriff am: 14. Februar 2025).

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich dieses Projekt selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind. Die Arbeit wurde noch nicht bei einer Prüfungsbehörde eingereicht oder veröffentlicht. Zusätzlich erkläre ich, dass ich Generative KI-Technologien ausschließlich zur Verbesserung des Codes und der Rechtschreibung eingesetzt habe. Dabei wurden keine sensiblen Daten eingegeben oder ganze Texte generiert. Der Einsatz dieser Technologien beschränkte sich strikt auf technische Unterstützung ohne Einfluss auf die inhaltliche Eigenleistung.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

23. Februar 2026	Chin-I Feng
Datum	Unterschrift

Anhang

A1 Unsicherheitsmodell-Code

Der folgende Codeauszug zeigt die Implementierung des horizonabhängigen Prognoseunsicherheitsmodells. Die Standardabweichung σ_h steigt mit dem Prognosehorizont an, während die Fehlerterme gaußverteilt erzeugt werden.

```
import numpy as np


class ForecastScenarioGenerator:
    """
    Generates deterministic, horizon-dependent Gaussian noise
    for electricity price and demand forecasts.

    Noise is:
    - zero-mean Gaussian
    - sigma increases with forecast horizon
    - reproducible via base_seed
    """

    def __init__(self, horizon_steps, sigma0, sigmaH, schedule="sqrt", base_seed=42):
        self.H = int(horizon_steps)
        self.sigma0 = float(sigma0)
        self.sigmaH = float(sigmaH)
        self.schedule = schedule
        self.base_seed = int(base_seed)

        h = np.arange(1, self.H + 1, dtype=np.float32)
        x = h / self.H

        if schedule == "linear":
            sigma = self.sigma0 + (self.sigmaH - self.sigma0) * x
        elif schedule == "sqrt":
            sigma = self.sigma0 + (self.sigmaH - self.sigma0) * np.sqrt(x)
        else:
            raise ValueError(f"Unknown sigma schedule: {schedule}")

        self.sigma = np.maximum(sigma, 0.0).astype(np.float32)

    def generate_episode_noise(self, episode_len: int, episode_id: int) -> np.ndarray:
        rng = np.random.default_rng(self.base_seed + episode_id)
        return rng.normal(0.0, 1.0, size=(episode_len, self.H)).astype(np.float32)
```

Abb. A.1: Implementierung des Prognoseunsicherheitsmodells

A2 SoC-Update-Logik-Code

Der folgende Codeauszug zeigt die Implementierung der SoC-Aktualisierung auf Basis der diskreten Energiebilanz. Die zu- oder abgeführte Energie wird unter Berücksichtigung der Wirkungsgrade auf die Batteriekapazität bezogen und anschließend auf die definierten Hard-Grenzen begrenzt.

```
energy_cmd_kWh = a_cmd_kW * self.dt


if energy_cmd_kWh >= 0.0:
    delta_soc = (energy_cmd_kWh * self.eta_c) / self.capacity
else:
    delta_soc = (energy_cmd_kWh / self.eta_d) / self.capacity


self.soc = np.clip(
    self.soc + delta_soc,
    self.soc_hard_min,
    self.soc_hard_max
)
```

Abb. A.2: Implementierung der SoC-Update-Logik

A3 Degradationsmodell-Code

Der folgende Codeauszug zeigt die Implementierung des Degradationsmodells auf Basis EFC. Die Alterung wird proportional zur absoluten SoC-Änderung modelliert, wobei die SoH-Reduktion linear mit der Anzahl der Vollzyklen zunimmt.

```
delta_soc_actual = abs(self.soc - self._last_soc)
efc_step = delta_soc_actual / 2.0


self._efc_acc += efc_step
self._last_soc = self.soc


deg_cost_eur = efc_step * self.deg_cost_per_EFC
self.soh = max(
    self.soh_min,
    self.soh - self.soh_deg_per_EFC * efc_step
)
```

Abb. A.3: Implementierung des Degradationsmodells

A4 Zustandsraum-Implementierung

Der folgende Codeauszug zeigt die Implementierung der Zustandsvektorkonstruktion innerhalb der RL-Umgebung. Neben dem aktuellen SoC und SoH werden normierte Markt- und Lastgrößen, die letzte Aktion sowie zyklische Zeitmerkmale (Sinus-/Kosinus-Transformation) in den Zustandsvektor integriert. Die Normierung erfolgt auf Basis der maximalen Beobachtungswerte, um eine numerisch stabile Lernumgebung zu gewährleisten.

```
def _get_obs(self, price_obs: float, demand: float | None):
    sin_tod, cos_tod, sin_doy, cos_doy = self._get_time_features(self.t)

    price_norm = float(price_obs) / (self._max_price + 1e-6)

    if self._max_demand is None:
        demand_norm = 0.0
    else:
        demand_norm = 0.0 if demand is None else float(demand) / (self._max_demand + 1e-6)

    last_action_norm = float(self.last_action / self.p_max)

    obs_components = [
        self.soc,
        self.soh,
        last_action_norm,
        sin_tod,
        cos_tod,
        sin_doy,
        cos_doy,
        price_norm,
        demand_norm,
    ]

      .
      .
      .

    return np.array(obs_components, dtype=np.float32)
```

Abb. A.4: Codeausschnitt des Zustandsraums

A5 Aktionsraum-Implementierung

Der folgende Codeausschnitt zeigt die Implementierung des Aktionsraums innerhalb der RL-Umgebung. Abhängig vom gewählten Lernalgorithmus wird zwischen einem kontinuierlichen und einem diskreten Aktionsraum unterschieden.

```
# Discrete actions
# Arbitrage behavior (allow_grid_export=True):
#   - actions in [-Pmax, +Pmax]
# Peak shaving behavior (allow_grid_export=False):
#   - 21 actions split into:
#       0..9   : discharge-to-load (10 levels)
#       10     : idle
#       11..20 : charge-from-grid (10 levels)
if self.use_discrete:
    if self.allow_grid_export:
        if discrete_action_values is None:
            discrete_action_values = np.linspace(-1.0, 1.0, 21).tolist()
        self.discrete_action_values =
        np.array(discrete_action_values, dtype=np.float32) * self.p_max
    else:
        self._discharge_levels = (np.linspace(0.1, 1.0, 10, dtype=np.float32) * self.p_max)
        self._charge_levels = (np.linspace(0.1, 1.0, 10, dtype=np.float32) * self.p_max)
        self._idle_index = 10
else:
    if discrete_action_values is None:
        discrete_action_values = np.linspace(-1.0, 1.0, 21).tolist()
    self.discrete_action_values = np.array(discrete_action_values, dtype=np.float32) * self.p_max
```

Abb. A.5: Codeausschnitt des Aktionsraums

A6 Belohnungsfunktion-Implementierung

Der folgende Codeausschnitt zeigt die Implementierung der Belohnungsberechnung innerhalb der RL-Umgebung. Die Belohnung setzt sich aus dem ökonomischen Ergebnis des jeweiligen Zeitschritts, den Degradationskosten sowie im Peak-Shaving-Szenario einem zusätzlichen Strafterm bei Überschreitung der definierten Netzimportgrenze zusammen.

Die konkrete Zusammensetzung hängt vom gewählten Szenario ab: Im Arbitrage-Modus wird ausschließlich der Energiehandel sowie die Degradation berücksichtigt, während im Peak-Shaving-Modus zusätzlich eine quadratische Strafkomponente zur Begrenzung von Lastspitzen aktiviert wird.

```
reward = float(revenue_eur - deg_cost_eur + float(penalty_demand_cap))
```

Abb. A.6: Allgemeine Belohnungsfunktion

```
self._grid_import_cap_offpeak_kWh = float(np.quantile(demand_kWh_series, 0.6))

self._grid_import_cap_peak_kWh = float(np.quantile(demand_kWh_series, 0.3))
```

Abb. A.7: Berechnung der Netzimportgrenzen im Peak-Shaving-Szenario

```
.
.
.

if self.allow_grid_export:
    # ---------------- Arbitrage MODE (MARKET BUY/SELL) ----------------
    if energy_cmd_kWh >= 0.0:
        delta_soc_cmd = (energy_cmd_kWh * self.eta_c) / self.capacity
    else:
        delta_soc_cmd = (energy_cmd_kWh / self.eta_d) / self.capacity

    soc_pre_cmd = self.soc + delta_soc_cmd

    if energy_cmd_kWh >= 0.0:
        soc_headroom = self.soc_hard_max - self.soc
        energy_max_kWh = (soc_headroom * self.capacity) / max(self.eta_c, 1e-6)
        energy_eff_kWh = float(np.clip(energy_cmd_kWh, 0.0, energy_max_kWh))
    else:
        soc_above_min = self.soc - self.soc_hard_min
        energy_min_kWh = -(soc_above_min * self.capacity * self.eta_d)
        energy_eff_kWh = float(np.clip(energy_cmd_kWh, energy_min_kWh, 0.0))

    if energy_eff_kWh >= 0.0:
        delta_soc = (energy_eff_kWh * self.eta_c) / self.capacity
    else:
        delta_soc = (energy_eff_kWh / self.eta_d) / self.capacity

    self.soc = float(np.clip(self.soc + delta_soc, self.soc_hard_min, self.soc_hard_max))

    violated_soft_cmd = False

    revenue_eur = -price_per_kWh * energy_eff_kWh

.
.
.
```

Abb. A.8: Codeausschnitt zur Implementierung der Erlös- und Kostenberechnung im Arbitrage-Szenario

```
else:
    # ---------------- PEAK SHAVING MODE (MARKET BUY, but no SELL) ----------------
    # Charge   : grid -> bess
    # Discharge: bess -> load

    if energy_cmd_kWh > 0.0:
        # CHARGE
        soc_headroom = self.soc_hard_max - self.soc
        energy_to_batt_max_kWh = soc_headroom * self.capacity
        energy_to_batt_cmd_kWh = energy_cmd_kWh * self.eta_c
        energy_to_batt_kWh = float(np.clip(energy_to_batt_cmd_kWh, 0.0, energy_to_batt_max_kWh))
        delta_soc = energy_to_batt_kWh / self.capacity
        self.soc = float(np.clip(self.soc + delta_soc, self.soc_hard_min, self.soc_hard_max))
        grid_charge_kWh = energy_to_batt_kWh / max(self.eta_c, 1e-6)

    elif energy_cmd_kWh < 0.0:
        # DISCHARGE to load (no selling)
        req_deliver_kWh = min(abs(energy_cmd_kWh), demand_kWh)
        energy_available_from_batt_kWh = (self.soc - self.soc_hard_min) * self.capacity
        max_deliver_kWh = energy_available_from_batt_kWh * self.eta_d
        supplied_to_load_kWh = float(np.clip(req_deliver_kWh, 0.0, max_deliver_kWh))
        energy_from_batt_kWh = supplied_to_load_kWh / max(self.eta_d, 1e-6)
        delta_soc = -energy_from_batt_kWh / self.capacity
        self.soc = float(np.clip(self.soc + delta_soc, self.soc_hard_min, self.soc_hard_max))

    # remaining office load is imported from grid
    grid_import_load_kWh = max(0.0, demand_kWh - supplied_to_load_kWh)
    if price_true >= self._price_peak_threshold:
        cap_now_kWh = float(self._grid_import_cap_peak_kWh)
    else:
        cap_now_kWh = float(self._grid_import_cap_offpeak_kWh)

    exceed_kWh = max(0.0, grid_import_load_kWh - cap_now_kWh)
    penalty_demand_cap = -self.demand_cap_penalty_k * (exceed_kWh ** self.demand_cap_penalty_power)
    violated_soft_cmd = False
    total_grid_kWh = grid_import_load_kWh + grid_charge_kWh
    revenue_eur = -price_per_kWh * total_grid_kWh
    self._last_grid_total_kW = float(total_grid_kWh / max(self.dt, 1e-9))
```

Abb. A.9: Codeausschnitt zur Berechnung des Strafterms bei Überschreitung der Netzimportgrenze im Peak-Shaving-Szenario

A7 Rule-Based-Controller-Implementierung

Der folgende Codeausschnitt zeigt die Implementierung der regelbasierten Entscheidungslogik innerhalb der Simulationsumgebung.

```
def _price_to_factor(self, price: float) -> float:
    """
    Convert raw price to a factor in [+1, -1] using percentile scaling.
    """
    price_norm = (price - self.p_low) / (self.p_high - self.p_low)
    price_norm = float(np.clip(price_norm, 0.0, 1.0))

    return 1.0 - 2.0 * price_norm
 .
 .
 .
def act(self, obs):
    soc = float(obs[0])
    price = self._get_current_price(obs)

    # 1) Price-based factor
    factor = self._price_to_factor(price)

    # 2) SoC safety override (controller-level safe band)
    if soc >= self.soc_max_safe and factor > 0.0:
        factor = 0.0
    if soc <= self.soc_min_safe and factor < 0.0:
        factor = 0.0

    # 3) Convert factor into kW command
    p_cmd = factor * float(self.env.p_max)
 .
 .
 .
    return np.array([p_cmd], dtype=np.float32)
```

Abb. A.10: Codeausschnitt zur Implementierung des Rule-Based Controllers