

Optimierung eines PID-gesteuerten DC-DC-Konverters mit maschinellem Lernen

Patryk Krzyzanski

Erstprüfer: Prof. Dr.-Ing. Bernhard Wicht
Zweitprüfer: Dr.-Ing. Markus Olbrich

Leibniz Universität Hannover
Abteilung: Institut für Mikroelektronische Systeme

June 19, 2024

Abstract

In dieser Arbeit wird die Verwendung neuronaler Netze zur Optimierung und Steuerung eines PID-regulierten DC-Konverters untersucht. Das Ziel besteht darin, ein System zu entwickeln, das in der Lage ist, sowohl die altersbedingte Degradation als auch Fertigungstoleranzen von Schaltungskomponenten wie Kapazität und Induktivität zu überwachen und anzupassen, um eine dauerhaft optimale Leistung des Konverters sicherzustellen. Ein besonderer Fokus liegt auf dem Trainingsprozess und der Architektur des neuronalen Netzes. Der Trainingsprozess wird mithilfe von Methoden wie Deep Deterministic Policy Gradient (DDPG) und Bayesscher Optimierung umgesetzt. Das Training und die Schaltungssimulation werden unter Einsatz von Transientenanalyse mit SystemC durchgeführt, um eine präzise Bewertung und Auswertung der Simulationsergebnisse zu ermöglichen. Es werden Techniken zur Optimierung der Hyperparameter des neuronalen Netzes vorgestellt. Herausforderungen und Lösungsansätze im Kontext der neuronalen Netzarchitektur und des Trainings werden diskutiert. Abschließend werden die erzielten Ergebnisse und ihre Implikationen für zukünftige Forschungen präsentiert.

Contents

1 Einleitung	3
1.1 Hintergrund und Motivation	3
1.2 Problemstellung	3
1.3 Relevanz und Forschungslage	3
1.4 Forschungsmethode und Ansatz	3
1.5 Verwendete Methoden	4
2 Grundlagen	6
2.1 Informationstechnologie	6
2.1.1 Einführung in Neuronale Netzwerke	6
2.1.2 Vorwärtspropagation in Neuronalen Netzwerken	7
2.1.3 Einführung in die Backpropagation	8
2.1.4 Optimierungstechniken	10
2.1.5 Vermeidung von Overfitting in Neuronalen Netzen	11
2.1.6 Vanishing-Gradient-Problem	12
2.1.7 Bedeutung der Lernrate	13
2.1.8 Bedeutung der Batch-Größe	13
2.2 Reinforcement Learning	14
2.2.1 Einleitung	14
2.2.2 Markovsche Prozesse und deren Rolle im Reinforcement Learning	15
2.2.3 Das Multi-Armed Bandit Problem im Kontext der Degradation	16
2.2.4 Markow-Entscheidungsprozesse und die Bellman-Gleichung	16
2.2.5 Policy-Based Reinforcement Learning	17
2.2.6 Q-Learning	18
2.2.7 Deep Deterministic Policy Gradient (DDPG)	21
2.2.8 Reward-Funktion für Trading-Bots	26
3 Realisierung	28
3.1 Motivation und Zielsetzung	28
3.2 Herausforderungen und Ziele	28
3.3 Wahl der Architektur und verwendete Technologien	28
3.3.1 Reinforcement Learning mit Python	29
3.3.2 SystemC und C++ für die Schaltungssimulation	29
3.4 Überblick über den Aufbau und Datenfluss	30
3.4.1 Makrozyklus: Systemüberblick und Datenauswertung	30
3.4.2 Mikrozyklus: Detailanalyse und Regelungsstrategien	31
3.4.3 Makrozyklus Schritt 1: Zustandssimulation und Datengenerierung	32
3.4.4 Makrozyklus Schritt 2: Entscheidungsfindung durch den Actor	32
3.4.5 Gesamtdarstellung des Regelkreises mit PID-gesteuertem DCDC-Konverter	33
3.4.6 Selbsttestmodul im Lernprozess des Agenten	34
3.5 Zusammenführung des Actor-Critic-Verfahrens	34

4 Ergebnisse	39
4.1 Hyperparameter	40
4.2 Methodik des Vergleichs	40
4.3 Vergleich und Validierung der Modellleistung	40
4.4 Phase 1: Einfache und Fortgeschrittene Netzwerkmodelle	41
4.5 Phase 2: Vertiefte Analyse und Optimierung in Erweiterten Parameterräumen	44
4.5.1 Teil I. Anfängliche Lernprozesse und Batch-Füllung	46
4.5.2 Teil II. Fehlleitung durch den Kritiker	47
4.5.3 Teil III. Feinjustierung und Konvergenz in Richtung Optimum	47
4.5.4 Teil IV. Fortgeschrittene Konvergenz und Feinabstimmung	48
4.5.5 Teil V. Erzielung einer nahezu optimalen Regelungsleistung	49
4.5.6 Synthese der Ergebnisse	49
4.6 Phase 3: Miniaturisierung und Leistungsanalyse	50
5 Diskussion	52
6 Zukünftige Arbeit	53

Chapter 1

Einleitung

1.1 Hintergrund und Motivation

Die Anwendung von Reinforcement Learning (RL) im Bereich der Finanzmärkte hat in den letzten Jahren erheblich an Bedeutung gewonnen. Insbesondere die Entwicklung von Trading-Bots, die in der Lage sind, autonom Handelsentscheidungen zu treffen, bietet ein enormes Potenzial für die Verbesserung von Handelsstrategien und die Maximierung von Gewinnen. Die Effizienz und Effektivität dieser Systeme hängen jedoch maßgeblich von der Qualität der zugrunde liegenden Modelle und Algorithmen ab.

1.2 Problemstellung

Der Einsatz von RL in Handelsumgebungen bringt spezifische Herausforderungen mit sich. Ein wesentlicher Aspekt ist die Handhabung zeitsequenzieller Daten, bei denen die zeitliche Dimension eine entscheidende Rolle spielt. Ein weiterer zentraler Punkt ist die Gestaltung der Reward-Funktion, die den Lernprozess des Trading-Bots maßgeblich beeinflusst und dessen Fähigkeit zur Erzielung profitabler Handelsentscheidungen bestimmt.

1.3 Relevanz und Forschungslage

Frühere Studien und Anwendungen haben gezeigt, dass RL-Ansätze vielversprechend für die Entwicklung von autonomen Trading-Systemen sind. Arbeiten von Brunton und Kutz sowie Almawlawi et al. haben bereits die Wirksamkeit von RL-Modellen in verschiedenen Kontexten demonstriert [3] [2]. Jedoch bleibt die Herausforderung bestehen, diese Modelle effizient auf handelsübliche Rechner zu übertragen und gleichzeitig ihre Leistungsfähigkeit zu maximieren.

1.4 Forschungsmethode und Ansatz

Diese Arbeit verfolgt den Ansatz, ein RL-Modell für Trading-Bots zu entwickeln, das auf einem handelsüblichen Rechner mit begrenzten Ressourcen trainiert werden kann. Hierbei wird ein spezieller Konvolutions-Layer eingesetzt, der es ermöglicht, sehr lange Datenreihen zu verarbeiten, ohne die Qualität der Ergebnisse zu beeinträchtigen. Zudem wird eine logarithmische Funktion verwendet, um die Verarbeitung der Daten im Konvolutions-Layer zu optimieren. Die Modellarchitektur basiert auf einem Deep Deterministic Policy Gradient (DDPG), der mit Konvolutions-Layern und Long Short-Term Memory (LSTM) Netzwerken trainiert wird. Dieses Modell nutzt mehrere Kanäle gleichzeitig, um Abhängigkeiten zu erfassen und zu verarbeiten.

Um die Ressourcennutzung zu optimieren, wird eine parallele Trainingsarchitektur verwendet. Ein zentrales Actor-Critic-Netzwerk lernt aus den Erfahrungen vieler parallel laufender Simulationen, was die Speicherauslastung und die Effizienz der Trainingsprozesse verbessert.

1.5 Verwendete Methoden

In dieser Arbeit werden folgende Methoden und Modelle verwendet:

- **Deep Deterministic Policy Gradient (DDPG):** Ein algorithmischer Ansatz zur Entscheidungsfindung in kontinuierlichen Aktionsräumen.
- **Konvolutions-Layer:** Spezielle Layer, die es ermöglichen, sehr lange Datenreihen effizient zu verarbeiten.
- **Long Short-Term Memory (LSTM):** Ein spezieller RNN-Typ zur Verarbeitung und Vorhersage von zeitsequenziellen Daten.
- **Logarithmische Funktionen:** Optimierung der Datenverarbeitung im Konvolutions-Layer.
- **Parallele Architektur:** Nutzung eines zentralen Actor-Critic-Netzwerks, das aus mehreren parallel laufenden Simulationen lernt.

Bibliography

- [1] Brunton, S. L., & Kutz, J. N. (2019). Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge University Press.
- [2] Almawlawe, A., et al. (2023). Applications of Reinforcement Learning in Financial Trading. Journal of Financial Markets.

Chapter 2

Grundlagen

Dieses Kapitel dient als umfassende Grundlage für die Erforschung der Rolle neuronaler Netze in der Optimierung und Steuerung von PID-regulierten DC-Konvertern. Im Fokus stehen sowohl die Grundlagen der DC-DC-Konvertertechnologie als auch spezielle Herausforderungen, die in diesem Kontext auftreten können, wie beispielsweise die altersbedingte Degradation von Schaltungskomponenten. Darüber hinaus bietet das Kapitel einen Überblick über moderne Optimierungsmethoden wie DDPG (Deep Deterministic Policy Gradients). Der Inhalt dieses Kapitels zielt darauf ab, den Leser umfassend auf die Herausforderungen, technischen Lösungen und innovativen Ansätze in diesem sich schnell entwickelnden Forschungsfeld vorzubereiten.

2.1 Informationstechnologie

2.1.1 Einführung in Neuronale Netzwerke

Neuronale Netzwerke bilden das rechnerische Grundgerüst für eine Vielzahl von Aufgaben in den Bereichen maschinelles Lernen und künstliche Intelligenz. Sie sind von dem komplexen Netzwerk an Neuronen im menschlichen Gehirn inspiriert und versuchen, biologische Lernprozesse nachzuahmen [1]. In diesem Zusammenhang bieten sie ein robustes und flexibles Rahmenwerk zur Lösung komplexer Herausforderungen [5].

Vorteile von Neuronalen Netzwerken

Neuronale Netzwerke bieten mehrere entscheidende Vorteile, die ihren Einsatz in unterschiedlichen Anwendungsbereichen attraktiv machen:

- **Parallelität:** Sie sind für die parallele Verarbeitung konzipiert und ermöglichen daher schnelle Berechnungen sowie Echtzeitverarbeitung.
- **Nichtlineare Funktionsapproximation:** Die Netzwerke sind besonders gut geeignet, nichtlineare Funktionen zu approximieren [5], was sie vielseitig einsetzbar macht.
- **Modellgeneralisierung:** Neuronale Netzwerke können aus einer begrenzten Datenmenge generalisieren und somit präzise Vorhersagen für unbekannte Eingaben treffen.

Diese Vorteile bilden die Grundlage für ihre breite Anwendbarkeit, die im nächsten Abschnitt erläutert wird.

Unterschied zwischen Biologischen und Künstlichen Neuronalen Netzwerken

Künstliche neuronale Netzwerke bestehen aus rechnerischen Einheiten, den sogenannten Neuronen. Diese sind durch anpassbare Gewichtungen verbunden, die der Stärke synaptischer Verbindungen in biologischen Systemen ähneln. Lernen erfolgt durch die Anpassung dieser Gewichtungen, ähnlich wie sich die Stärken synaptischer Verbindungen in biologischen Systemen als Reaktion auf Reize ändern [1].

Deep Learning als Spezialisierung

Deep Learning stellt einen spezialisierten Unterbereich des maschinellen Lernens dar, der neuronale Netzwerke mit drei oder mehr Schichten verwendet. Diese tiefen Netzwerke führen eine hierarchische Merkmalsextraktion durch, die es ihnen ermöglicht, immer komplexere Muster und Merkmale zu erkennen, während die Daten durch die Schichten fließen.

Zusammenfassung und Ausblick

Zusammenfassend bieten neuronale Netzwerke ein leistungsfähiges Rahmenwerk für eine Vielzahl von Aufgaben, von einfacher Mustererkennung bis hin zu komplexen Entscheidungsfindungsprozessen. Der Einsatz von mehrschichtigen Architekturen und nichtlinearen Aktivierungsfunktionen erweitert die Fähigkeiten traditioneller maschineller Lernalgorithmen [1].

Mit dieser Grundlage werden die folgenden Abschnitte einen vertieften Einblick in die mathematischen Aspekte von neuronalen Netzwerken bieten. Insbesondere werden wir uns darauf konzentrieren, wie neuronale Netzwerke bei der Optimierung und Steuerung von PID-regulierten DC-Konvertern Anwendung finden. Dabei liegt der Fokus auf der Berücksichtigung von Alterungsprozessen und Abnutzung von Schaltungselementen wie Kapazitäten und Induktivitäten und wie diese Einflüsse mathematisch modelliert und optimiert werden können.

2.1.2 Vorwärtspropagation in Neuronalen Netzwerken

Die Vorwärtspropagation ist ein wesentlicher Prozess in neuronalen Netzwerken, der die Übertragung von Eingabedaten durch die Netzwerkarchitektur ermöglicht, um die Ausgabe zu erzeugen [11]. Sie ist eine Abfolge von mathematischen Operationen, die Gewichtungen, Biases und Aktivierungsfunktionen beinhalten [4]. Dieser Abschnitt zielt darauf ab, die Schlüsselemente und mathematischen Grundlagen zu diskutieren, die die Vorwärtspropagation beeinflussen und stellt aktuelle Forschungsergebnisse und praktische Implikationen vor.

Schicht-für-Schicht-Propagation

Die Vorwärtspropagation beginnt bei den Eingabedaten X , bezeichnet als $A^{[0]}$. In jeder Schicht wird aus dem Ausgang $A^{[l-1]}$ der vorherigen Schicht ein Vektor $Z^{[l]}$ berechnet. Die Werte für jede nachfolgende Schicht $A^{[l]}$ werden entsprechend der Gleichung (2.1) ermittelt. Dieser Prozess wiederholt sich von Schicht zu Schicht und bildet das Kernstück der Vorwärtspropagation.

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \quad (2.1)$$

Aktivierungsfunktionen

Aktivierungsfunktionen in neuronalen Netzen spielen eine entscheidende Rolle, indem sie die gewichtete Summe $Z^{[l]}$ in einen aktivierten Ausgabewert $A^{[l]}$ umwandeln. Diese Transformation wird typischerweise durch σ dargestellt, wie in der folgenden Gleichung gezeigt:

$$A^{[l]} = \sigma(Z^{[l]}) \quad (2.2)$$

ReLU-Aktivierungsfunktion Die ReLU (Rectified Linear Unit) Aktivierungsfunktion, dargestellt in Abbildung 2.1b, ist eine der am häufigsten verwendeten Aktivierungsfunktionen in neuronalen Netzen. Sie wird definiert als $f(x) = \max(0, x)$ und ist besonders effektiv, um Nichtlinearitäten in den Daten zu modellieren.

$$f(x) = \max(0, x) \quad (2.3)$$

Softmax-Aktivierungsfunktion Die Softmax-Funktion ist eine Aktivierungsfunktion, die häufig in der Ausgabeschicht von Klassifizierungsnetzwerken verwendet wird. Sie wandelt einen Vektor von reellen Zahlen in Wahrscheinlichkeiten um. Die Formel für die Softmax-Funktion ist:

$$\text{Softmax}(z) = \frac{e^z}{\sum e^z} \quad (2.4)$$

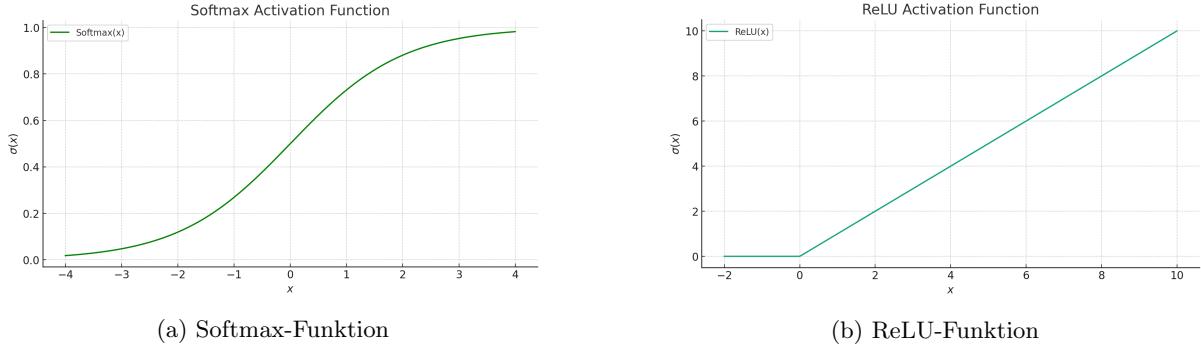


Figure 2.1: Vergleich der Softmax- und ReLU-Aktivierungsfunktionen.

Hierbei steht e^z für den Exponentialwert jedes Elements im Vektor z , und der Nenner ist die Summe aller Exponentialwerte im Vektor. Diese Umwandlung stellt sicher, dass die Ausgaben des Netzwerks als Wahrscheinlichkeiten interpretiert werden können, wobei die Summe aller Wahrscheinlichkeiten 1 ergibt.

Gewichtsmatrix $W^{[l]}$ und Bias-Vektor $b^{[l]}$

Die Gewichtsmatrix für die Schicht l wird als $W^{[l]}$ bezeichnet, und $b^{[l]}$ ist der Bias-Vektor für dieselbe Schicht [6]. Diese Parameter werden während des Backpropagation-Prozesses trainiert, um den Fehler zwischen der vorhergesagten und der tatsächlichen Ausgabe zu minimieren [1].

$$A^{[l]} = \sigma \left(\begin{pmatrix} w_{1,1}^{[l-1,l]} & w_{1,2}^{[l-1,l]} & \dots & w_{1,m}^{[l-1,l]} \\ w_{2,1}^{[l-1,l]} & w_{2,2}^{[l-1,l]} & \dots & w_{2,m}^{[l-1,l]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1}^{[l-1,l]} & w_{n,2}^{[l-1,l]} & \dots & w_{n,m}^{[l-1,l]} \end{pmatrix} \begin{pmatrix} A_1^{[l-1]} \\ A_2^{[l-1]} \\ \vdots \\ A_m^{[l-1]} \end{pmatrix} + \begin{pmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_n^{[l]} \end{pmatrix} \right) \quad (2.5)$$

Matrix-Vektor-Operationen und Parallelität

Die Gleichung (2.5) illustriert die Verarbeitungsschritte während der Vorwärtspropagation in neuronalen Netzwerken. Eine besondere Stärke dieser Darstellung ist die Visualisierung der Matrix-Vektor-Operationen. Diese Operationen sind von Natur aus parallel, was bedeutet, dass sie mehrere Berechnungen gleichzeitig durchführen können. Dies zeigt sich besonders in der Matrixmultiplikation, bei der jede Zeile der Gewichtsmatrix gleichzeitig mit dem Aktivierungsvektor multipliziert wird. In modernen Computerarchitekturen, insbesondere bei Grafikprozessoren (GPUs), kann diese inhärente Parallelität der Matrix-Operationen effektiv ausgenutzt werden, um die Berechnungsgeschwindigkeit erheblich zu steigern. Das bedeutet, dass die Berechnungen von $Z^{[l]}$ und $A^{[l]}$ simultan für alle Neuronen in Schicht l durchgeführt werden können. Dies ist ein entscheidender Vorteil, insbesondere in tiefen Netzwerken mit vielen Schichten und Neuronen. Die Verwendung der Matrixnotation erleichtert nicht nur das Verständnis der Struktur und des Arbeitsablaufs eines neuronalen Netzwerks, sondern hebt auch die Möglichkeit hervor, leistungsstarke parallele Hardwarearchitekturen effizient zu nutzen.

2.1.3 Einführung in die Backpropagation

Backpropagation, oder auch "Rückpropagation des Fehlers", ist ein Optimierungsverfahren, das hauptsächlich in der Ausbildung von künstlichen neuronalen Netzwerken verwendet wird. Das grundlegende Ziel der Backpropagation ist es, den Fehler, der während des Trainingsprozesses an der Ausgabeschicht berechnet wird, rückwärts durch das Netzwerk zu propagieren und durch Änderung der Gewichte zu minimieren. Dies ermöglicht die effiziente Anpassung der Gewichtungen und Bias-Werte des Netzwerks, um die Leistung zu verbessern.

Das Grundprinzip In einem neuronalen Netzwerk wird die Ausgabe durch eine Kette von Transformationen berechnet, die jeweils durch eine Schicht von Neuronen repräsentiert werden. Jedes Neuron in einer Schicht nimmt die Ausgaben der vorherigen Schicht auf, führt eine gewichtete Summe durch und wendet eine Aktivierungsfunktion an. Die endgültige Ausgabe wird dann mit dem tatsächlichen Zielwert

verglichen, um einen Fehlerwert zu ermitteln. Dieser Fehler wird verwendet, um die Gewichtungen und Bias-Werte im Netzwerk so anzupassen, dass der Fehler minimiert wird.

Die Bedeutung der Kostenfunktion Die Kostenfunktion C_0 , oft als Verlustfunktion bezeichnet, quantifiziert den Fehler zwischen den vorhergesagten Ausgaben und den tatsächlichen Zielwerten. Die Minimierung dieser Kostenfunktion ist das Hauptziel während des Trainingsprozesses, und Backpropagation ist das Schlüsselwerkzeug, das dabei hilft.

Rückpropagation in Neuronalen Netzwerken Zur Minimierung der Kostenfunktion C_0 , die wie folgt definiert ist:

$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{[L]} - y_j)^2 \quad (2.6)$$

wird die Technik der Rückpropagation verwendet. Das grundlegende Konzept besteht darin, den Fehler von der Ausgabeschicht rückwärts durch das Netzwerk zu propagieren. In Gleichung 2.6 haben wir beispielhaft einen quadratischen Fehler als Kostenfunktion verwendet.

[12]

Der Gradient des Fehlers in der Ausgabeschicht Die partielle Ableitung der Kostenfunktion C_0 in Bezug auf den Ausgabewert $a_j^{[L]}$ wird durch folgende Formel definiert:

$$\frac{\partial C_0}{\partial a_j^{[L]}} = 2(a_j^{[L]} - y_j) \quad (2.7)$$

Dieser Ausdruck stellt den Gradienten des Fehlers in der Ausgabeschicht dar. Er gibt an, wie empfindlich die Kostenfunktion C_0 auf Änderungen in $a_j^{[L]}$ reagiert. Dieser Wert wird zur Anpassung der Gewichtungen des Netzwerks während des Trainingsprozesses verwendet.

Fehler Rückpropagieren Die Ableitungen der Aktivierungsfunktion $a_j^{[L]}$ und der linearen Kombination $z_j^{[L]}$ sind:

$$\begin{aligned} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} &= \sigma'(z_j^{[L]}) \\ \frac{\partial z_j^{[L]}}{\partial w_{jk}^{[L]}} &= a_k^{[L-1]} \end{aligned}$$

Kettenregel Anwenden Nun kombinieren wir alle diese Teile mit der Kettenregel:

$$\frac{\partial C_0}{\partial w_{jk}^{[L]}} = \frac{\partial C_0}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial w_{jk}^{[L]}}$$

Durch Anwendung der Kettenregel erhalten wir:

$$\frac{\partial C_0}{\partial w_{jk}^{[L]}} = 2(a_j^{[L]} - y_j) \cdot \sigma'(z_j^{[L]}) \cdot a_k^{[L-1]}$$

Gradienten der Kostenfunktion Der Gradient der Kostenfunktion C_0 in Bezug auf alle Gewichtungen wird durch folgende Matrix dargestellt:

$$\nabla W^{[L]} C_0 = \left(2(a^{[L]} - y) \odot \sigma'(Z^{[L]}) \right) A^{[L-1]T}$$

Die Gradientenmatrix für die Gewichtungen in Bezug auf die Kostenfunktion C_0 kann in Matrixform dargestellt werden:

$$\nabla_{W^{[L]}} C_0 = \begin{pmatrix} 2(a_1^{[L]} - y_1) \cdot \sigma'(z_1^{[L]}) \cdot a_1^{[L-1]} & \dots & 2(a_1^{[L]} - y_1) \cdot \sigma'(z_1^{[L]}) \cdot a_m^{[L-1]} \\ 2(a_2^{[L]} - y_2) \cdot \sigma'(z_2^{[L]}) \cdot a_1^{[L-1]} & \dots & 2(a_2^{[L]} - y_2) \cdot \sigma'(z_2^{[L]}) \cdot a_m^{[L-1]} \\ \vdots & \ddots & \vdots \\ 2(a_n^{[L]} - y_n) \cdot \sigma'(z_n^{[L]}) \cdot a_1^{[L-1]} & \dots & 2(a_n^{[L]} - y_n) \cdot \sigma'(z_n^{[L]}) \cdot a_m^{[L-1]} \end{pmatrix}$$

Diese Matrix gibt die Änderungsrate der Kostenfunktion C_0 in Bezug auf das entsprechende Gewicht an. [12]

Schlussfolgerungen In diesem Kapitel wurde die Backpropagation-Technik ausführlich behandelt, ein Verfahren zur Optimierung von neuronalen Netzwerken. Mittels mathematischer Formeln wurde illustriert, wie der Fehler in der Ausgabeschicht des Netzwerks rückwärts propagiert wird, um eine Kostenfunktion C_0 zu minimieren. Besonders relevant ist die Anwendung der Kettenregel, die es erlaubt, die Änderungsrate der Kostenfunktion C_0 in Bezug auf die Gewichtungen zu berechnen. Diese Informationen sind entscheidend für die Anpassung der Gewichtungen und somit für die Verbesserung der Netzwerkleistung. Es sei darauf hingewiesen, dass nicht die einzigen Methoden sind, in der Praxis können je nach Anforderungen und Anwendungsfall auch andere Fehlerbestimmungsmethoden, Aktivierungsfunktionen und Netzwerkarchitekturen verwendet werden.

2.1.4 Optimierungstechniken

Die Wahl des Optimierers hat einen erheblichen Einfluss auf die Leistung eines neuronalen Netzwerks. Hier stellen wir einige gängige Optimierungsalgorithmen vor und erläutern den mathematischen Hintergrund des ADAM-Optimierers.

Stochastic Gradient Descent (SGD) Der Stochastic Gradient Descent (SGD) ist ein fundamentales Optimierungsverfahren. Es unterscheidet sich von traditionellen Gradientenabstiegsverfahren dadurch, dass es nicht den gesamten Datensatz verwendet, um die Gradienten der Kostenfunktion zu jedem Schritt zu berechnen. Stattdessen nutzt der SGD nur eine zufällig ausgewählte Teilmenge oder sogar nur eine einzige Dateninstanz, um die Gradienten zu schätzen und die Modellgewichtungen anzupassen. Insbesondere im Rahmen des Reinforcement Learnings, wo zu jedem Zeitpunkt lediglich ein Teildatensatz verfügbar ist, erweist sich der SGD als besonders geeignet. Die stochastische Natur dieses Verfahrens kann helfen, aus lokalen Minima herauszukommen und zu einer besseren Konvergenz des Modells beizutragen. [7]

Momentum Momentum berücksichtigt sowohl den aktuellen Gradienten als auch die vorherigen Gradienten, um die Gewichtungen zu aktualisieren. Dadurch wird eine Art "Schwung" erzeugt, der dem Optimierer hilft, lokale Minima zu überwinden.

RMSprop Dieser Optimierer passt die Lernrate während des Trainings dynamisch an. Er verwendet den gleitenden Mittelwert der quadrierten Gradienten, um die Gewichtungen zu aktualisieren.

ADAM (Adaptive Moment Estimation) ADAM kombiniert die Vorteile von Momentum und RMSprop. Die Gewichtsaktualisierung in ADAM ist durch die folgende Gleichung definiert:

$$\theta_{t+1} = \theta_t - \alpha \times \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (2.8)$$

wobei \hat{m}_t und \hat{v}_t Schätzungen des ersten und zweiten Moments der Gradienten sind. Sie werden wie folgt berechnet:

$$\hat{m}_t = \frac{1}{1 - \beta_1^t} \times m_t \quad (2.9)$$

$$\hat{v}_t = \frac{1}{1 - \beta_2^t} \times v_t \quad (2.10)$$

$$m_t = \beta_1 \times m_{t-1} + (1 - \beta_1) \times g_t \quad (2.11)$$

$$v_t = \beta_2 \times v_{t-1} + (1 - \beta_2) \times g_t^2 \quad (2.12)$$

wobei g_t der Gradient bei Schritt t ist, β_1 und β_2 Hyperparameter sind, und α die Lernrate ist. [7]

Schlussfolgerungen Der Optimierer ist entscheidend für die Effizienz des Trainingsprozesses eines neuronalen Netzwerks. Der ADAM-Optimierer ist insbesondere eine ausgezeichnete Wahl für viele Anwendungen, da er sowohl die Vorteile von Momentum als auch von RMSprop kombiniert.

2.1.5 Vermeidung von Overfitting in Neuronalen Netzen

Overfitting ist ein kritisches Problem in der Anwendung neuronaler Netze, da es die Generalisierbarkeit des Modells auf neue Daten beeinträchtigt. Es tritt auf, wenn ein neuronales Netzwerk die Trainingsdaten "zu gut" lernt, einschließlich des Rauschens und der Ausreißer, und dadurch schlecht auf neue, unbekannte Daten generalisiert. Ein klares Anzeichen für Overfitting ist, wenn die Leistung auf dem Trainingsdatensatz steigt, während die Leistung auf dem Validierungsdatensatz stagniert oder abnimmt. [7]

Strategien zur Vermeidung von Overfitting:

1. **Regularisierung:** Es gibt verschiedene Formen der Regularisierung wie L1 und L2, die durch die Einführung eines Regularisierungsterms in die Kostenfunktion die Modellkomplexität einschränken.
2. **Dropout:** Einige Neuronen werden während des Trainings zufällig "ausgeschaltet", um das Modell daran zu hindern, sich zu stark auf bestimmte Merkmale zu verlassen.
3. **Daten Augmentation:** Durch kleine Veränderungen der Eingabedaten kann der Trainingsdatensatz erweitert und die Generalisierungsfähigkeit des Modells verbessert werden.
4. **Frühzeitiges Stoppen:** Das Training wird gestoppt, sobald die Leistung auf dem Validierungsdatensatz sich nicht mehr verbessert.
5. **Stochastischer Gradientenabstieg:** Dieser Algorithmus kann als eine Form der Regularisierung betrachtet werden.
6. **Vereinfachung der Architektur:** Reduzierung der Anzahl der Neuronen oder Schichten im Netzwerk.
7. **Ensemble-Methoden:** Mehrere Modelle werden trainiert und ihre Vorhersagen kombiniert, um eine robustere Vorhersage zu erhalten. Dies kann als eine Form der Regularisierung betrachtet werden.

Regularisierung

Die Kostenfunktion $J(\theta)$ wird durch die Hinzufügung eines Regularisierungsterms erweitert:

$$J(\theta) = \text{Ursprüngliche Kosten} + \frac{\lambda}{2m} \sum_{i=1}^n \theta_i^2$$

Hierbei ist λ der Regularisierungsparameter und θ sind die Gewichtungen des Modells. [7]

Dropout

Während des Trainings wird eine binäre Maske M mit einer Wahrscheinlichkeit p generiert und die Ausgabe eines jeden Layers wird mit dieser Maske multipliziert.

Daten Augmentation

Der Trainingsdatensatz D wird durch Transformationen T erweitert, um die Generalisierungsfähigkeit des Modells zu verbessern:

$$D' = D \cup T(D)$$

Frühzeitiges Stoppen

Das Training wird gestoppt, sobald die Validierungsfehlerfunktion $J_{\text{valid}}(\theta)$ anfängt zu steigen oder nicht mehr abnimmt.

Vereinfachung der Architektur

Die Anzahl der Schichten L oder Neuronen N wird reduziert, d.h., $L' < L$ oder $N' < N$.

Ensemble-Methoden

Mehrere Modelle f_1, f_2, \dots, f_n werden trainiert und ihre Vorhersagen werden gemittelt oder anderweitig kombiniert:

$$f_{\text{Ensemble}}(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$$

2.1.6 Vanishing-Gradient-Problem

Neuronale Netze, insbesondere tiefe Architekturen, sind anfällig für das Problem der verschwindenden oder explodierenden Gradienten [1]. Während des Backpropagation-Prozesses können die Gradienten entweder zu klein werden, was als "Verschwinden" bezeichnet wird, oder zu groß, was als "Explodieren" bezeichnet wird. Beide Fälle erschweren das Training erheblich und können dazu führen, dass das Netzwerk nicht konvergiert oder instabil wird.

Warum ist das ein Problem?

Verschwindende Gradienten können dazu führen, dass die Gewichtsaktualisierungen für die ersten Schichten im Netzwerk minimal sind, wodurch das Netzwerk ineffizient trainiert wird. Explodierende Gradienten können dazu führen, dass die Gewichtsaktualisierungen zu groß werden, was das Training des Netzwerks destabilisiert [1].

Anpassung der Lernrate und optimierte Backpropagation-Techniken

Eine Möglichkeit besteht darin, die Lernrate α dynamisch während des Trainings anzupassen. Wenn der Gradient ∇J einen bestimmten Schwellenwert überschreitet, könnte die Lernrate reduziert werden:

$$\alpha = \alpha \times \text{decay}$$

Gradient Clipping

Das sogenannte "Gradient Clipping" begrenzt den Gradienten ∇J auf einen maximalen Wert C [3]:

$$\nabla J = \min(\nabla J, C)$$

Architektur- und Ressourcenüberlegungen

Die Wahl der Architektur und der Ressourcen spielt auch eine Rolle bei der Bewältigung von Gradientenproblemen. Ein gut durchdachtes Design kann helfen, das Problem der verschwindenden und explodierenden Gradienten zu mildern [1].

ReLU-Aktivierungsfunktion

Die ReLU-Aktivierungsfunktion ist eine der am häufigsten verwendeten Aktivierungsfunktionen in neuronalen Netzen. Mathematisch wird die ReLU-Funktion wie folgt definiert:

$$f(x) = \max(0, x)$$

Warum ReLU beim Vanishing-Gradient-Problem hilft: Die ReLU-Aktivierungsfunktion hat mehrere Eigenschaften, die sie im Umgang mit dem Vanishing-Gradient-Problem nützlich machen:

- **Lineare Aktivierung im positiven Bereich:** ReLU ist im positiven Bereich linear, wodurch der Gradient konstant gleich 1 bleibt. Dies erleichtert das Backpropagation-Verfahren, da der Gradient nicht beim Durchlaufen dieses Neurons verschwindet.
- **Sparsamkeit:** ReLU-Aktivierung führt dazu, dass einige Neuronen einen Ausgang von Null haben. Dies fördert die Sparsamkeit des Netzes, was dazu beitragen kann, Overfitting zu reduzieren.

- **Einfache Berechnung:** Im Vergleich zu anderen Aktivierungsfunktionen wie Sigmoid oder Tanh ist die ReLU-Funktion rechentechnisch weniger aufwendig, was zu schnelleren Trainingsschritten führt.
- **Nichtlinearität:** Obwohl sie in einem Bereich linear ist, ist die ReLU-Funktion insgesamt nicht-linear. Dies ermöglicht dem Netzwerk, komplexe Muster zu erfassen, während gleichzeitig das Vanishing-Gradient-Problem gemildert wird.

Überwachung und manuelle Anpassung

Es ist oft notwendig, das Verhalten des Netzwerks während des Trainings zu überwachen. Wenn Anzeichen für verschwindende oder explodierende Gradienten festgestellt werden, kann eine manuelle Anpassung der Architektur oder der Hyperparameter erforderlich sein.

Anpassungen im aktuellen Ansatz

In der vorliegenden Arbeit wurden bereits mehrere spezifische Methoden zur Bewältigung des Problems implementiert:

- **Normalisierung des Rewards:** Der Reward-Wert R wurde angepasst:

$$R_{\text{norm}} = \frac{R - \min(R)}{\max(R) - \min(R)}$$

- **Normalisierung der Eingänge:** Die Eingangsdaten X wurden normalisiert:

$$X_{\text{norm}} = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

- **Verwendung von tanh:** Die Aktivierungsfunktion tanh wurde verwendet, um die Ausgabe $f(x)$ zu normalisieren:

$$f(x) = \tanh(x)$$

- **ReLU-Aktivierungsfunktion:** Die Verwendung der ReLU-Aktivierungsfunktion (Rectified Linear Unit) kann dazu beitragen, das Vanishing-Gradient-Problem zu mildern und die Netzwerkeffizienz zu verbessern, was indirekt Overfitting verhindern kann.

2.1.7 Bedeutung der Lernrate

Die Lernrate ist ein kritischer Hyperparameter, der die Geschwindigkeit der Modellanpassung steuert. Wie aus der Literatur bekannt ist, kann eine falsch gewählte Lernrate den Lernprozess verlangsamen oder sogar zu einer fehlgeschlagenen Konvergenz führen [11]. In ähnlicher Weise betont Heaton, dass die Wahl der Lernrate und des Momentums entscheidend für die Leistung des Trainings ist [6].

subsectionHerausforderungen und Lösungsansätze Die Einstellung der Lernrate ist häufig ein Prozess des Ausprobierens, und die beste Methode zur Ermittlung der optimalen Lernrate ist nicht eindeutig. Goodfellow et al. erklären, dass die Lernrate das effektive Kapazitätsmodell in einer komplexeren Weise steuert als andere Hyperparameter [5]. Daher ist es von entscheidender Bedeutung, sowohl das Training als auch den Testfehler zu überwachen, um zu diagnostizieren, ob das Modell unter- oder überangepasst ist.

2.1.8 Bedeutung der Batch-Größe

Die Batch-Größe ist ein weiterer wichtiger Hyperparameter in der Ausbildung von neuronalen Netzwerken. Sie beeinflusst nicht nur die Rechenzeit, sondern auch die Modellgenauigkeit. Die Wahl der optimalen Batch-Größe ist ein Kompromiss zwischen Recheneffizienz und Modellqualität. Im Folgenden werden einige relevante Aspekte diskutiert.

Berechnungseffizienz

- **Große Batches:** Eine große Batch-Größe ermöglicht es, viele Datenpunkte gleichzeitig durch das Netzwerk zu leiten, wodurch die Recheneffizienz erhöht wird. Minibatch-Stochastic Gradient Descent (SGD) ist in der Regel 10-mal schneller als Full-Batch-SGD, insbesondere bei Verwendung von parallelen Vektoroperationen auf modernen CPU- oder GPU-Architekturen [11].
- **Kleine Batches:** Bei einer kleinen Batch-Größe wird weniger Speicher benötigt, und das Modell kann schneller auf den Daten trainieren, auch wenn dies zu einer höheren Varianz im Gradienten führen kann [10].

Genauigkeit und Overfitting

- **Große Batches:** Größere Batches bieten in der Regel genauere Schätzungen des Gradienten, können jedoch zum Overfitting neigen, wenn nicht richtig reguliert [5].
- **Kleine Batches:** Die Verwendung kleinerer Batches kann eine regulierende Wirkung haben und kann das Generalisierungsvermögen des Modells verbessern [5].

Hardware-Beschränkungen

Die Größe der Batches kann auch durch die verfügbare Hardware begrenzt sein. Insbesondere ist der Arbeitsspeicher oft der begrenzende Faktor [1].

Empfehlungen

Es ist eine gängige Praxis, Batch-Größen in Potenzen von 2 zu wählen, da dies häufig zu einer besseren Laufzeiteffizienz führt, besonders auf GPUs [5].

2.2 Reinforcement Learning

2.2.1 Einleitung

Nachdem wir die grundlegenden Konzepte neuronaler Netze und die Mechanismen von Forward- und Backpropagation ausführlich behandelt haben, wenden wir uns nun dem Thema Reinforcement Learning (RL) zu. Die zuvor erklärten neuronalen Netze dienen als Grundlage für RL und werden insbesondere als Funktionenapproximatoren verwendet, um Agenten zu simulieren und zu trainieren. In diesem Abschnitt beschäftigen wir uns mit den Kernkonzepten und Herausforderungen des Reinforcement Learning.

Grundlagen des Reinforcement Learning Reinforcement Learning ist ein maschinelles Lernverfahren, bei dem ein Agent in einer Umgebung agiert, um ein bestimmtes Ziel zu erreichen. Im Kern des Prozesses stehen zwei Hauptkomponenten: der Agent und die Umgebung (auch als Simulation bezeichnet). Der Agent nimmt den aktuellen Zustand der Umgebung wahr und trifft auf dieser Grundlage Entscheidungen, die als Aktionen bezeichnet werden [10]. Diese Aktionen werden an die Umgebung übermittelt, welche daraufhin in einen neuen Zustand übergeht und dem Agenten eine Belohnung (engl. Reward) ausgibt.

Dieser Prozess kann in zwei Haupttypen von RL-Verfahren unterteilt werden: das modellbasierte und das modellfreie RL [11]. Bei modellbasierten Verfahren verwendet der Agent ein Übergangsmodell der Umgebung, um die Belohnungssignale zu interpretieren und Entscheidungen zu treffen. Im Gegensatz dazu lernt der Agent bei modellfreien Verfahren eine direktere Darstellung des Verhaltens, oft durch das Erlernen einer Q-Funktion, die die Qualität einer Aktion in einem bestimmten Zustand bewertet [11].

Es ist wichtig zu betonen, dass RL-Agenten durch direkte Interaktion mit ihrer Umgebung lernen, ohne die Notwendigkeit einer beaufsichtigenden Instanz oder eines vollständigen Modells der Umgebung [12]. Jede Interaktion, die als Zeitschritt bezeichnet wird, ermöglicht dem Agenten, seine Leistung durch Versuch und Irrtum zu verbessern [10].

Die Funktionsweise von RL kann als zyklisch beschrieben werden: Der Agent beobachtet die Umgebung, trifft eine Entscheidung, die Umgebung ändert ihren Zustand und gibt eine Belohnung zurück, und der neue Zustand wird dem Agenten wieder präsentiert. Durch diesen zyklischen Prozess lernt der Agent, optimale Entscheidungen zu treffen, um das gestellte Ziel zu erreichen.

Herausforderungen im RL RL stellt verschiedene Herausforderungen dar, die es von anderen maschinellen Lernmethoden unterscheiden. Ein wichtiger Aspekt ist die Sequenzialität der Entscheidungen. Die Agenten müssen nicht nur einmalige Entscheidungen treffen, sondern auch eine Sequenz von Aktionen planen, um ein optimales Ergebnis zu erzielen [10]. Darüber hinaus müssen die Agenten in einer Umgebung mit Unsicherheit und Unvollkommenheit agieren, was den Einsatz von Methoden wie dem *Markov-Entscheidungsprozess* erforderlich macht [3].

Erweiterte Methoden und DDPG Der im nächsten Abschnitt behandelte DDPG (Deep Deterministic Policy Gradients) Algorithmus stellt eine fortschrittliche Methode in der Reinforcement-Learning-Domäne dar. Er übertrifft traditionellere Ansätze wie Policy Gradient oder Deep Q Networks (DQN) durch seine effizientere und schnellere Fähigkeit zur Identifizierung optimaler Policies [10].

2.2.2 Markovsche Prozesse und deren Rolle im Reinforcement Learning

Die Grundlage für viele Reinforcement Learning-Verfahren sind Markov-Entscheidungsprozesse (MDPs). Diese MDPs stellen eine stochastische Version des optimalen Steuerungsproblems dar, welches von Bellman eingeführt wurde [12]. Grundlegend geht es bei einem MDP um Entscheidungen, die in einem Zustand getroffen werden, um eine maximale Belohnung in der Zukunft zu erzielen.

Grundlagen der Markovschen Prozesse Ein Markov-Entscheidungsprozess ist charakterisiert durch:

- Eine Menge von Zuständen, in denen sich das System befinden kann.
- Eine Menge von Aktionen, die in jedem Zustand durchgeführt werden können.
- Eine Belohnungsfunktion, die die sofortige Belohnung für den Übergang von einem Zustand in einen anderen definiert.
- Eine Übergangsfunktion, die die Wahrscheinlichkeit beschreibt, von einem Zustand in einen anderen überzugehen, nachdem eine bestimmte Aktion ausgeführt wurde.

MDPs basieren auf der Markov-Eigenschaft, die besagt, dass die Zukunft nur vom gegenwärtigen Zustand abhängt und nicht von den vorhergehenden Zuständen. Das bedeutet, dass das System keine Erinnerung an frühere Zustände hat und die Zukunft unabhängig von der Vergangenheit ist, solange der aktuelle Zustand bekannt ist.

Ein entscheidendes Konzept in MDPs ist die sogenannte "Policy", eine Strategie, die festlegt, welche Aktion in jedem Zustand ausgeführt werden soll, um die kumulative Belohnung über die Zeit zu maximieren [12].

Ein einfacher Markov-Prozess kann als Zustandsdiagramm dargestellt werden. Im folgenden Beispiel gibt es drei Zustände: S_1 , S_2 , und S_3 .

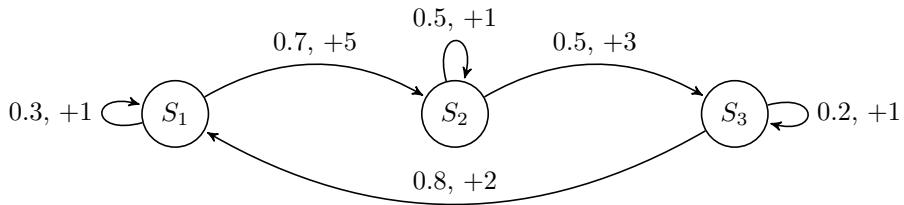


Figure 2.2: Beispiel für einen einfachen Markov-Prozess mit Übergangsbelohnungen

Legende

- Knoten: Zustände (S_1 , S_2 , S_3 usw.)
- Kanten: Erste Zahl ist die Übergangswahrscheinlichkeit, zweite Zahl ist die Belohnung für den Übergang

Im Diagramm repräsentieren die Knoten die Zustände und die Kanten repräsentieren die Übergangswahrscheinlichkeiten zwischen den Zuständen.

Anwendung von Markovschen Prozessen im Reinforcement Learning Die Anwendung von MDPs im Reinforcement Learning ermöglicht es Agenten, optimale Strategien für komplexe Aufgaben zu erlernen. Dies geschieht durch die Interaktion des Agenten mit der Umgebung und die Anpassung seiner Strategie basierend auf den erhaltenen Belohnungen. Methoden wie "Policy Iteration" und "Value Iteration" werden verwendet, um optimale Policies zu ermitteln [11].

Ein erweitertes Konzept von MDPs sind die POMDPs (Partially Observable Markov Decision Processes), bei denen der Agent nicht immer den genauen Zustand der Umgebung kennt. Dies führt zu komplexeren Strategien, bei denen der Agent versucht, Unsicherheiten durch Informationsbeschaffung zu reduzieren [10].

Abschließend können wir sagen, dass Markov-Entscheidungsprozesse ein zentrales Konzept im Reinforcement Learning darstellen, das es Agenten ermöglicht, in einer Vielzahl von Umgebungen effektive Strategien zu entwickeln.

2.2.3 Das Multi-Armed Bandit Problem im Kontext der Degradation

Das Multi-Armed Bandit Problem, häufig als "Bandit Problem" abgekürzt, stellt ein klassisches Dilemma in der Entscheidungsfindung unter Unsicherheit dar. Ursprünglich aus der Glücksspielwelt stammend, bei dem es darum geht, den gewinnbringendsten von mehreren Spielautomaten (den "Banditen") auszuwählen, hat dieses Problem vielfältige Anwendungen in den Bereichen künstliche Intelligenz und Maschinenlernen gefunden. Die Herausforderung liegt hierbei im Balancieren zwischen Exploration (Erkunden neuer oder weniger bekannter Aktionen) und Exploitation (Nutzen der besten bekannten Aktionen) [12].

In Bezug auf Degradationsvorgänge wird das Bandit Problem herangezogen, um den kontinuierlichen Verfall von Zuständen zu modellieren. Die Degradation, die sich über eine längere Zeitspanne erstreckt, lässt sich mittels eines markovschen Prozesses darstellen. Charakteristisch hierfür ist, dass trotz Durchführung einer Aktion der Zustandsübergang nur minimal ist, sodass es so wirkt, als würde das System in denselben Zustand zurückkehren.

Der gewählte Ansatz beinhaltet eine Sequenz von Zuständen S_1, \dots, S_N . In jedem dieser Zustände gibt es mehrere Aktionen, die den jeweiligen PID-Koeffizienten in diesem spezifischen Zustand repräsentieren. Interessanterweise wird durch die Modellierung der markovsche Prozess in ein Multi-Armed Bandit Problem überführt, sodass Zustände von S_0, \dots, S_N existieren. Bei jeder Aktion innerhalb eines Zustands, charakterisiert durch drei PID-Parameter, gelangt das System wieder in denselben Zustand [11].



Figure 2.3: Erweiterte Darstellung des Multi-Armed Bandit Problems mit Zustandsaktionen von 1 bis N

Legende

- Kanten: Erste Zahl p symbolisiert die Wahrscheinlichkeit der Aktion a
- zweite Zahl r die Belohnung der Aktion.

2.2.4 Markow-Entscheidungsprozesse und die Bellman-Gleichung

Markow-Entscheidungsprozesse sind grundlegend für das Verständnis von Entscheidungsproblemen in stochastischen Umgebungen. Sie werden oft modelliert mit Zuständen s , Aktionen a , Zustandsübergängen $P(s'|s, a)$ und Belohnungen $R(s, a, s')$. Die Bellman-Gleichung dient als Eckpfeiler für die Lösung von MDPs und kann wie folgt geschrieben werden:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma U(s')]$$

Diese Gleichung erklärt uns im Wesentlichen den Nutzen eines Zustands s unter einer optimalen Policy. Sie repräsentiert die erwartete Summe der Belohnungen für einen Agenten, der von Zustand s startet und sich in der Zukunft optimal verhält [11].

Wenn $\gamma = 0$: Übergang zu Banditenproblemen

Der Diskontfaktor γ ist wichtig, da er sofortige und zukünftige Belohnungen ausbalanciert. Wenn jedoch $\gamma = 0$, geraten wir in ein Szenario, das eher den Multi-Armed-Bandit-Problemen ähnelt, bei denen der Agent nur an sofortigen Belohnungen interessiert ist. In diesem Fall vereinfacht sich die Bellman-Gleichung zu:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s, a) [R(s, a, s')]$$

Hier berücksichtigt der Agent nicht den zukünftigen Nutzen der Zustände s' , in die er übergehen könnte. Er maximiert nur seine sofortige Belohnung, was das Problem zu einem Einzelschritt-Entscheidungsproblem macht [12].

2.2.5 Policy-Based Reinforcement Learning

Policy-Based Reinforcement Learning konzentriert sich auf das direkte Lernen einer Policy $\pi(s)$, die Zuständen s Aktionen a zuordnet. Diese Methoden, insbesondere Policy Gradient Ansätze, sind effektiv in Umgebungen mit diskreten und kontinuierlichen Aktionsräumen.[12].

Mathematische Grundlagen Die Aktualisierung der Policy-Parameter θ erfolgt anhand des Policy Gradient Ansatzes, wie in der folgenden Gleichung dargestellt:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} [\nabla_{\theta} \log \pi(a|s; \theta) \cdot R(s, a)] \quad (2.13)$$

Hierbei ist $J(\theta)$ die zu maximierende Zielfunktion, $R(s, a)$ die Belohnung und θ die Policy-Parameter [11]. Die Aktionen werden basierend auf der Wahrscheinlichkeitsverteilung der Policy, häufig modelliert durch eine Softmax-Funktion, ausgewählt [11], wie in Gleichung 2.3 dargestellt.

Visuelle Darstellung der Policy-Werte Ein Beispiel für eine trainierte Policy in einem einfachen Rasterumfeld zeigt Abbildung 2.4. Jede Zelle des Rasters repräsentiert einen Zustand mit einem zugeordneten Wert, der die erwartete zukünftige Belohnung unter dieser Policy anzeigt. Pfeile in den Zellen deuten die von der Policy gewählten Aktionen an, wobei die grünen Zellen eine positive Belohnung (Zielzustände) und die rote Zelle eine negative Belohnung (Strazfzustände) symbolisieren.

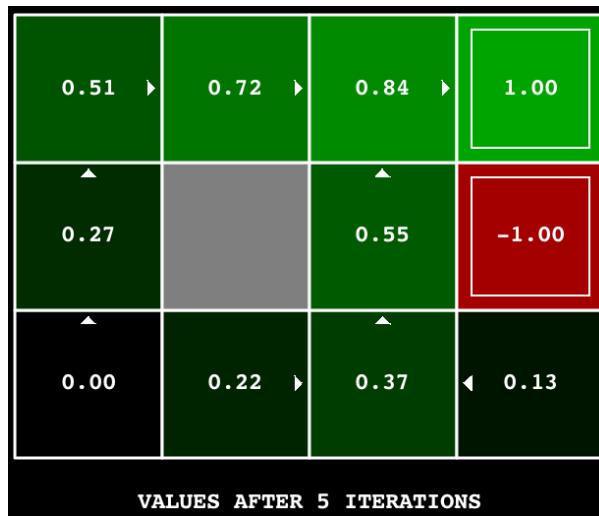


Figure 2.4: Trainierte Policy nach 5 Iterationen. Jede Zelle zeigt den geschätzten Wert des Zustands und die von der Policy empfohlene Aktion. Grüne Zellen kennzeichnen positive Belohnungen, die rote Zelle kennzeichnet eine negative Belohnung.

Pseudocode Die Aktualisierung der Policy-Parameter, wie in Gleichung 2.13 beschrieben, erfolgt nach dem folgenden Schema:

```

Initialisiere die Policy-Parameter  $\theta$ 
Setze die Lernrate  $\alpha$ 
for jede Episode do
    Initialisiere den Zustand  $s$ 
    while  $s$  kein terminaler Zustand ist do
        Wähle Aktion  $a$  basierend auf der aktuellen Policy  $\pi(s; \theta)$ 
        Führe Aktion  $a$  aus, beobachte Belohnung  $r$  und neuen Zustand  $s'$ 
        Aktualisiere die Policy-Parameter:
         $\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi(a|s; \theta) \cdot r$ 
         $s \leftarrow s'$  # Wechsel zum neuen Zustand
    end while
end for
```

Wahl der Aktion in Policy-Based-Methoden In Policy-Based Reinforcement Learning-Methoden erfolgt die Aktionsermittlung basierend auf einer Wahrscheinlichkeitsverteilung der aktuellen Policy $\pi(s; \theta)$, oft repräsentiert durch die Softmax-Funktion ,2.1a wie in Abbildung 2.4 illustriert [11]. Besonders effektiv sind diese Methoden in komplexen Umgebungen, wo lineare oder einfache Beziehungen zwischen Zuständen, Aktionen und Belohnungen fehlen [12]. Die direkte Optimierung der Policy ohne separate Value-Funktion bietet Vorteile, stellt jedoch eine Herausforderung dar, da eine umfangreiche Erkundung des Zustandsraums für die Konvergenz notwendig ist [10]. Ein häufiger Ansatz zur Aktionsermittlung in diskreten Räumen ist der Epsilon-Greedy-Algorithmus 2.14, der die Balance zwischen Exploration und Exploitation hält [12].

Einschränkungen von Policy-Based-Methoden Policy-Based Methoden im Reinforcement Learning, effektiv in diskreten und kontinuierlichen Aktionsräumen, stoßen in komplexen Umgebungen an Grenzen. Ihre Hauptherausforderung ist die benötigte umfangreiche Erkundung des Zustandsraums für die Konvergenz, was in kostspieligen oder riskanten Szenarien praktisch problematisch sein kann [12]. Zudem konvergieren sie langsamer als Value-Based-Methoden, besonders in Umgebungen mit vielen Zuständen oder hoher Dimensionalität, da die direkte Optimierung der Policy ohne konkrete Werteschätzungen aufwendiger ist [12]. Diese Methoden sind zwar theoretisch für kontinuierliche Aktionsräume geeignet, ihre Effektivität kann jedoch in der Praxis durch feingranulare Aktionen und schwer abschätzbare Konsequenzen begrenzt sein.

2.2.6 Q-Learning

Q-Learning ist ein modellfreier Algorithmus für das Erlernen der optimalen Entscheidungsfindung in der RL-Umgebungen. Der Algorithmus wurde 1989 von Chris Watkins in seiner Doktorarbeit eingeführt und hat sich als einer der beliebtesten Algorithmen im Bereich des RL-Umgebungen etabliert [10]. Anstelle eines Übergangsmodells wird eine .direkt aktualisiert, die die Qualität jeder möglichen Aktion in einem bestimmten Zustand repräsentiert [11].

Mathematische Grundlagen Die Aktualisierung der Q-Werte $Q(s, a)$ erfolgt anhand der Bellman-Gleichung:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

wobei s der aktuelle Zustand, a die gewählte Aktion, s' der neue Zustand, $R(s, a, s')$ die unmittelbare Belohnung, α die Lernrate und γ der Abschlagfaktor ist [11].

Pseudocode Der Pseudocode für den Q-Learning-Algorithmus könnte wie folgt aussehen:

```

Initialize  $Q(s, a)$  for all states and actions
Set parameters  $\alpha$  and  $\gamma$ 
for each episode do
    Initialize state  $s$ 
    while  $s$  is not a terminal state do
```

Choose action a based on policy (e.g., epsilon-greedy) derived from $Q(s, a)$

Take action a , observe reward r and new state s'

Update Q-value:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (r + \gamma \cdot \max(Q(s', a')) - Q(s, a))$$

$s \leftarrow s'$ # Move to the new state

end while

end for

Dieser Algorithmus benötigt keine Modellierung der Umgebung und ist daher besonders nützlich in komplexen oder unbekannten Umgebungen [10]. Es handelt sich um ein "Off-Policy"-Verfahren, was bedeutet, dass es die optimale Q-Funktion lernt, unabhängig von der während des Lernens verwendeten Richtlinie [10].

Die sogenannte Heatmap in Abbildung 2.6 visualisiert die Q-Werte, die der Agent in jedem Zustand der Rasterwelt zugewiesen hat. Die Farbgebung illustriert, wie Q-Learning durch Interaktion mit der Umgebung diese Werte optimiert, indem es Belohnungen maximiert, was durch die Bellman-Gleichung mathematisch fundiert ist. Dieser Prozess des stetigen Aktualisierens der Q-Werte, die die erwarteten zukünftigen Belohnungen für jede Aktion in jedem Zustand repräsentieren, zeigt die modellfreie Natur des Q-Learning-Algorithmus und seine Fähigkeit, optimale Strategien zu erlernen, auch wenn die genaue Dynamik der Umgebung unbekannt ist. Die in Abbildung 2.6 dargestellten Informationen ergänzen somit die theoretischen Konzepte, die in diesem Abschnitt diskutiert wurden, indem sie ein konkretes Beispiel für die Anwendung von Q-Learning in einer simulierten Umgebung bieten [7].

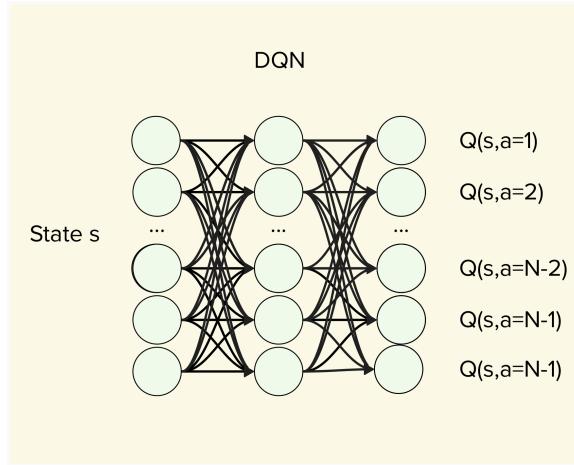


Figure 2.5: Schematische Darstellung eines Deep Q-Networks (DQN), das für RL-Umgebungen verwendet wird.

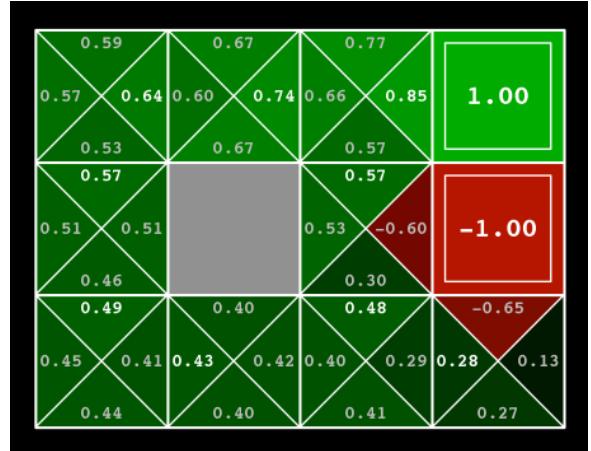


Figure 2.6: Darstellung einer Heatmap von Q-Werten in einer Rasterwelt.
[7]

Das ϵ -greedy Verfahren im RL-Umgebungen

$$a = \begin{cases} \underset{a}{\operatorname{argmax}} Q(s, a) & \text{mit Wahrscheinlichkeit } 1 - \epsilon, \\ \text{eine zufällige Aktion} & \text{mit Wahrscheinlichkeit } \epsilon. \end{cases} \quad (2.14)$$

Deep Q-Learning Der Kernunterschied zwischen Q-Learning und Deep Q-Learning (DQL) liegt in der Methodik der Approximation der Q-Werte, die im Zentrum beider Ansätze steht. Klassisches Q-Learning nutzt eine Q-Tabelle, um Werte für Zustands-Aktions-Paare zu speichern, was bei einer begrenzten Anzahl von Zuständen und Aktionen gut funktioniert. Jedoch wird diese Methode unpraktikabel in komplexen, hochdimensionalen Umgebungen. Hier kommt DQL ins Spiel, das neuronale Netze verwendet, um diese Werte effizient zu schätzen [10].

Wie Abbildung 2.5 veranschaulicht, ermöglicht die Verwendung von tiefen neuronalen Netzen in einem Deep Q-Network (DQN), eine effektive Handhabung von Situationen mit einer hohen Anzahl von Zuständen, die über das Fassungsvermögen einer herkömmlichen Q-Tabelle hinausgehen. Diese fortgeschrittenen Netzwerke sind in der Lage, direkt aus rohen visuellen Eingaben zu lernen und entwickeln im Laufe der Zeit eine verfeinerte Q-Funktion, welche die Entscheidungsfindung des Agenten verbessert.

Diese Verbesserung wird durch den Einsatz von Deep Learning-Techniken erreicht, die es dem DQN ermöglichen, abstrakte Repräsentationen für die Entscheidungsfindung zu lernen und zu nutzen, was eine signifikante Verbesserung gegenüber traditionellen Q-Learning-Techniken darstellt [3].

Exploration versus Exploitation Die Balance zwischen Exploration und Exploitation ist essentiell für die Effektivität von lernenden Agenten im Q-Learning, die sich im Laufe ihrer Interaktion mit der Umgebung stetig anpassen müssen.

Durch die Anwendung des ϵ -greedy Verfahrens 2.14 im RL-Umgebungen wird ein Kompromiss zwischen der Erforschung unbekannter Aktionen und der Ausnutzung bekannter Strategien gefunden. Dies ermöglicht es einem Agenten, sowohl Neues zu entdecken als auch bewährte Taktiken einzusetzen, um den unmittelbaren Gewinn zu maximieren [12].

Die Dynamik von ϵ spielt eine wichtige Rolle: Zu Beginn des Lernprozesses fördert ein höheres ϵ die Exploration. Mit zunehmender Erfahrung sollte ϵ jedoch sinken, was den Übergang zur gezielten Exploitation bewirkt [10].

Simulated Annealing erlaubt es, unter bestimmten Bedingungen auch Verschlechterungen des aktuellen Zustands zu akzeptieren. Dies wird durch die "Temperatur", einen Metaparameter, ermöglicht, der analog zur physikalischen Temperatur in der Metallurgie die Wahrscheinlichkeit für das Akzeptieren schlechterer Zustände bestimmt [11]. Im Metallurgieprozess bezeichnet Annealing den Vorgang, bei dem Metalle und Glas erhitzt und dann langsam abgekühlt werden, um eine stabile, kristalline Struktur zu erlangen [11].

Simulated Annealing beginnt mit einer hohen "Temperatur", die es dem Algorithmus ermöglicht, lokale Minima zu verlassen, indem temporär schlechtere Lösungen akzeptiert werden. Mit der Zeit wird die Temperatur reduziert, wodurch der Algorithmus sich einem globalen Optimum annähert. Eine ähnliche Vorgehensweise findet sich im ϵ -greedy Verfahren, bei dem ϵ , ein Parameter, der die Exploration im Verhältnis zur Exploitation steuert, allmählich verringert wird. Wie bei der kontrollierten Abkühlung, die Materialien in eine stabilere Struktur überführt, reduziert das Senken von ϵ im Laufe der Zeit die Neigung zur Erkundung und fördert eine zunehmend spezialisierte Nutzung der besten bekannten Handlungsoptionen [11].

Durch die methodische Reduktion von ϵ wird vermieden, dass der Lernprozess in suboptimalen Lösungen verharrt. Stattdessen bleibt die Möglichkeit erhalten, bessere Aktionen zu entdecken und zu nutzen, was zu einer effektiveren und effizienteren Suche führt [11].

Die Feinabstimmung von ϵ ist entscheidend, da eine nicht angepasste Rate entweder zu wenig Exploration oder zu verzögerter Optimierung führen kann [10].

Replay Buffers Replay Buffers verringern die Korrelation aufeinanderfolgender Erfahrungen und fördern so die Stabilität beim Lernen von RL-Agenten [10]. Sie speichern eine Sammlung von Erfahrungstupeln aus unterschiedlichen Policies und Trajektorien und ermöglichen eine effektivere Trainingsmethode durch zufällige Stichprobeneziehung, was die Netzwerkaktualisierungen diversifiziert [10].

Mathematisch wird der Replay Buffer wie folgt definiert [10]:

$$\mathcal{D} = \{e_t = (s_t, a_t, r_t, s_{t+1}, d_t) \mid t \in \{1, \dots, T\}\} \quad (2.15)$$

Sampling aus dem Replay Buffer erfolgt per Zufallsauswahl ohne Zurücklegen, um einen Minibatch von n Erfahrungen zu generieren:

$$\mathcal{B} = \{e_i \mid e_i \in \mathcal{D}, i \in \mathcal{I}\} \quad (2.16)$$

wobei $\mathcal{I} \subset \{1, \dots, T\}$, $|\mathcal{I}| = n$.

Die Kapazität des Buffers und das gleichförmige Sampling sorgen für ein diversifiziertes Lernen und vermeiden lokale Optima [10]. Die effektive Nutzung eines Replay Buffers erfordert eine ausreichende Kapazität, die typischerweise zwischen 10.000 und 1.000.000 Erfahrungen variiert [10]. Ein priorisiertes Sampling könnte eine intelligenteren Auswahlmethode darstellen, die eine ausgewogenere Lernumgebung fördert [10].

Fortgeschrittene Anpassungsfähigkeit und Lernmechanismen in Tiefen Q-Netzwerken (DQN) und ihre Bedeutung in der Entwicklung Künstlicher Intelligenz Die Tiefen Q-Netzwerke (DQN) haben sich als revolutionär in der Landschaft des maschinellen Lernens erwiesen. Mit ihrer Fähigkeit, eine Policy direkt aus rohen Pixeln zu lernen, signalisierte das DQN einen Wendepunkt in der Forschung von Reinforcement Learning (RL). Wie in [10] dargestellt, hat das DQN die Tür zu einer

Vielzahl von Forschungsinnovationen geöffnet und zu Beginn eine Leistung demonstriert, die auf menschenähnlichem Niveau auf einem Atari-Benchmark lag. Es ist erwähnenswert, dass trotz der Weiterentwicklungen DQN in seiner ursprünglichen Form nicht mehr als Primärwahl für aktuelle Anwendungen gilt, es bleibt jedoch ein Schlüsselement unter den bestperformenden DRL-Agenten. Die Fähigkeit des DQN, optimale Policy aus einer komplexen Umgebung wie der eines Atari-Spiels zu extrahieren, ist vergleichbar mit der Navigation in einem sich ständig verändernden Labyrinth, auf der Suche nach dem optimalen Weg ([11]). Die Anwendung der neuronalen Netze, speziell die tiefen Architekturen, ermöglicht es dem DQN, abstrakte Repräsentationen zu erlernen und diese für die Entscheidungsfindung zu nutzen. Eine Herausforderung, auf die in [12] hingewiesen wird, ist jedoch, dass DQN bei Spielen wie Montezuma's Revenge, die tiefergehende Planung erfordern, an seine Grenzen stößt. Obwohl DQN wesentliche Fortschritte im maschinellen Lernen gezeigt hat, verbleiben Herausforderungen in seiner Fähigkeit, komplexe Planungsaufgaben zu meistern. Das beeindruckende Potenzial der DQN-Methode wurde weiterhin durch das System AlphaGo von DeepMind illustriert, das tiefes RL verwendete, um menschliche Experten im Go-Spiel zu schlagen, einem Spiel, das für seine immense Anzahl möglicher Positionen bekannt ist und weit über das hinausgeht, was traditionelle Spiele bieten ([11]). Dies bestätigt, dass trotz der Herausforderungen, DQN und seine Weiterentwicklungen ein hohes Maß an Anpassungsfähigkeit und Lernfähigkeit aufweisen, das es ihnen ermöglicht, auch in komplexen Umgebungen zu bestehen. Im Licht dieser Quellen lässt sich zusammenfassen, dass DQN und dessen Erweiterungen ein bedeutsames Fundament für die Entwicklung von lernenden Agenten bieten, das sowohl die Forschung als auch praktische Anwendungen weiterhin beeinflusst.

Grenzen von Deep Q-Learning in kontinuierlichen Aktionsräumen Deep Q-Learning zeigt zwar in diskreten Aktionsräumen eine beeindruckende Leistung, stößt aber bei der Diskretisierung von kontinuierlichen Aktionsräumen auf bedeutende Herausforderungen. Die Notwendigkeit, kontinuierliche Aktionen in diskrete Schritte zu unterteilen, kann zu einem Verlust an Präzision und einer Verschlechterung der Leistung führen, insbesondere in Bereichen wie der Robotik, wo fein abgestimmte Kontrollaktionen erforderlich sind. Diese Limitation kann auch durch die Verwendung von Bewegungsprimitiven anstatt von niederstufigen Aktionen gemildert werden, was zwar das Lernen beschleunigt, aber die Bandbreite der Verhaltensweisen, die der Roboter lernen kann, einschränkt [11]. Weiterhin kann die Diskretisierung zu instabilen Steuerungsvorgängen führen, was insbesondere in sicherheitskritischen Systemen wie autonomen Fahrzeugen problematisch ist [13]. Zusätzlich wird die Anwendung von Deep Q-Learning durch die erhöhte Berechnungskomplexität und Schwierigkeiten bei der Konvergenz in großen Zustandsräumen eingeschränkt [11].

2.2.7 Deep Deterministic Policy Gradienten (DDPG)

Während das Deep Q-Network (DQN) für jeden Zustand s eine Bewertung aller möglichen Aktionen a liefert, wie in Abbildung 2.5 illustriert, stößt es in kontinuierlichen Aktionsräumen an seine Grenzen. Die Unzulänglichkeit von DQN in solchen Szenarien – da die Bewertung einer unendlichen Anzahl von Aktionen nicht praktikabel ist – führt uns zu fortgeschritteneren Methoden wie dem Deep Deterministic Policy Gradient (DDPG).[10]).

Einführung in die Actor-Critic-Architektur des DDPG Der DDPG-Algorithmus repräsentiert einen bedeutenden Fortschritt im Reinforcement Learning für kontinuierliche Aktionsräume. Zentral für seine Funktionsweise ist die Actor-Critic-Architektur, die aus zwei Schlüsselkomponenten besteht: dem Actor und dem Critic, die beide durch tiefe neuronale Netze approximiert werden [12].

Die Rolle des Actors Der Actor 2.7 hat die Aufgabe, eine optimale Policy $\mu(s)$ zu erlernen, die für jeden gegebenen Zustand s eine Aktion a ausgibt, mit dem Ziel, den erwarteten Reward zu maximieren [12]. Diese Policy-Funktion wird in einem kontinuierlichen Aktionsraum direkt modelliert, wobei jede Aktion ein kontinuierlicher Wert ist. Der Actor ist daher das Entscheidungselement innerhalb der Architektur [12].

Die Funktion des Critics Der Critic 2.7 hingegen bewertet die Aktionen, die vom Actor vorgeschlagen werden wie in Abbildung 2.8 gezeigt. Er nimmt den aktuellen Zustand und die vom Actor vorgeschlagene Aktion auf und schätzt den Q-Wert $Q(s, a)$. Diese Schätzung repräsentiert den erwarteten zukünftigen Reward, der aus der Ausführung der Aktion a im Zustand s resultiert, noch bevor die eigentliche Simulation in der Umgebung gestartet wird [12]. Somit versucht der Critic, die Qualität einer Aktion im

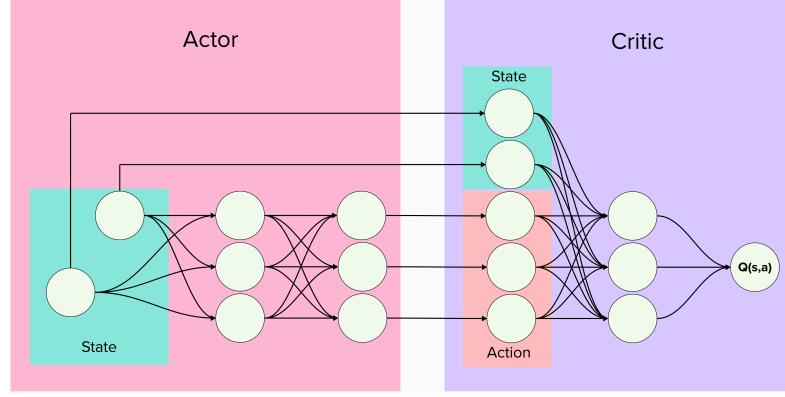


Figure 2.7: Illustration des Actors und Critics im DDPG-Algorithmus.

Kontext des aktuellen Zustands zu bewerten und liefert dem Actor Feedback, das zur Optimierung der Policy verwendet wird.

Die Actor-Critic-Interaktion In Abbildung 2.7 ist die Interaktion zwischen dem Actor und dem Critic illustriert. Der Actor generiert Aktionen basierend auf der aktuellen Policy, und der Critic bewertet diese Aktionen, indem er den Q-Wert zur Verfügung stellt. Diese Interaktion ermöglicht es dem DDPG-Algorithmus, sowohl die optimale Policy als auch die Wertfunktion effektiv zu lernen [12].

Aktualisierung des Critics im DDPG Die Aktualisierung des Critics im Deep Deterministic Policy Gradient Algorithmus ist ein entscheidender Schritt für das effektive Lernen von handlungsorientierten Strategien in kontinuierlichen Aktionsräumen. Die grundlegende Idee des DDPG ist es, die Stärken von Q-Learning 2.2.6 und Policy Gradient Methoden 2.13 zu kombinieren, um die Effizienz des Lernprozesses in solchen Umgebungen zu verbessern [8].

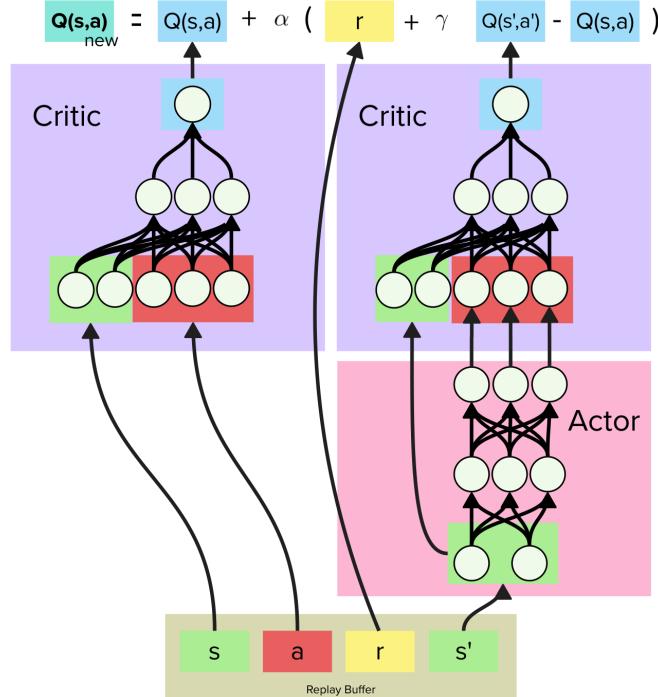


Figure 2.8: Berechnung des Q-Werts durch den Critic (Quelle: [wei2020ddpg]).

Update-Prinzip des Critics Der Critic bewertet die Politik des Actors, indem er die Q-Werte für die vom Actor vorgeschlagenen Aktionen schätzt. Diese Q-Werte repräsentieren das erwartete zukünftige Reward für den Zustand-Aktions-Paar (s, a) . Der Critic wird aktualisiert, indem der Temporal Difference (TD) Error minimiert wird 2.1.3, der die Differenz zwischen den geschätzten Q-Werten und den tatsächlichen Rewards aus der Umgebung widerspiegelt [13]. Die Update-Regel für den Critic ist eine Variante der Bellman-Gleichung 2.2.4 , die im DQN verwendet wird 2.2.6, und kann formal durch die folgende Gleichung ausgedrückt werden:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (r + \gamma \cdot Q(s', \mu(s')) - Q(s, a)) \quad (2.17)$$

Hierbei ist α die Lernrate, r der sofortige Reward, γ der Diskontierungsfaktor für zukünftige Rewards, und $\mu(s')$ die Policy des Actors, die den nächsten Zustand s' auf die nächste Aktion abbildet, wie in Abbildung 2.8 gezeigt.

Der TD Error für das Training des Critics kann dann als die Differenz zwischen dem geschätzten Q-Wert und dem Target-Q-Wert definiert werden:

$$TD_{error} = Q_{target}(s_t, a_t) - Q(s_t, a_t)$$

Diese Diskrepanz wird quadriert 2.6, um als Loss-Funktion zu fungieren, die dann minimiert wird , um das Lernen zu verbessern 2.7 [12].

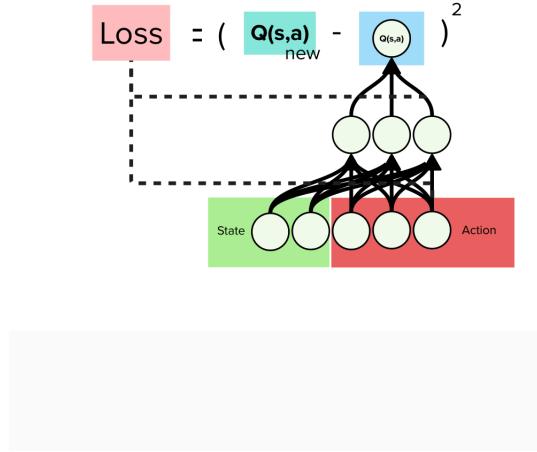


Figure 2.9: Aktualisierung des Q-Werts im Critic-Netzwerk.

Gradient Descent und Kettenregel Um die Gewichte des Critics zu aktualisieren, wird der Gradient Descent angewendet, wobei der Gradient der Loss-Funktion in Bezug auf die Gewichte berechnet 2.1.3 und die Gewichte entsprechend angepasst werden [1].

Bedeutung der Kritik Die Funktion des Critics ist es nicht nur, die Aktionen des Actors zu bewerten, sondern auch, eine Lernsignal für den Actor zu generieren. Dies geschieht durch die Rückkopplung des TD Errors an den Actor, wodurch dieser informiert wird, welche Aktionen zu einer Verbesserung oder Verschlechterung des erwarteten Rewards führen. Der Actor nutzt diese Information, um seine Policy in Richtung von Aktionen zu steuern, die zu einem höheren Reward führen [9].

Optimierung des Actor-Netzwerks im DDPG In Anlehnung an die zuvor dargestellte Aktualisierung des Critics ist das Ziel des Actor-Updates, eine Policy-Funktion zu erlernen, die Aktionen ausgibt, um den erwarteten Reward zu maximieren. Dieser Lernprozess basiert auf der Rückkopplung durch den Critic, der bereits präzise Q-Werte liefert. Die Aktualisierung erfolgt durch Anwendung eines Gradienten-Ascent-Verfahrens, welches auf die Maximierung der Q-Werte ausgerichtet ist [8, 13].

Die mathematische Darstellung des Updates wird durch die Gleichung ausgedrückt, die die Ableitung des Losses in Bezug auf die Parameter θ_μ des Actor-Netzwerks darstellt:

$$\nabla_{\theta_\mu} \text{LOSS}_{Actor} = -\mathbb{E}_{s \sim \mathcal{D}} [\nabla_a Q(s, a)|_{a=\mu(s)} \nabla_{\theta_\mu} \mu(s)] \quad (2.18)$$

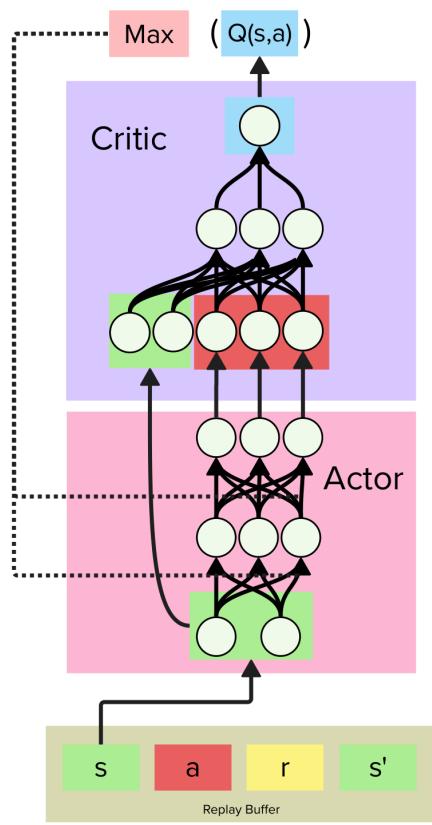


Figure 2.10: Gradienten-Ascent für die Aktualisierung des Actors, um die Q-Werte zu maximieren. Der Actor nutzt die vom Critic berechneten Q-Werte, um die Policy-Funktion zu optimieren und den erwarteten Reward zu maximieren.

Der Update-Mechanismus des Actors Wie in Abbildung 2.10 dargestellt, zielt das Update des Actors darauf ab, eine Policy-Funktion zu erlernen, die Aktionen so ausgibt, dass der erwartete Reward maximiert wird. Der Actor wählt Aktionen anhand der aktuellen Policy aus, die anschließend in der Umgebung ausgeführt werden, was zur Transition von Zustand s zu Zustand s' und zur Vergabe eines Rewards führt. Wie in [8] beschrieben, wird der Actor durch die Anwendung der Kettenregel 2.1.3 auf den erwarteten Return aus der Startverteilung aktualisiert, was eine direkte Anpassung der Actor-Parameter ermöglicht. Die Verlustfunktion des Actors wird durch die negativen Gradienten der Q-Werte in Bezug auf die Aktionen, berechnet an der Stelle der durch die Policy vorgeschlagenen Aktionen, definiert. Diese wird dann genutzt, um die Parameter der Actor-Policy in Richtung eines höheren erwarteten Rewards zu verschieben [8].

Die Bedeutung der Aktualisierung des Actors Die Aktualisierung des Actors ist ein kritischer Schritt im Lernprozess, da sie das Trainingssignal für den Actor aus den Bewertungen des Critics ableitet. Diese Interaktion ermöglicht es dem Actor, seine Policy zu verbessern und optimale Aktionen im Kontext der gegebenen Umgebung auszuwählen [9]. Die kontinuierliche Verbesserung der Policy durch den Actor führt zu einer verbesserten Entscheidungsfindung und letztlich zu einem höheren kumulativen Reward.

DDPG nutzt die Experience Replay-Technik, die zuerst in DQN eingeführt wurde 2.16, um die Korrelation zwischen aufeinanderfolgenden Lernschritten zu reduzieren. Dies erhöht die Datenvarianz und hilft, eine stabilere und effektivere Lernumgebung zu schaffen, wie in [13] beschrieben.

Exploration und Exploitation im DDPG Der DDPG-Algorithmus erweitert die traditionelle ϵ -greedy Strategie 2.14 um eine raffinierte Explorationstaktik, bei der ein kontrolliertes Rauschen auf die Auswahl kontinuierlicher Aktionen angewendet wird. Dieses adaptive Rauschen, wie Simulated-Annealing-Prozess 2.2.6, wird schrittweise verringert und ermöglicht so eine effektive Exploration des Aktionsraums. Durch diese graduelle Reduktion, verringert der Algorithmus die anfängliche Abhängigkeit von der Rauschintensität und fördert die Entdeckung optimaler Handlungsstrategien.

Integration von Experience Replay Neben dieser adaptiven Explorationsmethode implementiert DDPG auch die Erfahrungswiedergabe (siehe Gleichung 2.16), ein Ansatz, der zuerst in DQN zum Einsatz kam. Diese Technik ist entscheidend, um die Abhängigkeiten zwischen aufeinanderfolgenden Stichproben zu beseitigen und sie als unabhängig und identisch verteilt zu behandeln, was die Stabilität des Lernprozesses deutlich erhöht [13]. Die Anpassung des Actor-Netzwerks, das die zugrundeliegende Policy repräsentiert, folgt konsequent den Prinzipien der Kettenregel und der Rückpropagation, um eine konvergente Annäherung an die optimale Policy zu gewährleisten, die wiederum die Gesamteffizienz des Algorithmus steigert [13].

Die Rolle der Target Networks im DDPG Im DDPG-Algorithmus werden Target Networks eingesetzt, um die Stabilität während des Trainings zu erhöhen. Diese Netzwerke sind Kopien des Critic und des Actor, die mit einem geringeren Update-Faktor τ aktualisiert werden. Durch diese verzögerte Aktualisierung dienen die Target Networks als langsam veränderliche Zielwerte für die Updates des Critic und des Actor. Dies trägt dazu bei, die Korrelation zwischen den Schätzungen von $Q(s, a)$ und $Q(s', a')$ zu verringern und verhindert damit, dass der Lernalgorithmus durch schnelle Änderungen der bewerteten Policies instabil wird [13]. Während des Trainingsprozesses bleibt das Target Network fest, das heißt, seine Gewichte werden nicht bei jedem Schritt aktualisiert, sondern in regelmäßigen Abständen, um die Stabilität der Zielwerte zu gewährleisten. Dies bedeutet, dass das Target Network als Ankerpunkt für den Actor und den Critic fungiert und ihnen eine konsistente Richtung für die Aktualisierung gibt. Insbesondere beim Update des Critics wird der TD-Fehler gegen die stabilen Zielwerte des Target Networks berechnet, was die Varianz der Updates reduziert und die Konvergenz des Netzwerks fördert. Beim Aktualisieren des Actors hingegen wird die Policy basierend auf einer Rückkopplung optimiert, die von den stabilen Q-Werten des Critic Target Networks abgeleitet ist. Die Verwendung von Target Networks im DDPG ist analog zum Einsatz des Experience Replay: Beide Techniken zielen darauf ab, die Korrelationen zwischen aufeinanderfolgenden Lernschritten zu reduzieren und die Datenvarianz zu erhöhen. Sie helfen dabei, eine stabilere und effektivere Lernumgebung zu schaffen, wie in der Forschung von [9] und [13] beschrieben.

Überleitung zu praktischen Anwendungen Nachdem die Grundlagenkapitel die Schlüsseltechniken des maschinellen Lernens, wie die Vorwärts- und Rückwärtspropagation, die Berechnung von Gradienten

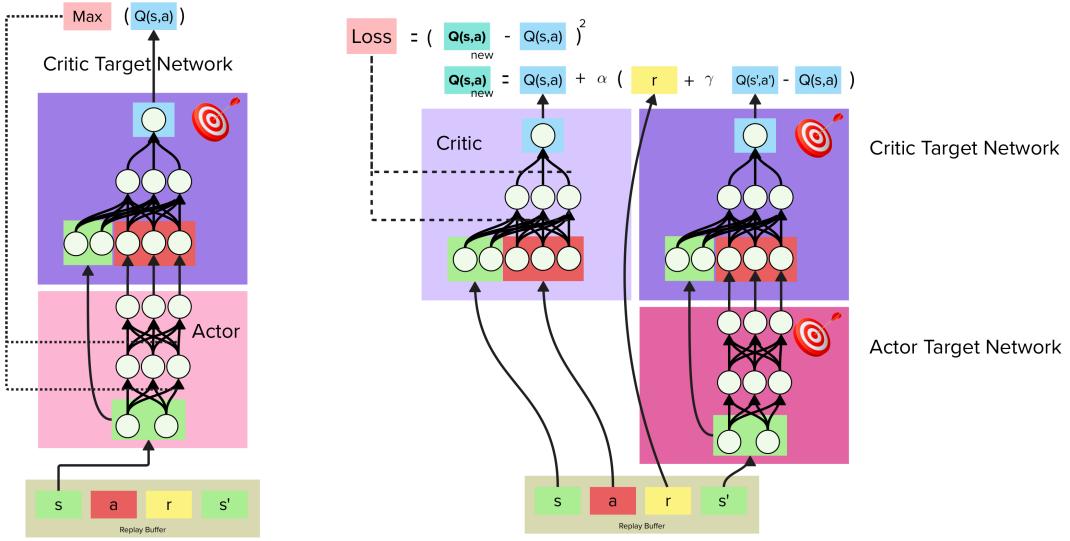


Figure 2.11: Interaktion zwischen Actor, Critic und den Target Networks im DDPG-Algorithmus.

sowie verschiedene Netzwerkarchitekturen, eingeführt haben, steht nun die Anwendung dieser Methoden auf reale Probleme im Fokus. Im nächsten Kapitel werden wir die Leistungsfähigkeit von RL-Techniken an einem praktischen Beispiel demonstrieren: der Optimierung von PID-Reglern für die Steuerung von DC-DC-Wandlern. Unser Ziel ist es, einen Actor zu trainieren, der in der Lage ist, die PID-Koeffizienten präzise anzupassen, um auf verschiedene Degradationsstufen des Wandlers zu reagieren und somit seine Effizienz signifikant zu steigern. Diese Anwendung repräsentiert eine Brücke zwischen theoretischen Konzepten und realen ingenieurtechnischen Herausforderungen und zeigt, wie fortgeschrittene Algorithmen des maschinellen Lernens zur Lösung komplexer Steuerungsprobleme beitragen können.

2.2.8 Reward-Funktion für Trading-Bots

Kompakte Formel:

$$\text{Reward}_t = \frac{\text{Gain}_t - \text{BestScenarioGain}_t}{V_t}$$

wobei:

$$\text{Gain}_t = V_t - V_{t-1}$$

$$\text{BestScenarioGain}_t = \max(0, \text{PotentialGainIfFullyInvested}_t - V_{t-1})$$

$$\text{PotentialSharesIfFullyInvested}_t = \frac{C_{t-1} \times \text{TransactionPenalty}}{P_{t-1}} + S_{t-1}$$

$$\text{PotentialGainIfFullyInvested}_t = \text{PotentialSharesIfFullyInvested}_t \times P_t \times \text{TransactionPenalty}$$

Erläuterung: 1. **Realer Gewinn** (Gain_t): Die Differenz zwischen dem aktuellen Portfoliowert (V_t) und dem vorherigen Portfoliowert (V_{t-1}).

2. **Bestes Szenario** ($\text{BestScenarioGain}_t$): Der Gewinn im besten hypothetischen Szenario, falls der Preis steigt, abzüglich des vorherigen Portfoliowerts (V_{t-1}). Falls der Preis fällt, ist der Gewinn im besten Szenario 0.

- **Hypothetischer Aktienbestand** ($\text{PotentialSharesIfFullyInvested}_t$): Die Anzahl der Aktien, die man hätte kaufen können, wenn man im vorherigen Zeitschritt alle verfügbaren Barmittel in Aktien

investiert hätte, plus die bereits vorhandenen Aktien (S_{t-1}).

$$\text{PotentialSharesIfFullyInvested}_t = \frac{C_{t-1} \times \text{TransactionPenalty}}{P_{t-1}} + S_{t-1}$$

- **Hypothetischer Gewinn** ($\text{PotentialGainIfFullyInvested}_t$): Der hypothetische Gewinn, der sich aus dem hypothetischen Aktienbestand, dem aktuellen Aktienpreis und den Transaktionskosten ergibt.

$$\text{PotentialGainIfFullyInvested}_t = \text{PotentialSharesIfFullyInvested}_t \times P_t \times \text{TransactionPenalty}$$

3. **Normalisierung** (Reward_t): Die Differenz zwischen dem realen Gewinn und dem Gewinn im besten Szenario, geteilt durch den aktuellen Portfoliowert. Dadurch wird die Belohnung in Relation zum aktuellen Portfoliowert gesetzt.

Interpretation: - **Positive Belohnung:** Der Agent hat besser abgeschnitten als im besten hypothetischen Szenario (oder zumindest gleich gut, wenn der Preis gefallen ist). - **Negative Belohnung:** Der Agent hat schlechter abgeschnitten als im besten hypothetischen Szenario (d.h., er hätte mehr Gewinn machen können, wenn er anders gehandelt hätte). - **Null Belohnung:** Der Agent hat genauso gut abgeschnitten wie im besten hypothetischen Szenario.

Vorteile der kompakten Formel: - **Übersichtlicher:** Die Formel ist kürzer und leichter zu erfassen. - **Intuitiver:** Die einzelnen Komponenten sind klarer benannt und ihre Bedeutung ist leichter verständlich.

Chapter 3

Realisierung

3.1 Motivation und Zielsetzung

Das zentrale Anliegen dieser Arbeit ist die Entwicklung eines effizienten Agenten, basierend auf den Prinzipien des Reinforcement Learning, der in der Lage ist, auf Degenerationsprozesse in elektronischen Schaltungen zu reagieren. Dieser Agent soll durch ein neuronales Netzwerk realisiert werden und in der Lage sein, optimale PID-Reglereinstellungen für sich verändernde Degenerationszustände innerhalb einer Schaltung zu bestimmen. Ein DC-DC-Konverter dient als Ausgangsmodell, um den Agenten zu trainieren und die grundlegenden Prinzipien zu veranschaulichen, mit dem Ziel, eine generalisierbare Lösung zu entwickeln, die auf unterschiedliche Schaltungskontexte anwendbar ist.

3.2 Herausforderungen und Ziele

Die Entwicklung eines leistungsfähigen Reinforcement-Learning-Systems stellt mehrere Herausforderungen dar, die in diesem Experiment adressiert werden müssen:

1. **Agentenwahl und -konfiguration:** Die Auswahl und Konfiguration des Agenten, insbesondere die Architektur des neuronalen Netzes, sind entscheidend, um effektives Lernen zu gewährleisten. Der Agent muss in der Lage sein, die komplexen Zusammenhänge zwischen den Zuständen der Schaltung und den entsprechenden Aktionen zu erfassen.
2. **Aufbau einer effizienten Simulation:** Eine Simulation, die schnell genug ist, um zeitnahe Trainingsdurchläufe zu ermöglichen, ist für die Konvergenz des Modells von entscheidender Bedeutung. Die Simulationsumgebung muss akkurate Rückmeldungen in einer Zeitspanne liefern, die praktikables und iteratives Training zulässt.
3. **Bestimmung der Reward-Funktion:** Die Definition einer angemessenen Reward-Funktion ist essentiell, um das Agentenverhalten in Richtung der gewünschten Zielvorgaben zu lenken. Die Reward-Funktion muss so gestaltet sein, dass sie das System zuverlässig zum gewünschten Leistungsziel führt.
4. **Validierung des trainierten Netzwerks:** Neuronale Netzwerke können oft auch unter suboptimalen Einstellungen zufriedenstellende Ergebnisse liefern. Die eigentliche Herausforderung besteht darin, das Modell so zu trainieren, dass es die Gegebenheiten und Dynamiken der Simulationsumgebung nicht nur nachahmt, sondern auch versteht und die Leistungsfähigkeit der Schaltung präzise verbessert.

Die Überwindung dieser Herausforderungen erfordert einen methodischen Ansatz, der die Interaktion zwischen den Komponenten des Systems detailliert analysiert und optimiert.

3.3 Wahl der Architektur und verwendete Technologien

Für das Experiment wurden spezifische Technologien und Architekturen ausgewählt, die optimal auf die Anforderungen der Problemstellung abgestimmt sind. Die Entscheidung fiel auf eine Kombination

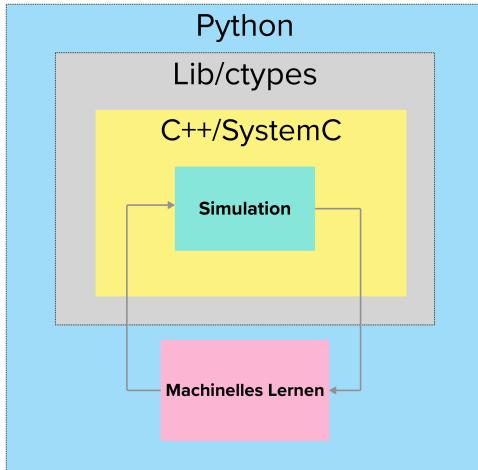


Figure 3.1: Die Architektur des integrierten Systems, das die Schaltungssimulation und das maschinelle Lernen verbindet.

aus Python für die Implementierung des Reinforcement Learning-Teils und SystemC mit C++ für die Simulation der elektronischen Schaltung.

3.3.1 Reinforcement Learning mit Python

Der Reinforcement Learning-Teil des Experiments wurde in Python entwickelt, unter Verwendung von modernen Frameworks wie TensorFlow 2 und Keras. Diese Bibliotheken erleichtern den Entwurf und das Training von neuronalen Netzwerken durch ihre hohe Abstraktion und Benutzerfreundlichkeit. Zusätzlich wurde CUDA verwendet, um das Training der Modelle auf Grafikkarten zu beschleunigen und somit die Konvergenzzeit signifikant zu reduzieren.

3.3.2 SystemC und C++ für die Schaltungssimulation

Der Teil des Experiments, der sich auf die elektronische Schaltung bezieht, wurde in SystemC implementiert, einem C++ basierten Modellierungsframework, das sich durch seine Fähigkeit zur schnellen und präzisen Simulation von Hardwarekomponenten auszeichnet. SystemC bietet eine hohe Modularität und Eignung für die Entwicklung komplexer Schaltungssysteme und ermöglicht eine detaillierte Analyse der Schaltungsverhalten unter verschiedenen Bedingungen.

Integration von Python und SystemC Die Integration der in Python entwickelten Reinforcement Learning-Komponente mit der in SystemC implementierten Schaltungssimulation stellt eine besondere technische Herausforderung dar. Diese wurde durch den Einsatz spezieller Schnittstellen und Protokolle gemeistert, welche die beiden unterschiedlichen Umgebungen miteinander kompatibel machen und einen reibungslosen Datenaustausch gewährleisten.

Lib/ctypes als Brücke Lib/ctypes ist eine Bibliothek in Python, die es ermöglicht, C-Funktionen innerhalb der Python-Umgebung aufzurufen. Dies erlaubt eine effiziente Integration der C++-basierten Schaltungssimulation in das Python-basierte Lernmodell, ohne dass die Leistung darunter leidet.

Modulare und erweiterbare Softwareentwicklung Die gesamte Entwicklung folgte den Prinzipien der Kapselung, objektorientierten Programmierung und Interface-gesteuerten Design, was die spätere Anpassung und Erweiterung auf andere Schaltungsmodelle ermöglicht. Dieser modulare Ansatz trägt dazu bei, dass die entwickelten Lösungen auch in anderen Kontexten und für verschiedene Schaltungstypen anwendbar sind.

Für das Kernstück des Experiments, die Wahl der geeigneten Reinforcement Learning-Architektur, fiel die Entscheidung auf den Deep Deterministic Policy Gradient (DDPG). Dieser Ansatz bietet mehrere Vorteile, die ihn für das vorliegende Experiment besonders qualifizieren. Bei der Wahl der Architektur für dieses Experiment wurden drei Hauptgründe berücksichtigt:

- Behandlung kontinuierlicher Aktionsräume: Die gewählte Architektur kann effektiv mit kontinuierlichen Aktionsräumen umgehen, was für analoge Anwendungen und reale Einsatzgebiete entscheidend ist.
- Schnelle Konvergenz: Der Deep Deterministic Policy Gradient (DDPG) zeichnet sich durch seine schnelle Konvergenz aus. Da die Simulationen im Experiment zeitaufwendig sind, ist eine zügige Konvergenz wichtig für die Effizienz des Trainingsprozesses.
- Modularität der Architektur: Die aus Äktor (Actor) und Kritiker (Critic) bestehende Architektur ermöglicht eine Variation von Gewichten und Schichten in getrennten Netzwerkteilen, was besonders bei der Übertragung des Modells auf physische Implementierungen wie einen Mikrochip von Vorteil ist.

Die Architektur wurde also aufgrund ihrer Eignung für kontinuierliche Aktionsräume, ihrer schnellen Konvergenz und ihrer Modularität ausgewählt, um den Anforderungen des Experiments gerecht zu werden.

3.4 Überblick über den Aufbau und Datenfluss

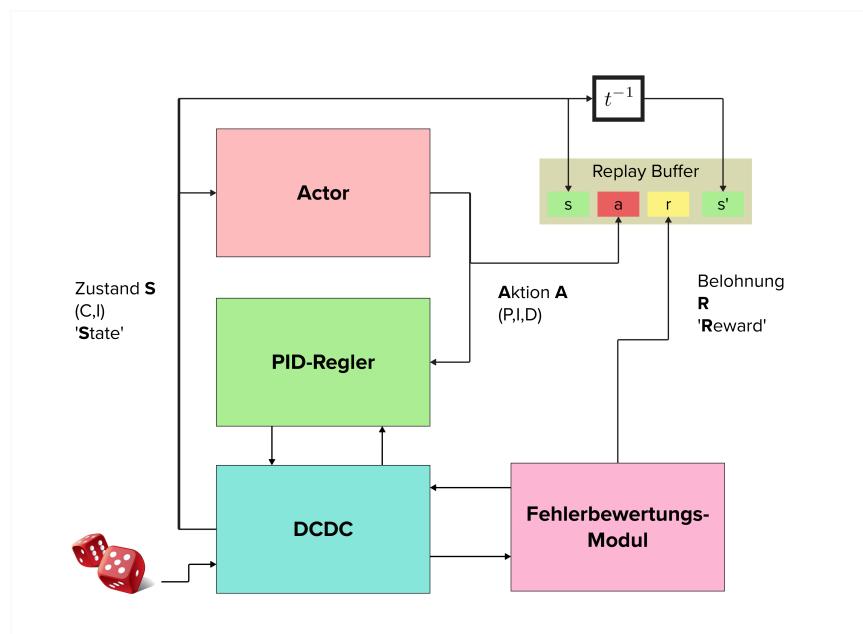


Figure 3.2: Detaillierte Darstellung des Systems zur Datensammlung für das Training des neuronalen Netzes.

Einleitung Dieser Abschnitt gewährt einen Überblick über die Struktur und den Datenfluss innerhalb Moduln 3.2. Die detaillierten theoretischen Grundlagen der Systemkomponenten wurden im Grundlagen-Teil dieser Arbeit ausführlich behandelt. An dieser Stelle werden die Funktionen und Interaktionen der einzelnen Module im Kontext der Datenerfassung für das Training des neuronalen Netzes dargestellt.

Datenfluss im Reinforcement Learning System Im Zentrum des Datenflusses unseres Systems steht die Befüllung des Replay Buffers 3.42.2.6 mit wertvollen Informationen für das Training des Agenten. Diese Informationen bestehen aus den Zuständen der Schaltung, den getroffenen Aktionen des Agenten und den daraus resultierenden Belohnungen.

3.4.1 Makrozyklus: Systemüberblick und Datenauswertung

Im Makrozyklus unseres Systems, wie in der beigefügten Abbildung 3.3 dargestellt, konzentrieren wir uns auf die Simulation der Systemzustände und die anschließende Datenauswertung für das Training des neuronalen Netzwerks. Dieser Zyklus beinhaltet die Sammlung von Daten über verschiedene Zustände der

Schaltung, einschließlich der Simulation der Systembedingungen sowie der Auswertung und Trainierung des neuronalen Netzwerks. Jeder Schritt in diesem Zyklus trägt dazu bei, ein umfassendes Verständnis der Systemdynamik zu entwickeln, was für die effiziente Anpassung des Netzwerks entscheidend ist. Der Makrozyklus wurde hauptsächlich in Python implementiert, um die Flexibilität und Erweiterbarkeit der Datenverarbeitungs- und Analyseprozesse zu maximieren.

3.4.2 Mikrozyklus: Detailanalyse und Regelungsstrategien

Im Mikrozyklus liegt der Fokus auf der Simulation der Verhaltensweisen des DC-DC-Konverters unter verschiedenen Bedingungen, insbesondere im Zusammenspiel mit einem PID-geregelten System. Diese Detailanalyse ermöglicht es, fein abgestimmte Regelungsstrategien zu entwickeln, die eine präzise Steuerung der Systemkomponenten unter variierenden Betriebsbedingungen erlauben. Der Mikrozyklus wurde in SystemC implementiert, was sich als besonders effizient für die Transientenanalyse einzelner Komponenten und die Simulation unter unterschiedlichen Bedingungen erwiesen hat. Die Durchführung eines Mikrozyklus auf einer Standard-CPU dauert etwa 0,3 Sekunden, was die schnelle Reaktionsfähigkeit und Anpassungsfähigkeit unseres Systems unterstreicht.

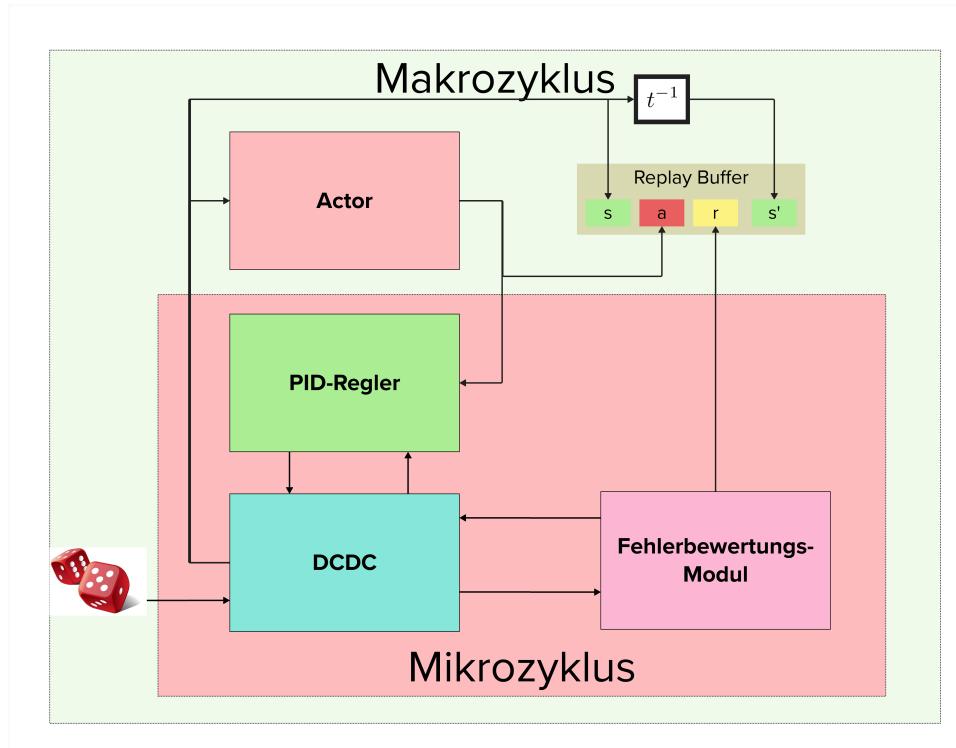


Figure 3.3: Visualisierung des Mikro- und Makrozyklus mit Darstellung der zeitlichen Unterteilung in zwei Dimensionen.

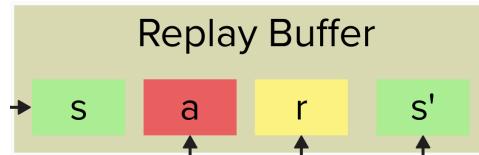


Figure 3.4: Schematische Darstellung des Replay Buffers, der als Datenspeicher für das Reinforcement Learning System dient. Er zeichnet die Sequenz der erlebten Zustände s , Aktionen a , Belohnungen r und nachfolgenden Zustände s' auf, welche für die stetige Anpassung und Verbesserung des Agenten verwendet werden.

3.4.3 Makrozyklus Schritt 1: Zustandssimulation und Datengenerierung

Die Generierung der Schaltungszustände, die als Input für den Lernprozess dienen, erfolgt mittels einer Zufallsfunktion. In der realen Welt erfolgen Zustandsübergänge in einer physischen Schaltung in langen Zeitintervallen, was eine direkte Simulation unpraktisch macht. Deshalb greifen wir auf eine alternative Methode zurück:

- Initial wird über eine Gleichverteilung ein Degenerationszustand der Schaltung simuliert. Die zufällige Auswahl dieser Zustände geschieht innerhalb festgelegter Grenzen, um eine Vielfalt an möglichen Zuständen zu gewährleisten.
- Der simulierte Zustand wird durch einen Satz von Kapazitäts- und Induktivitätswerten dargestellt. Diese Werte werden durch den "Würfel" im System visualisiert 3.5, wobei die Pfeile vom Würfel zu den Zustandsvariablen C und L diesen Prozess der zufälligen Zustandsgenerierung abbilden.

In diesem Projekt werden exemplarische Einstellungen für die Zustandsgenerierung verwendet, die eine Annäherung an realitätsnahe Bedingungen darstellen. Bei einer Überführung in praktische Anwendungen würden diese Einstellungen so angepasst, dass sie mit den in der Realität verifizierten Werten übereinstimmen. Die aktuellen Grenzwerte für die Zustandsgenerierung sind wie folgt definiert:

- Untergrenze: Induktivität = 5.0×10^{-4} , Kapazität = 1.0×10^{-6}
- Obergrenze: Induktivität = 5.0×10^{-2} , Kapazität = 1.0×10^{-4}

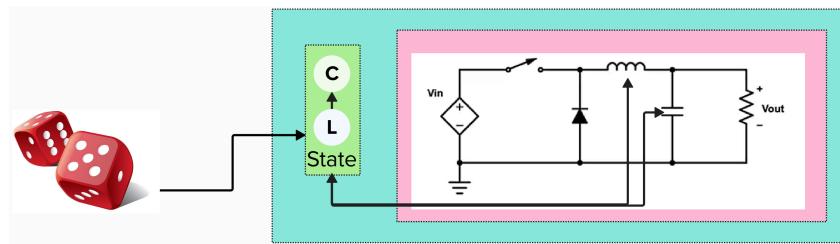


Figure 3.5: Visualisierung der Zustandsgenerierung mit dem Würfel und die Setzung der Werte in der DCDC-Schaltung.

3.4.4 Makrozyklus Schritt 2: Entscheidungsfindung durch den Actor

Nachdem der initiale Schaltungszustand durch eine Zufallsverteilung generiert wurde, kommt der Actor ins Spiel, der auf Basis der aktuellen Kapazitäts- (C) und Induktivitätswerte (L) Entscheidungen trifft. Der Actor, ein wesentlicher Bestandteil des Reinforcement Learning Systems, ist eine komplexe Funktion, die sich aus mehreren Gewichtungen (weights) und Verzerrungen (biases) zusammensetzt. Diese werden im Laufe des Vorwärtsdurchlaufs (Forward Propagation 2.1.2) durch das Netzwerk multipliziert und summiert.

- Innerhalb des Actors wird die Eingabe durch jede Schicht des neuronalen Netzwerks transformiert, wobei nach jedem Schichtdurchgang eine nicht-lineare Aktivierungsfunktion 2.2 angewandt wird, um die Linearität der Operationen zu durchbrechen.
- Im spezifischen Kontext unseres Systems ist die Aktivierungsfunktion der letzten Schicht eine Hyperbelfunktion (\tanh), die die Ausgabe des Netzwerks beeinflusst und zu einem komplexen, nicht-linearen Ergebnis führt.

Die Ausgabe des Actors sind die PID-Werte, die für die Steuerung des nächsten Schrittes im System verwendet werden. Diese Werte sind das Resultat des durch die \tanh -Funktion modifizierten Outputs und stellen somit eine fein abgestimmte Reaktion auf den Zustand der Schaltung dar. Diese Werte werden anschließend skaliert, um realistische PID-Reglerparameter zu erhalten:

- Der Proportionalwert (Kp) wird zwischen 0 und 10 skaliert.
- Der Integralwert (Ki) wird zwischen 0 und 1 skaliert.

- Der Differentialwert (K_d) wird zwischen 0 und 0.1 skaliert.

Diese Skalierung ist entscheidend, um die Ausgangswerte des neuronalen Netzes in praktikable Steuerparameter zu überführen. Die Transformation gewährleistet, dass die PID-Werte in einem Bereich liegen, der für die Regelung des Systems adäquat ist und reflektiert die praktischen Anforderungen an die Systemsteuerung.

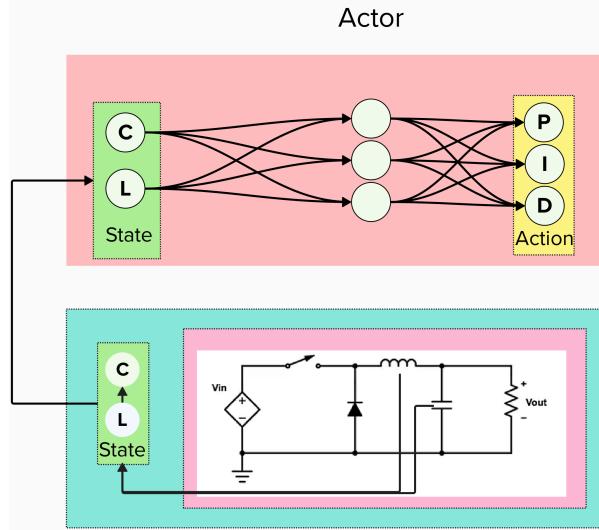


Figure 3.6: Die Forward Propagation im Actor-Modul, die den Zustand S (C, L) aufnimmt und über ein mehrschichtiges neuronales Netzwerk PID-Aktionswerte ausgibt.

3.4.5 Gesamtdarstellung des Regelkreises mit PID-gesteuertem DCDC-Konverter

Dieser Abschnitt bietet einen umfassenden Überblick über den Regelkreis, der einen PID-Regler, einen Pulsweitenmodulator (PWM) und einen DCDC-Konverter umfasst. Im Mittelpunkt steht die Interaktion dieser Komponenten, die gemeinsam eine konstante Ausgangsspannung gewährleisten. Die Mikroebene der Simulation wird besonders hervorgehoben, um die Auswirkungen der Wechselwirkungen zwischen den einzelnen Elementen auf die Gesamtleistung des Systems zu verdeutlichen. Abbildung 3.7 veranschaulicht den gesamten Regelkreis inklusive des PID-gesteuerten DCDC-Konverters.

Interaktion der Regelkreiskomponenten Der Regelungsprozess beginnt mit den vom Actor gelieferten PID-Koeffizienten K_p , K_i , und K_d , die den PID-Regler ansteuern (siehe Abschnitt ??). Der Regler vergleicht kontinuierlich die aktuelle Spannung mit dem Referenzwert und generiert ein Signal zur Korrektur, das an den PWM weitergeleitet wird.

Funktionsweise des Pulsweitenmodulators (PWM) Der PWM-Modulator (Abschnitt ??) transformiert das Signal des PID-Reglers in eine pulsbreitenmodulierte Form, die den DCDC-Konverter ansteuert. Dieser Konverter sorgt für die Umwandlung der Eingangsspannung in eine stabile Ausgangsspannung. Die Signalverarbeitung im PWM bestimmt die Schaltfrequenz des Transistors im Konverter, um die Last angemessen zu versorgen.

Rolle des DCDC-Konverters Der DCDC-Konverter spielt eine zentrale Rolle im Regelkreis, indem er die Energie der Versorgungsspannung in eine nutzbare Ausgangsspannung umwandelt (siehe Abschnitt ??). Die Arbeitsweise des Konverters ist eng mit den PWM-Signalen verknüpft, welche das Öffnen und Schließen des Transistors steuern. Die PWM regelt die Impulsbreite basierend auf den Steuerbefehlen des PID-Reglers. Diese Impulse beeinflussen das Tastverhältnis und damit die Energiemenge, die durch den Transistor geleitet wird. Die Energie wird in der Induktivität gespeichert und als Strom an die Last abgegeben, wobei die Kapazität als Puffer dient. Änderungen in der Last führen zu Spannungsschwankungen, die der PID-Regler erfasst und ausgleicht, um eine stabile Ausgangsspannung zu gewährleisten.

Integration von Simulation und Realität Die Simulation des Regelkreises erfolgte in SystemC, wobei der PID-Regler direkt aus mathematischen Modellen abgeleitet und der PWM sowie der DCDC-Konverter für eine genauere Analyse implementiert wurden. Dies ermöglicht einen Ausgleich zwischen Genauigkeit und Simulationsgeschwindigkeit.

3.4.6 Selbsttestmodul im Lernprozess des Agenten

Das Selbsttestmodul ist ein essentielles Element unserer Systemarchitektur, welches die Schnittstelle zwischen Theorie und Praxis repräsentiert. Es bewertet und optimiert die Effektivität der vom Agenten erlernten Schaltstrategien.

Funktion und Notwendigkeit des Selbsttests Der Selbsttest bewertet die Schaltungsreaktionen auf definierte Testszenarien, um die Lernergebnisse des Agenten zu überprüfen, wie in Abbildung 3.9 dargestellt.

Methodik der Performanzbewertung Die Bewertungsmethodik integriert die Abweichungen zwischen Soll- und Ist-Spannung, wobei größere Fehler durch Quadrierung stärker gewichtet werden. Extreme Spannungsspitzen werden, wie in Abbildung 3.10 gezeigt, exponentiell bestraft.

Auswirkungen auf das Trainingsziel Diese Bewertungsmethodik führt zu einem quantifizierbaren Wert, der die Qualität der Regelung anzeigt und den Agenten anleitet, den Fehlerwert zu minimieren. Dies wird durch den negativen Reward widergespiegelt, der in den Lernprozess des Agenten einfließt.

Zeitliche Dimension der Analyse Selbsttests werden in wiederkehrenden Zyklen durchgeführt und bewertet, wobei der kumulierte Fehlerwert im Replay Buffer gespeichert wird, um die Strategie zu optimieren, wie in Abbildung 3.3 illustriert.

3.5 Zusammenführung des Actor-Critic-Verfahrens

Im vorherigen Kapitel haben wir eine Trainingsumgebung für verstärkendes Lernen etabliert, die das Sammeln von Zuständen, Aktionen und Belohnungen (Rewards) umfasst. Diese Daten werden in einem Replay Buffer gespeichert, aus dem sie entsprechend der Batch-Größe für das Training entnommen werden. (siehe Abschnitt 2.1.8)

Die Dynamik des Replay Buffers Aus dem Replay Buffer (siehe Abschnitt 2.2.6) werden Datensätze extrahiert, die simultan zur Berechnung des Gradienten und zur Aktualisierung der Actor-Critic Architektur herangezogen werden. Der Replay Buffer ist das Herzstück unserer Trainingsdynamik. Er speichert die Erfahrungen, die der Algorithmus im Laufe der Simulation macht, und ermöglicht es uns, die komplexen Zusammenhänge zwischen Aktionen und resultierenden Belohnungen zu extrahieren.

Die Aufgabe des Critics Der Critic bewertet die vom Actor ausgeführte Policy und den gegebenen Zustand, evaluiert die Effektivität der vom Actor vorgeschlagenen Aktionen im Kontext der Schaltung (siehe Abschnitt 2.17). Die generierten Schätzwerte des Critics werden mit den tatsächlichen Belohnungen verglichen, womit der Critic das Verhalten der Schaltung in Abhängigkeit von der jeweiligen Konfiguration prognostiziert. Der Critic Durch das Abgleichen seiner Vorhersagen mit den tatsächlichen Belohnungen lernt der Critic, das Verhalten der Schaltung präzise vorherzusagen und unterstützt damit die zielgerichtete Anpassung der PID-Koeffizienten.

Optimierungsmechanismus des Actors Die Feinjustierung des Actors ist ein zentraler Bestandteil des Lernprozesses innerhalb der Actor-Critic Architektur. Anstatt komplexe Bewertungen vorzunehmen, wird ein Gradient berechnet, der die Gewichte des Actors so anpasst, dass die Schätzung des Critics in die richtige Richtung gelenkt wird, um den Reward zu maximieren (siehe Abschnitt 2.18). Dieser Ansatz ermöglicht es dem Actor, aus den Prognosen des Critics zu lernen und seine Policy systematisch so zu verfeinern, dass die Belohnungsausschüttung erhöht wird. Es handelt sich hierbei um den Kern des Lernvorgangs, bei dem der Actor durch die Adjustierung seiner Gewichte basierend auf der Kritik und den Belohnungen iterativ verbessert wird.

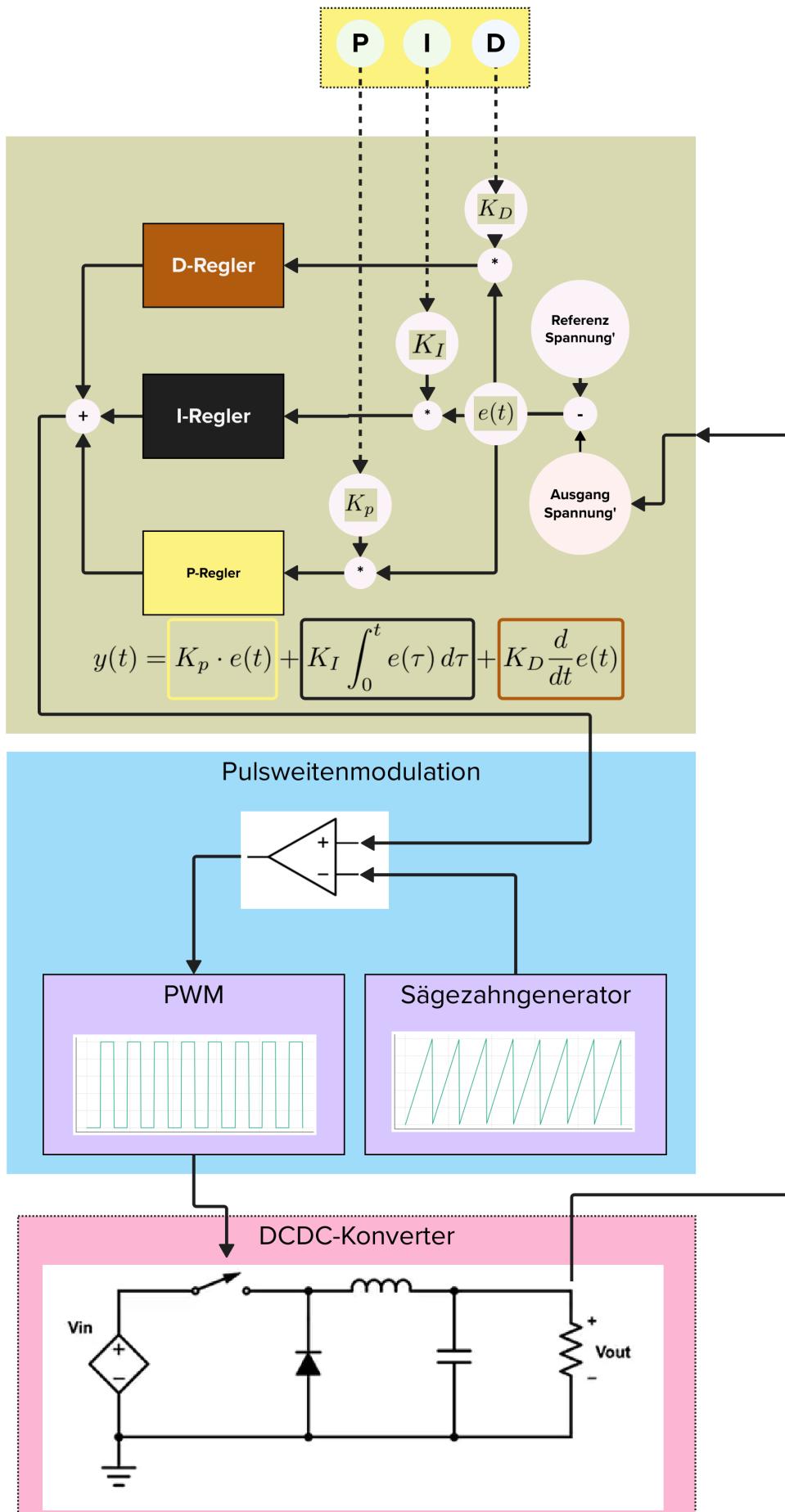


Figure 3.7: Überblick über den Regelkreis mit dem PID-Regler, PWM und DCDC-Konverter.

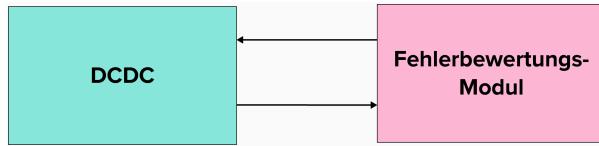


Figure 3.8: Darstellung des Selbsttestmoduls im Regelkreis mit PID-Regler, PWM und DCDC-Konverter.

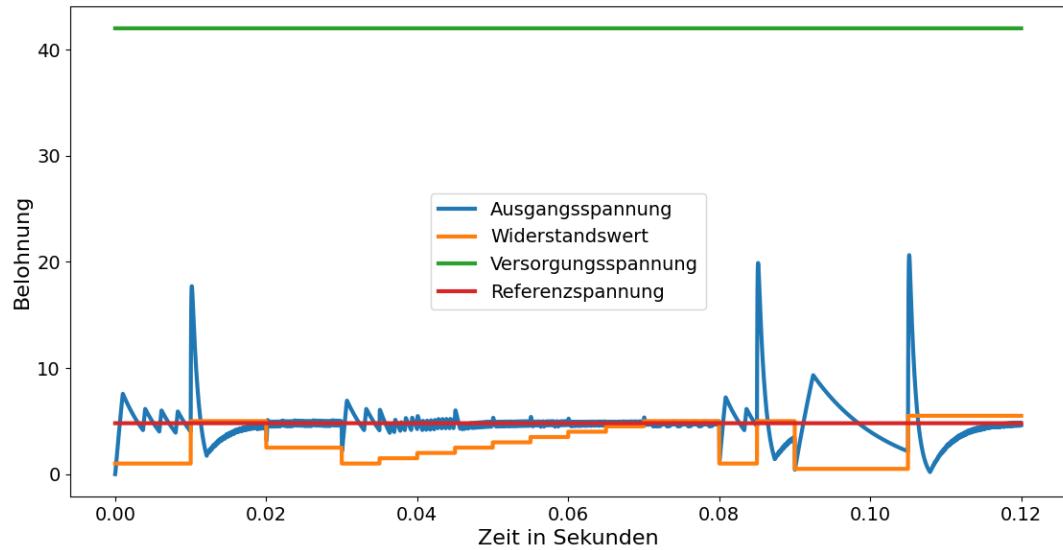


Figure 3.9: Visualisierung des Rewards über die Zeit während der Simulation.

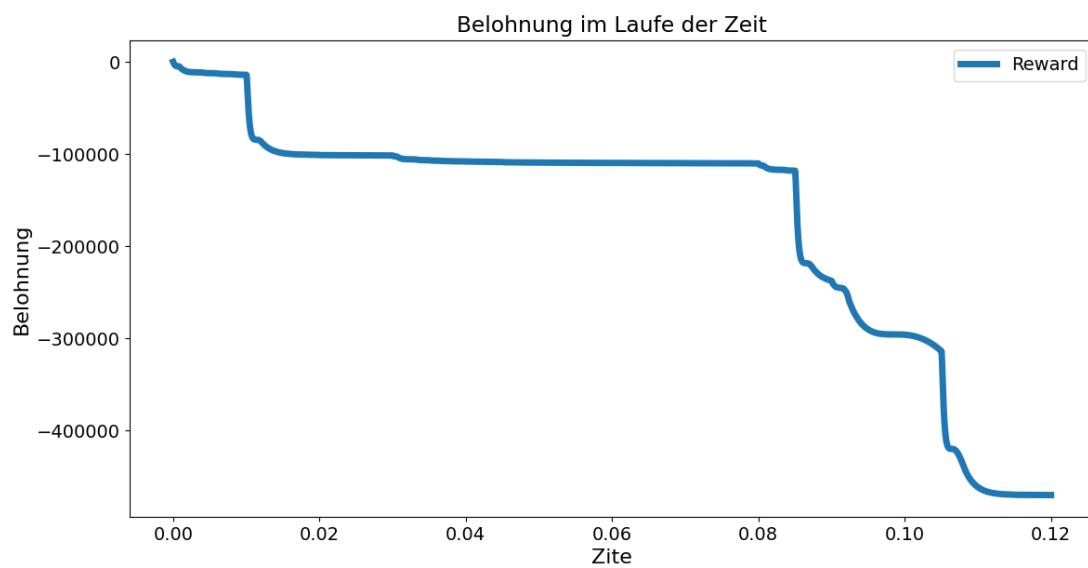


Figure 3.10: Kumulativer Reward über die Zeit, aufgezeichnet während der Systemtests.

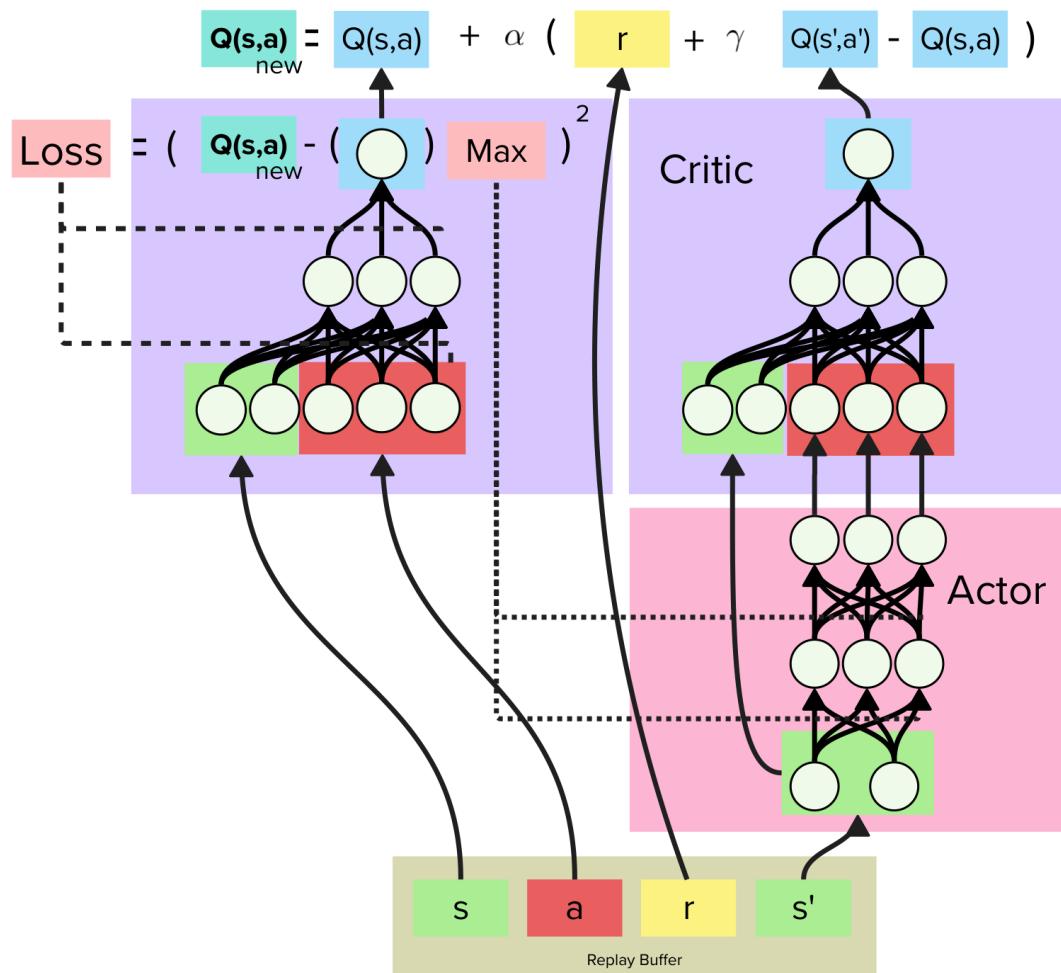


Figure 3.11: Zusammengefasste Visualisierung des Update-Prozesses in der Actor-Critic Architektur.

Kontinuierliche Verbesserung durch Training Die stetige Aktualisierung des Gradienten nach jedem Simulationsschritt sorgt für eine kontinuierliche Anpassung und Verfeinerung der Actor-Critic Architektur (siehe Abschnitt 3.5), was die Belohnungen im Laufe der Zeit maximiert. Angenommen, das Ziel ist eine direkte Optimierung der Belohnungen in unserer Simulation – die Herausforderung besteht darin, dass die komplexe Pipeline von Aktionen bis hin zum endgültigen Reward nicht mit einfachen mathematischen Mitteln abgebildet werden kann. Daher nutzen wir den Actor-Critic Ansatz, um einen einfacheren Gradienten in Bezug auf den erwarteten Reward zu maximieren, was indirekt die Optimierung der Aktionen ermöglicht.

Chapter 4

Ergebnisse

Nach der erfolgreichen Realisierung einer robusten Actor-Critic-Architektur haben wir eine entscheidende Phase erreicht: die Präsentation unserer Forschungsergebnisse. Anfänglich zeigte unser Modell vielversprechende Resultate mit scheinbarer Leichtigkeit. Allerdings stellten wir fest, dass eine weitere Verfeinerung notwendig war, um exzellente Leistungswerte zu erzielen. Diese kontinuierliche Verbesserung war besonders im Feintuning-Prozess bemerkbar, wenngleich die Qualität der Ergebnisse anfangs unsicher war.

Um die Validität unserer Ergebnisse zu überprüfen, griffen wir auf Bayesian Optimization zurück, eine Technik, die für ihre herausragende Fähigkeit zur Auffindung optimaler Parameterwerte bekannt ist. Das Modell wurde speziell so konzipiert, dass es mit dieser Technik kompatibel ist.

Mit Bayesian Optimization konnten wir gezielt spezifischen Degradationszustand auswählen und diesen als Benchmark heranziehen. Diese gezielte Auswahl ermöglichte es uns, die Effektivität und Genauigkeit der Architektur zu bestätigen.

Abbildung 4.1 zeigt eine dreidimensionale Visualisierung verschiedener PID-Konfigurationen, die durch Bayesian Optimization ermittelt wurden. Die verschiedenen Farbnuancen korrespondieren mit den erzielten Rewards, wobei dunklere Farbtöne höhere Belohnungen symbolisieren. Die systematische und gleichmäßige Abtastung des Parameterraums durch Bayesian Optimization ist deutlich zu erkennen. Die Intensivierung der Suche in Bereichen, die vielversprechende Werte zeigten, verdeutlicht die zielgerichtete und effiziente Natur des Optimierungsprozesses.

Bayesian Optimization of PID Controller

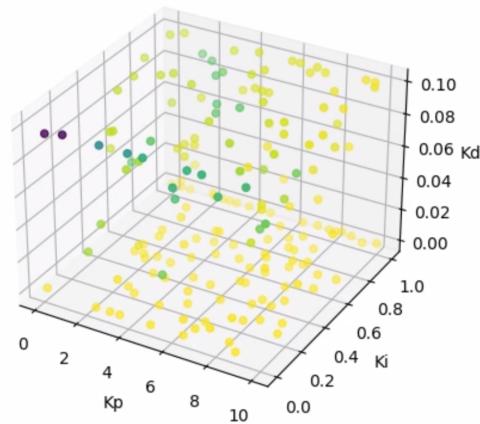


Figure 4.1: Dreidimensionale Darstellung der Bayesian Optimization von PID-Controller-Parametern.

Die Präzision dieser Optimierungsmethode ermöglichte es uns, einen Referenzwert für die Regelungsleistung zu etablieren. Dieser Wert wurde anschließend als Benchmark für das manuelle Feintuning der Hyperparameter unserer Regelung verwendet.

4.1 Hyperparameter

In diesem Abschnitt präsentieren wir eine Übersicht der Hyperparameter, unter denen unsere DDPG-Modelle trainiert wurden. Die Bedeutung dieser Parameter wurde bereits in einem früheren Kapitel detailliert erörtert. Hier fokussieren wir uns auf die spezifischen Einstellungen, die für das Training der unterschiedlichen Modelle verwendet wurden.

Table 4.1: Allgemeiner Vergleich der Hyperparameter für das kleine und große DDPG-Modell

Parameter	Einfaches Modell	Fortgeschrittenes Modell
ALPHA 2.1.7	Höhere Lernrate	Niedrigere Lernrate
WORKER	Standardanzahl	Standardanzahl
ITERATION	Einzeliteration	Einzeliteration
STEPS	Weniger Trainingsschritte	Mehr Trainingsschritte
BATCH_SIZE 2.1.8	Kleinere Batch-Größe	Größere Batch-Größe
EXPLOITATION	Standard-Exploitation	Standard-Exploitation
LAYERS 2.1.2	Weniger Schichten	Mehr Schichten
LAYER_1	Standardanzahl Neuronen	Standardanzahl Neuronen
LAYER_2	Standardanzahl Neuronen	Standardanzahl Neuronen
NOISE 2.2.6	Moderate Exploration	Höhere Exploration
GAMMA 2.2.4	Keine Diskontierung	Keine Diskontierung

Die detaillierten Werte der Hyperparameter werden im folgenden Abschnitt präsentiert, wo wir spezifische Trainingsergebnisse und deren Vergleich darstellen. Die Auswahl und Einstellung dieser Parameter spielen eine entscheidende Rolle im Lernprozess der Modelle und beeinflussen maßgeblich deren Leistung und Effizienz. Im folgenden Abschnitt werden wir die spezifischen Trainingsresultate und ihre Validierung detailliert präsentieren.

4.2 Methodik des Vergleichs

Zur Validierung der Leistung unserer Modelle haben wir uns für eine spezifische Herangehensweise entschieden, die sich von der Standardtrainingsroutine unterscheidet. Anstatt das Modell kontinuierlich durch den gesamten Trainingspool lernen zu lassen, wurde jeder zehnte "EXPLOITATION" Schritt genutzt, um das Modell anhand eines vordefinierten Referenzwertes zu evaluieren. Dieser Referenzwert wurde bewusst nicht in das Training einbezogen, um eine unabhängige Beurteilung der Modellleistung zu ermöglichen. Die deterministische Natur unserer Netzwerke erleichterte diesen Prozess erheblich, da sie nach dem Training konsistente und wiederholbare Ergebnisse liefern. Dies ermöglichte es uns, die Leistung des Modells präzise zu beurteilen, ohne dass eine Mittelung über mehrere Zustände erforderlich war, wie es bei stochastischen Modellen der Fall gewesen wäre. Die deterministische Architektur gewährleistet, dass die gleichen Eingaben stets zu denselben Ausgaben führen, was die Validierung und das Testen vereinfacht und zuverlässiger macht.

4.3 Vergleich und Validierung der Modellleistung

Im Folgenden wird der umfassende Lernprozess des DDPG-Modells dargestellt, wobei ein besonderes Augenmerk auf die ersten 10.000 Iterationen gelegt wird, wie in Abbildung 4.8 zu sehen ist. Dieser Abschnitt des Trainings ist besonders aussagekräftig, da abgesehen von den divergierenden Modellen, alle anderen Modelle nahe am Optimum zu konvergieren scheinen.

Phase 1: Verifikation und Robustheit des DDPG-Modells Die erste Phase unserer Untersuchung dient der Verifikation der Leistungsfähigkeit und Robustheit des Deep Deterministic Policy Gradient (DDPG)-Modells im Kontext der Optimierung von DC-DC-Wandlern. Hierbei konzentrieren wir uns auf zwei Varianten des DDPG-Modells: ein einfacheres, schnell zu trainierendes Modell und ein fortgeschrittenes Modell mit komplexerer Architektur und längeren Trainingszeiten. Unser Hauptziel ist es, zu demonstrieren, dass das DDPG-Modell in der Lage ist, nahezu optimale Lösungen zu erzielen.

Um die Gültigkeit und Zuverlässigkeit unserer Modelle zu bestätigen, vergleichen wir die Ergebnisse des DDPG-Modells mit denen eines anderen bekannten Optimierungsverfahrens, das für seine hohe

Leistung bekannt ist. Diese methodische Triangulation – die Approximation der Lösungen durch zwei grundlegend verschiedene Ansätze – soll nicht nur die Ähnlichkeit der Ergebnisse aufzeigen, sondern auch die Stärke und Anpassungsfähigkeit des DDPG-Modells in diesem speziellen Anwendungsbereich unterstreichen. Wir analysieren, wie sich eine einfache quadratische Belohnungsfunktion, die die Abweichung von der Zielspannung bestraft, auf die Performance dieser Modelle auswirkt und wie dies im Vergleich zu dem alternativen Optimierungsverfahren steht.

Phase 2: Anpassung der Bestrafungsfunktion und Erweiterung des Suchraums In der zweiten Phase unserer Untersuchung wurde die Belohnungsfunktion modifiziert, um eine ausgewogenere Reaktion auf Spannungsabweichungen zu erzielen. Wir haben zwei Ansätze kombiniert: Für Spannungsabweichungen über einem Schwellenwert von 1 wird eine quadratische Bestrafung angewendet, um auf größere Spannungsschwankungen stärker zu reagieren und so die Stabilität des Systems zu verbessern. Für Abweichungen unter diesem Schwellenwert wird eine Bestrafung basierend auf dem absoluten Wert vorgenommen, was das System sensibler für kleinere Spannungsabweichungen macht.

Zusätzlich wurde der Suchraum für die Optimierung erheblich erweitert, um eine breitere Palette von Konfigurationsmöglichkeiten zu erforschen. Der neue Suchraum erstreckt sich nun über deutlich größere Bereiche: - Kp: 0 bis 1000 - Ki: 0 bis 100 - Kd: 0 bis 10

Diese Erweiterung des Suchraums soll aufzeigen, wie sich die Modelle, insbesondere die Bayesianische Optimierung, die bei großen Suchräumen bekannterweise Herausforderungen hat, im Vergleich zum DDPG-Modell verhalten. Es wird interessant zu beobachten sein, wie diese Änderungen die Leistung und Effektivität der Modelle beeinflussen, insbesondere in Bezug auf ihre Fähigkeit, mit größeren und komplexeren Konfigurationsräumen umzugehen.

Phase 3: Miniaturisierung In der dritten Phase unserer Untersuchung widmen wir uns der Miniaturisierung der Netzwerkmodelle. Unser Ziel ist es, die Fähigkeiten eines reduzierten Modells zu evaluieren und zu verstehen, wie es trotz geringerer Komplexität effektiv funktionieren kann. Diese Untersuchungen sind entscheidend, um das Potenzial der kompakten Modelle für den Einsatz in realen Anwendungsszenarien zu erkennen.

4.4 Phase 1: Einfache und Fortgeschrittene Netzwerkmodelle

In diesem Abschnitt werden die Ergebnisse der drei verschiedenen Ansätze zur Optimierung des DC-DC-Wandlers vorgestellt: die Bayesianische Optimierung sowie die kleinen und großen DDPG-Netzwerkmodelle. Die Belohnungen, die mit jeder Konfiguration erzielt wurden, bieten einen Einblick in die Effektivität jedes Ansatzes.

Table 4.2: Erweiterter Vergleich der Hyperparameter für das kleine und große DDPG-Modell unter Einbeziehung des Parameterraums

Parameter	Kleines Modell	Großes Modell
ALPHA	0.001	0.0001
WORKER	12	12
ITERATION	1	1
STEPS	2000	20000
BATCH_SIZE	72	250
EXPLOITATION	10	10
LAYERS	2	12
LAYER_1	158	158
LAYER_2	52	52
NOISE	0.3	0.45
GAMMA	0.0	0.0
Kp-Bereich	0 bis 10	0 bis 10
Ki-Bereich	0.0 bis 1.0	0.0 bis 1.0
Kd-Bereich	0.0 bis 0.2	0.0 bis 0.2

Hyperparameter Die Abbildungen ??, 4.5 und 4.6 zeigen die jeweiligen Belohnungen, die durch die unterschiedlichen Optimierungsverfahren erzielt wurden.

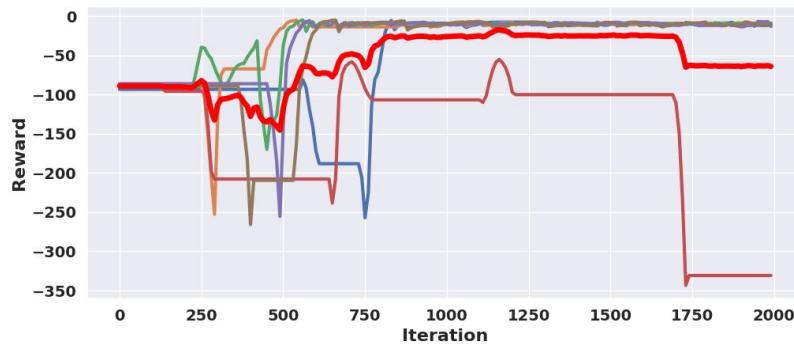


Figure 4.2: Belohnungsentwicklung über die Zeit für das kleine DDPG-Modell.

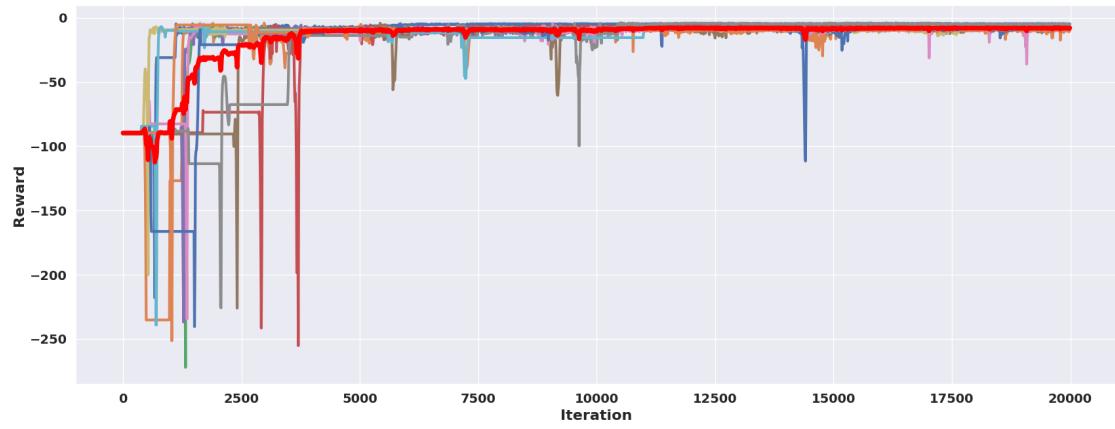


Figure 4.3: Belohnungsentwicklung über die Zeit für das große DDPG-Modell.

Table 4.3: Vergleich der Ergebnisse verschiedener Modelle

Modell	Ergebnis	Parameter
Bayesian Optimization	-3.63	Induktivität: 5.0e-3 Kapazität: 10.0e-6 Kp: 3.9250 Ki: 0.8521 Kd: 0.0139
Kleines DDPG-Modell	-3.9098	Induktivität: 5.0e-3 Kapazität: 10.0e-6 Kp: 2.4306 Ki: 0.6766 Kd: 0.0094
Großes DDPG-Modell	-3.4217	Induktivität: 5.0e-3 Kapazität: 10.0e-6 Kp: 3.4217 Ki: 0.6777 Kd: 0.0127

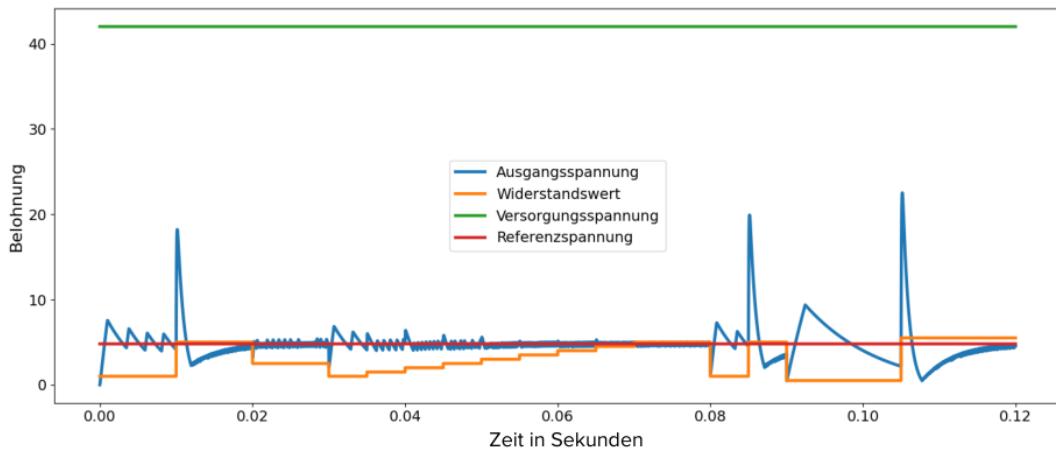


Figure 4.4: Darstellung des Regelungsverhaltens eines durch Bayes'sche Optimierung eingestellten PID-gesteuerten DC-DC-Konverters. Die Grafik zeigt die Belohnungsentwicklung über die Zeit und reflektiert die Anpassung der Ausgangsspannung (blau) an die Referenzspannung (rot) unter Berücksichtigung der Versorgungsspannung (grün) und des Widerstandswertes (orange). Die Ergebnisse verdeutlichen die Effektivität der Bayes'schen Optimierung bei der Feinabstimmung des Reglers für eine stabile Spannungsregelung.

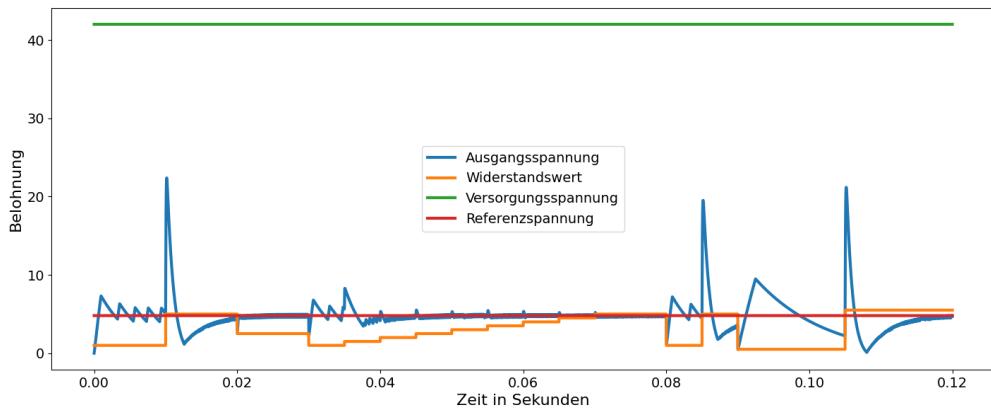


Figure 4.5: Regelungsverhaltens über die Zeit für das kleine DDPG-Modell.

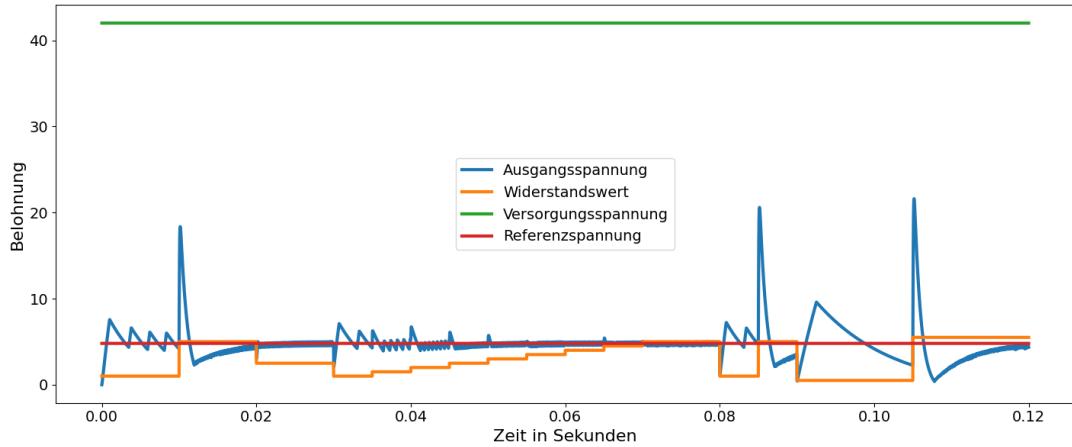


Figure 4.6: Regelungsverhaltens über die Zeit für das große DDPG-Modell.

Analyse der Ergebnisse: Leistungsvergleich der Optimierungsansätze Bei der Betrachtung der erzielten Ergebnisse aus den verschiedenen Optimierungsansätzen wird deutlich, dass sich die Leistungsdifferenzen vor allem im Belohnungswert (Reward) widerspiegeln. Alle drei Ansätze - Bayesianische Optimierung, kleines und großes DDPG-Modell - waren in der Lage, eine optimale oder nahezu optimale Lösung für die gestellte Aufgabe zu finden, wobei die Unterschiede in den tatsächlichen Spannungsverläufen minimal waren.

Interessanterweise zeigte das kleinere DDPG-Modell die geringste Leistung, was sich sowohl in niedrigeren Belohnungswerten als auch in einer weniger stabilen Konvergenz äußerte. Im Gegensatz dazu erreichte das große DDPG-Modell die besten Belohnungswerte, wobei alle Modelle eine gewisse Leichtigkeit bei der Konvergenz zeigten. Die Bayesianische Optimierung lag in ihrer Leistung dazwischen.

Synthese der Ergebnisse Interessanterweise zeigen sowohl die Bayesianische Optimierung als auch die DDPG-Modelle in ihren jeweiligen Anwendungen bemerkenswerte Übereinstimmungen in ihren Ergebnissen. Diese Konsistenz unterstreicht die Robustheit unserer Methoden und die Verlässlichkeit der erzielten Optimierungen. Trotz der unterschiedlichen Herangehensweisen und Techniken, die bei diesen beiden Methoden zum Einsatz kommen, konvergieren ihre Ergebnisse hin zu ähnlichen Lösungen. Dies deutet darauf hin, dass beide Ansätze effektiv die kritischen Bereiche des Parameterraums identifizieren und optimieren, was ein zentrales Ziel unserer Forschung ist.

4.5 Phase 2: Vertiefte Analyse und Optimierung in Erweiterten Parameterräumen

Table 4.4: Überblick über die Hyperparameter des DDPG-Modells inklusive Parameterraum

Parameter	Wert
ALPHA	1×10^{-6}
WORKER	6
ITERATION	1
STEPS	60000
BATCH_SIZE	500
EXPLOITATION	10
LAYERS	24
LAYER_1	158
LAYER_2	52
NOISE	0.45
GAMMA	0.0
MAX_KP	1000.0
MAX_KI	100.0
MAX_KD	10.0

In dieser Phase unserer Untersuchung konzentrieren wir uns auf die vertiefte Analyse des Lernprozesses unter Berücksichtigung der modifizierten Belohnungsfunktion und der Erweiterung des Suchraums. Diese Phase ermöglicht es uns, ein detailliertes Verständnis dafür zu entwickeln, wie unsere Modelle auf diese Änderungen reagieren und sich anpassen.

Anpassung der Belohnungsfunktion Um eine differenziertere Reaktion auf Spannungsabweichungen zu erzielen, haben wir die Belohnungsfunktion angepasst. Für größere Abweichungen, spezifisch über einem Schwellenwert von 1, wird eine quadratische Bestrafung angewendet, um auf diese stärker zu reagieren. Dies verbessert die Stabilität des Systems bei größeren Spannungsschwankungen. Für kleinere Abweichungen, die unter diesem Schwellenwert liegen, wird eine Bestrafung auf Basis des absoluten Wertes vorgenommen. Dies macht das System sensibler für kleinere Spannungsabweichungen und ermöglicht eine feinere Justierung.

Erweiterung des Suchraums Wir haben den Suchraum erheblich erweitert, um eine größere Bandbreite an Konfigurationsmöglichkeiten zu erforschen. Dies erlaubt es uns, die Robustheit unserer Modelle in einem größeren und komplexeren Parameterraum zu testen. Insbesondere wollten wir untersuchen,

wie die Bayesianische Optimierung, die in größeren Räumen typischerweise Herausforderungen begegnet, im Vergleich zum DDPG-Modell abschneidet. Diese Erweiterung hilft auch dabei, die Flexibilität und Anpassungsfähigkeit unserer Modelle in Szenarien mit weniger definierten Parametern zu demonstrieren.

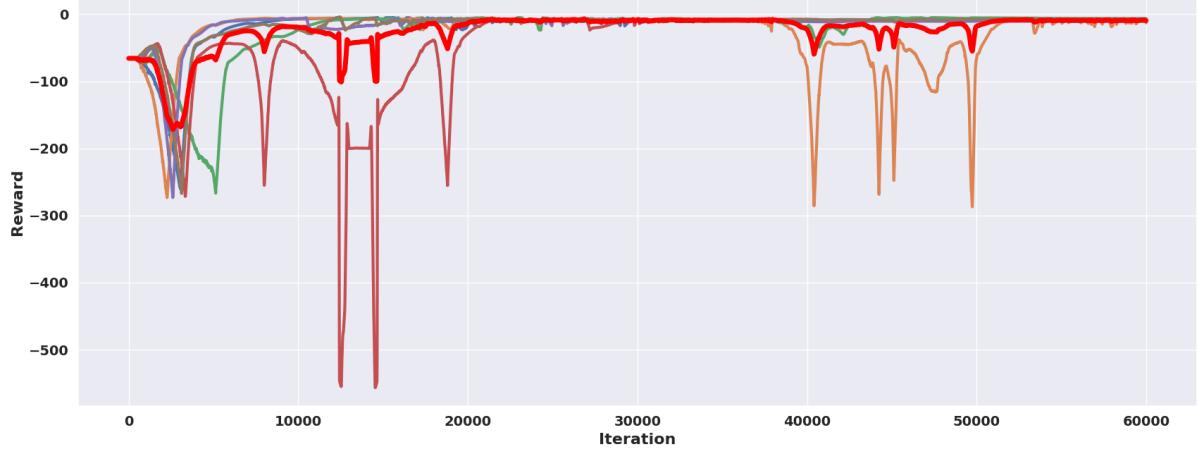


Figure 4.7: Gesamtansicht des Lernprozesses des DDPG-Modells.

Detaillierte Darstellung des Lernprozesses Um die Entwicklung und Konvergenz unserer Modelle zu veranschaulichen, werden wir den Lernprozess detailliert darstellen, einschließlich visueller Darstellungen und Screenshots. Diese Darstellung soll den Fortschritt der Modelle im Laufe des Trainings verdeutlichen und zeigen, wie sie sich den neuen Herausforderungen stellen und anpassen. Durch die Kombination von modifizierter Belohnungsfunktion und erweitertem Suchraum können wir die Vielseitigkeit und Leistungsfähigkeit unserer Ansätze in einer breiten Palette von Anwendungsfällen aufzeigen. Während dieses Trainingsprozesses wurde eine beträchtliche Anzahl von Epochen genutzt. Die gewählte Lernrate war dabei bewusst gering gehalten, um eine stabile Konvergenz zu gewährleisten. Wie aus dem Graphen ersichtlich, haben die meisten Modelle eine signifikante und positive Konvergenz gezeigt, was auf die Effektivität des Lernansatzes und die ausgewählten Hyperparameter schließen lässt.

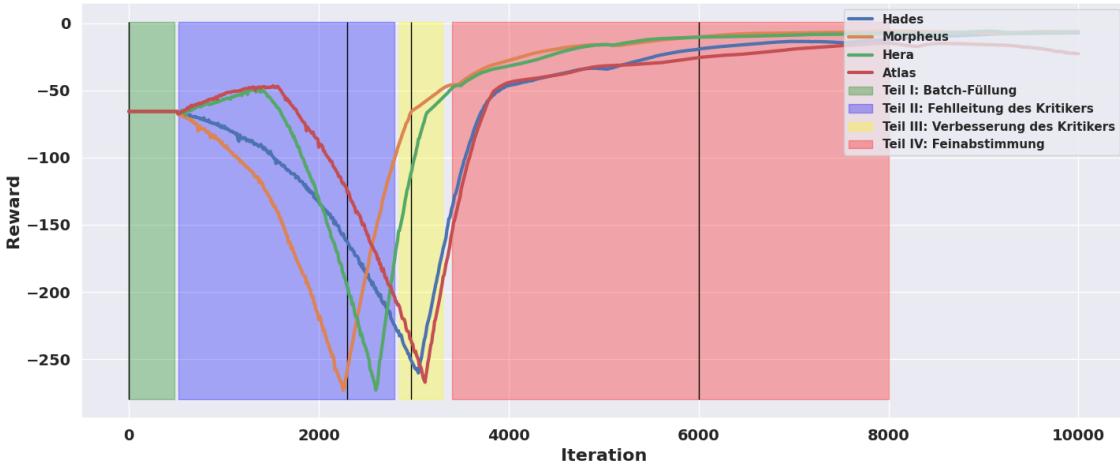


Figure 4.8: Detaillierte Darstellung der Lernprozesse und ausgewählten Stichproben in den verschiedenen Phasen des DDPG-Modells.

Die initiale Selektion für die detaillierte Analyse fokussiert auf die Modelle, die zu Beginn des Trainings die besten Resultate aufweisen. Diese Vorauswahl basiert auf der Performance und der Konvergenzgeschwindigkeit in den ersten 10.000 Iterationen, wie Abbildung 4.8 illustriert. Diese Modelle zeigen eine effektive Anpassung an den erweiterten Parameterraum und lassen auf ein optimiertes Lernverhalten schließen.

4.5.1 Teil I. Anfängliche Lernprozesse und Batch-Füllung

In Abbildung 4.8 ist die erste Phase des Lernprozesses, markiert mit grün, deutlich zu erkennen. In dieser Phase zeigt sich ein flacher Verlauf der Lernkurven, was darauf zurückzuführen ist, dass der Batch zunächst gefüllt werden muss, bevor das eigentliche Lernen beginnen kann. Diese Anfangsphase ist essenziell, da wir hier den Stochastischen Gradientenabstieg (siehe Abschnitt 2.1.4) verwenden. Das Modell nutzt eine Vielzahl von Zustandsübergängen, um den Gradienten zu berechnen, was bedeutet, dass eine ausreichende Menge an Daten gesammelt werden muss, bevor eine Optimierung der Parameter effektiv stattfinden kann.

Table 4.5: Stichprobe des Modells Morpheus bei Iteration 0

Parameter	Wert
Iteration	0
Reward	-65.68
Action	[500.00000022, 50.00000000, 4.99999991]
Induktivität	5.0×10^{-3}
Kapazität	10.0×10^{-6}

Ein weiterer interessanter Punkt, der in dieser Phase zu beobachten ist, betrifft die Ausgangswerte der Aktionen, die nahe bei [500, 50, 5] liegen. Dies ist auf die Aktivierungsfunktion zurückzuführen, die am Ende der 12 Schichten des Netzwerks angewendet wird. Da die Eingangswerte nicht normalisiert wurden und relativ klein sind, werden anfangs wahrscheinlich nur wenige Neuronen aktiviert, insbesondere unter Verwendung von Aktivierungsfunktionen wie ReLU, die das Überschreiten einer bestimmten Schwelle erfordern. Daraus folgt, dass die meisten Modelle anfänglich von ähnlichen Werten ausgehen, was eine Herausforderung für den Anfang des Lernprozesses darstellt.

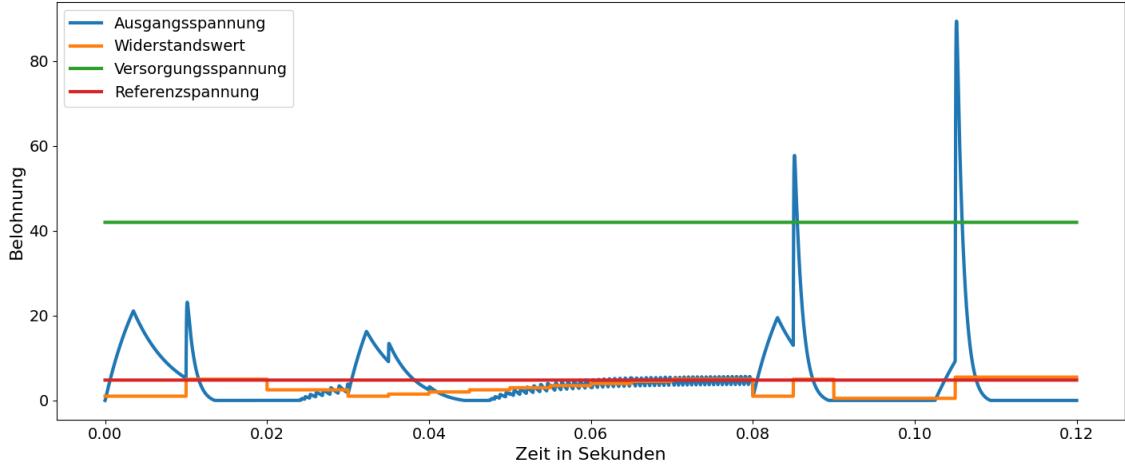


Figure 4.9: Stichprobe des Modells Morpheus bei Iteration 0 / Batchfüllung

Die erste Stichprobe zeigt die Anfangsversuche des PID-Reglers, eine Regelung durchzuführen. Wie in Abbildung 4.9 erkennbar, gelingt es dem Regler noch nicht, die Ausgangsspannung effektiv zu stabilisieren. Die blaue Linie, die die Ausgangsspannung darstellt, haftet nicht an der roten Referenzlinie, was auf eine unzureichende Steuerung hinweist. Insbesondere bei großen Stromspitzen überschreitet die Spannung die Versorgungsspannung und erreicht Werte bis zu 90 Volt, manchmal fällt sie auch auf null ab. Dies würde normalerweise als suboptimales Regelverhalten eingestuft.

Trotzdem, wenn die Störungen nicht zu groß sind und dem System ausreichend Zeit gegeben wird, beginnt es, sich auf den gewünschten Wert einzupendeln, obwohl noch starke Oszillationen sichtbar sind. Dies deutet darauf hin, dass das System das Potenzial hat, sich zu stabilisieren, obwohl der aktuelle Regelungsansatz noch verbesserungsbedürftig ist.

Table 4.6: Stichprobe des Modells Morpheus bei Iteration 2200

Parameter	Wert
Iteration	2200
Reward	-260.67
Action	[514.42673523, 50.32317324, 4.64697339]
Induktivität	5.0×10^{-3}
Kapazität	10.0×10^{-6}

4.5.2 Teil II. Fehlleitung durch den Kritiker

Die zweite Phase der Untersuchung offenbarte eine bemerkenswerte Dynamik zwischen dem Akteur und dem Kritiker. Während der Akteur begann, sein Verhalten basierend auf dem Feedback des Kritikers anzupassen, führten diese Änderungen zunächst zu keiner Verbesserung der Zielerreichung. Dies zeigte, dass der Akteur anfänglich den Empfehlungen eines noch nicht ausreichend trainierten Kritikers folgte, was zu suboptimalen Entscheidungen und einer Verschlechterung der Leistung gegenüber dem Referenzwert führte. Der signifikante Abfall des Rewards und die extremen Ausschläge der Ausgangsspannung, die in Abbildung 4.10 dargestellt sind, verdeutlichen die Konsequenzen der Fehlleitung. Die Ausgangsspannung erreichte Spitzen von bis zu 250 Volt, was in der realen Anwendung wahrscheinlich zu einer Überlastung des Systems geführt hätte. Diese Phase illustriert, wie kritisch es ist, dass der Kritiker präzise und zuverlässige Rückmeldungen liefert, um eine effektive Anpassung des Akteurs zu gewährleisten. Diese Beobachtungen wurden durch die Stichprobe des Modells Morpheus bei Iteration 2200 untermauert, wie in Tabelle 4.6 dargestellt.

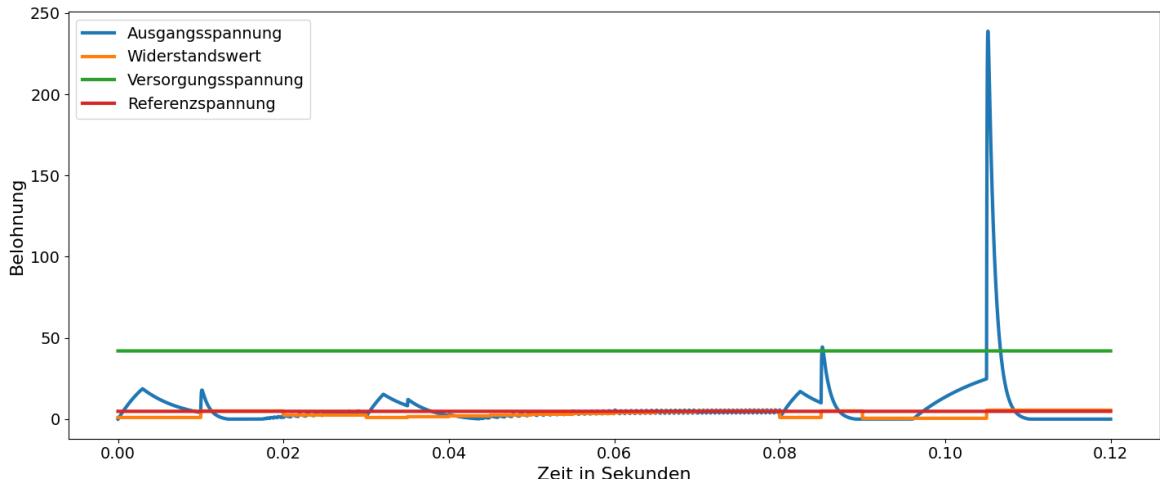


Figure 4.10: Auffällige Abweichungen im Regelverhalten des Modells Morpheus während Phase II.

4.5.3 Teil III. Feinjustierung und Konvergenz in Richtung Optimum

Mit fortschreitendem Training begann der Kritiker jedoch, die Lage korrekter einzuschätzen, und seine Bewertungen wurden zunehmend präziser. In der dritten Phase konnte beobachtet werden, wie der Akteur endlich begann, die angemessenen Anpassungen vorzunehmen, was zu einer allmählichen Verbesserung und Annäherung an die optimale Lösung führte. Die Richtung der Anpassungen stimmte nun mit den tatsächlichen Bewertungen des Kritikers überein, was zu einer iterativen Verbesserung des Gesamtverhaltens führte und den Lernprozess in die richtige Richtung lenkte.

Nachdem das Modell Morpheus 3000 Iterationen durchlaufen hatte, begann es, deutliche Fortschritte in der Regelungsleistung zu zeigen. Die vom Kritiker geleiteten Anpassungen des Akteurs führten zu einer bemerkenswerten Verbesserung der Belohnungswerte, was eine gezielte Annäherung an die gewünschte Referenzspannung mit einer verringerten Oszillationsamplitude signalisiert. Die Daten in Tabelle 4.7 und die zugehörigen Regelungsausschläge in Abbildung 4.11 legen nahe, dass trotz der gelegentlich über 100 Volt hinausgehenden Spitzenwerte die Regelungsstrategie des Akteurs die richtige Richtung einschlägt.

Die Bedeutung der Belohnungsfunktion in diesem Prozess kann nicht genug betont werden. Obwohl die PID-Koeffizienten andere Werte aufweisen als in der anfänglichen Lernphase, führt die Feinjustierung der Belohnungsfunktion zu einem Verhalten, das in seiner Leistung fast dem der ersten Phase entspricht. Es ist faszinierend zu sehen, wie das System trotz veränderter PID-Konfiguration zu einem ähnlichen Ausgangsverhalten konvergiert, was die Wichtigkeit und Wirksamkeit der Belohnungsfunktion in der Lernstrategie unterstreicht.

Table 4.7: Stichprobe des Modells Morpheus bei Iteration 3000

Parameter	Wert
Iteration	3000
Belohnung	-64.33275232194337
Aktion	[535.75499356, 51.67935919, 4.21606518]
Induktivität	5.0×10^{-3}
Kapazität	10.0×10^{-6}

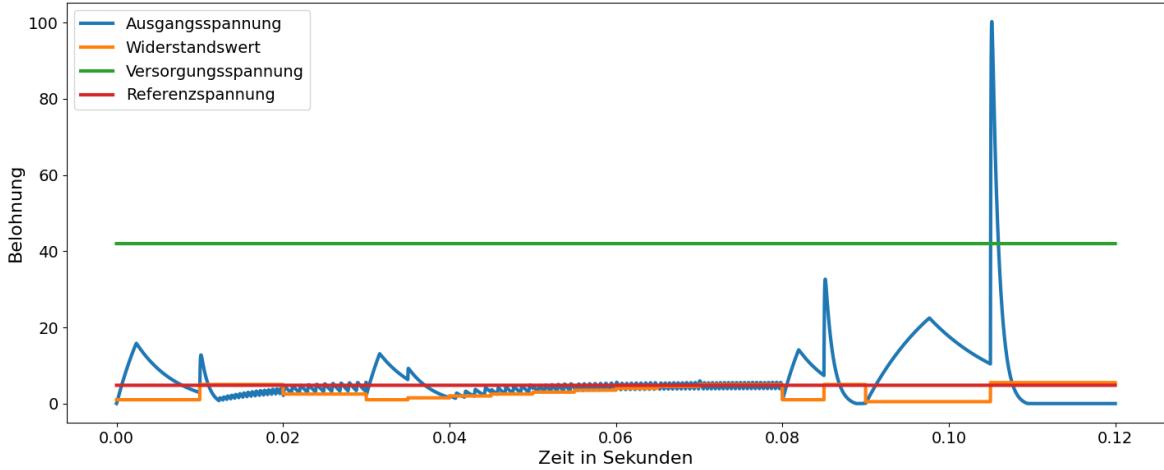


Figure 4.11: Verbesserung der Regelungsleistung des Modells Morpheus während der dritten Phase, illustriert durch die Annäherung der Ausgangsspannung an die Referenzspannung und die Verringerung der Oszillationen.

4.5.4 Teil IV. Fortgeschrittene Konvergenz und Feinabstimmung

Die vierte und letzte Phase des Lernprozesses zeichnete sich durch eine langsame, aber stetige Annäherung des Akteurs an das Optimum aus 4.12. Die Verhaltensänderungen wurden feiner und gezielter, was darauf hindeutet, dass das Modell begann, die subtilen Nuancen der Umgebung zu verstehen und seine Aktionen entsprechend anzupassen. Es war ein langsamer Prozess der Verfeinerung, der darauf abzielte, die optimalen Einstellungen zu finden. Interessanterweise war in dieser Phase auch eine gelegentliche Divergenz bei einem der Agenten zu beobachten. 4.7 Dies könnte darauf hinweisen, dass trotz der allgemeinen Tendenz zur Konvergenz immer noch das Potential für Instabilitäten oder für das Erkunden von neuen, unerwarteten Lösungspfaden bestand.

Table 4.8: Stichprobe des Modells Morpheus bei Iteration 6000

Parameter	Wert
Iteration	6000
Belohnung	-10.21
Aktion	[634.09879804, 55.28446324, 2.65851244]
Induktivität	5.0×10^{-3}
Kapazität	10.0×10^{-6}

Mit weiteren Fortschritten im Trainingsprozess erreicht das Modell Morpheus nach 6000 Iterationen eine Phase, in der eine signifikante Konvergenz erkennbar ist. Die Belohnungswerte verbessern sich weiter, und die Ausgangsspannung nähert sich stabil der Referenzspannung an, ohne die Versorgungsspannung zu überschreiten 4.12. Die Daten in Tabelle 4.8 zeigen, dass die Anpassungen des Akteurs zu einer angemessenen Regelungsleistung führen und die Oszillationen der Spannungswerte minimiert werden. Die Ergebnisse dieser Phase sind vielversprechend und lassen auf eine erfolgreiche Anpassung des Modells schließen.

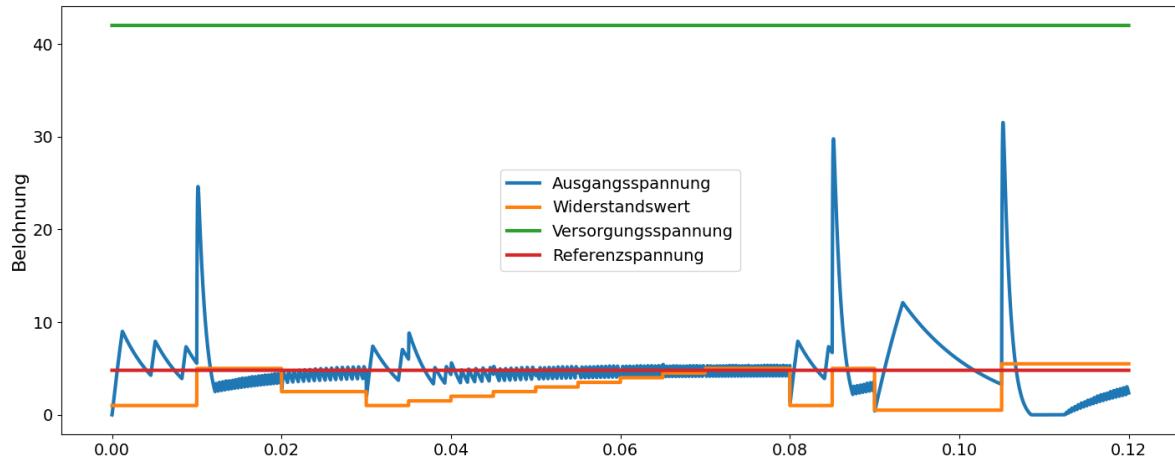


Figure 4.12: Die fortgeschrittene Konvergenz des Modells Morpheus mit einer sichtbaren Verbesserung der Regelungsleistung.

4.5.5 Teil V. Erzielung einer nahezu optimalen Regelungsleistung

Während der entscheidenden Endphase des Trainingsprozesses verzeichneten wir bei einem unserer Modelle, welches wir im Folgenden als Modell X bezeichnen, eine stetige Verbesserung der Regelungsleistung. Diese Beobachtung wird durch die in Abbildung 4.7 dokumentierte, kontinuierliche Leistungssteigerung bestätigt. Durch konsequente Feinabstimmung und die Auswahl der leistungsfähigsten Modelle zu verschiedenen Zeitpunkten des Trainings konnte ein herausragendes Ergebnis erzielt werden: Das Modell X erreichte eine maximale Belohnung von -3.99. Dies deutet auf eine Performance hin, die dem optimalen Steuerungsverhalten sehr nahekommmt.

Table 4.9: Maximale Leistung des Modells X nach umfangreichen Trainingsiterationen

Parameter	Wert
Iteration	Geschätzt
Belohnung	-3.9928830028461824
Aktion	[434.82286483, 91.83493555, 1.52657181]
Induktivität	5.0×10^{-3}
Kapazität	10.0×10^{-6}

Die Erreichung einer solch hohen Belohnung nach einer langen Reihe von Trainingsiterationen bestätigt die Effektivität unseres Ansatzes zur Modelloptimierung. Die Performance 4.13 von Modell X stellt einen bedeutenden Meilenstein dar, der zeigt, dass das Modell nicht nur theoretisch, sondern auch in der simulierten Anwendung in der Lage ist, sich effektiv anzupassen und zu optimieren. Dieses Ergebnis betont die Relevanz und Präzision unserer Belohnungsfunktion sowie die Robustheit des DDPG-Modells unter komplexen und anspruchsvollen Bedingungen.

4.5.6 Synthese der Ergebnisse

Dynamik zwischen Akteur und Kritiker Die Interaktion zwischen Akteur und Kritiker, insbesondere in der zweiten Phase, zeigte die Komplexität des Lernprozesses auf, bei dem die Anfälligkeit für Fehlleitung ohne angemessene Leitung evident wurde.

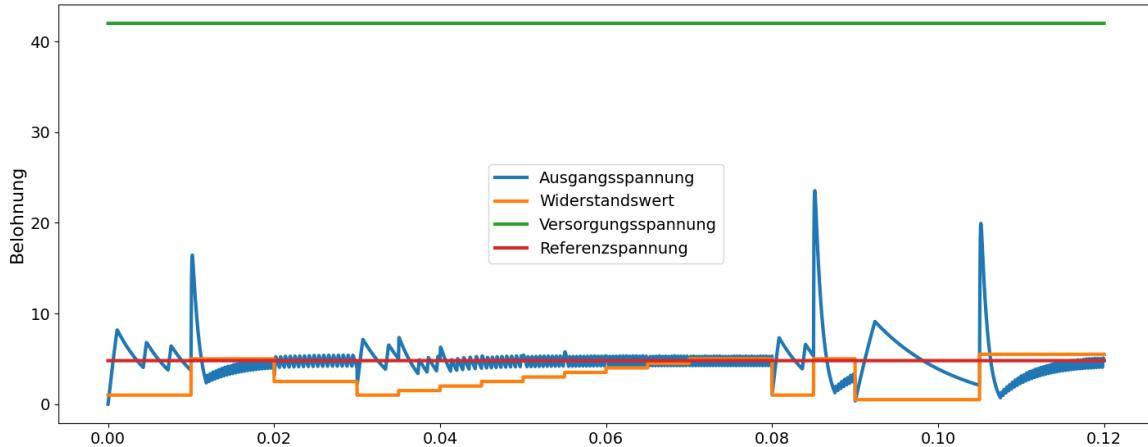


Figure 4.13: Darstellung der finalen Leistungssteigerung des Modells X, welche die nahezu optimale Regelungsleistung illustriert.

Konvergenz und Feinjustierung Die dritte Phase demonstrierte, dass trotz anfänglicher Fehlritte der Algorithmus fähig ist, sich anzupassen und zu verbessern, was sich in einer konstanten Annäherung an die gewünschte Leistung manifestierte.

Erreichen einer hohen Leistung Die letzte Phase unterstrich die Potenz des Algorithmus, auch unter längeren und anspruchsvollen Trainingsbedingungen eine nahezu optimale Regelungsleistung zu erzielen, was durch das Modell X hervorragend dargestellt wurde.

4.6 Phase 3: Miniaturisierung und Leistungsanalyse

Table 4.10: Konfiguration und Ergebnisse des miniaturisierten Modells

Parameter	Wert
ALPHA	1×10^{-5}
WORKER	10
ITERATION	1
STEPS	30000
BATCH_SIZE	250
EXPLOITATION	10
LAYERS	3
LAYER_1	10
LAYER_2	3
NOISE	0.4
GAMMA	0.0
Maximale Belohnung	-4.29
Entsprechende Aktion	[6.84119821, 0.75633377, 0.02226542]

Die Ergebnisse der Miniaturisierung sind bemerkenswert, da das verkleinerte Modell eine maximale Belohnung von -4.295256034278478 erreichte, was auf eine nahezu optimale Regelungsleistung hindeutet. Die entsprechenden Aktionen und detaillierten Konfigurationen des Modells sind in Tabelle 4.10 aufgeführt.

Die Leistung des miniaturisierten Modells, wie in Abbildung 4.14 gezeigt, demonstriert eindrucksvoll, dass eine effiziente Modellminiaturisierung ohne erheblichen Leistungsverlust möglich ist. Diese Erkenntnis ist besonders relevant für praxisnahe Anwendungen, wo kompakte Modelle aufgrund von Ressourcenbeschränkungen bevorzugt werden.

Die Ergebnisse wurden durch eine Kombination aus niedriger Lernrate und einer hohen Anzahl von Trainingsepochen erzielt. Die erzielten Ergebnisse 4.14 weisen darauf hin, dass das Modell trotz der

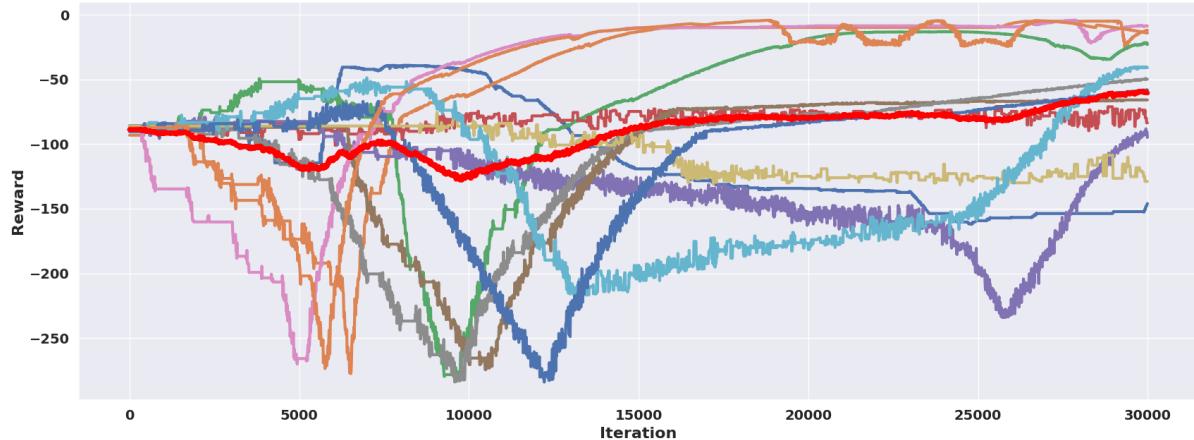


Figure 4.14: Leistungskurve des miniaturisierten Modells über die Trainingsdauer.

frühzeitigen Beendigung des Trainings bereits eine beachtliche Leistung demonstrierte. Dies legt nahe, dass mit einer Fortsetzung des Trainingsprozesses und einer weiteren Feinabstimmung der Hyperparameter möglicherweise noch höhere Performanzwerte hätten erreicht werden können. Die vorliegenden Ergebnisse bestätigen somit nicht nur die Effektivität des eingesetzten Trainingsansatzes, sondern auch das Potenzial für weitergehende Verbesserungen und Optimierungen.

Ein weiterer wichtiger Aspekt ist das Risiko des Overfittings bei komplexeren Modellen, wie bereits in Kapitel 2.1.5 diskutiert. Overfitting tritt auf, wenn ein Modell zu sehr auf die spezifischen Trainingsdaten ausgerichtet ist und dadurch seine Fähigkeit verliert, auf neue, unbekannte Daten angemessen zu reagieren. Es ist eine bekannte Tatsache, dass einfachere Modelle häufig eine bessere Generalisierungsfähigkeit aufweisen, da sie weniger anfällig für das Auswendiglernen spezifischer Trainingsdaten sind. Unsere Ergebnisse zeigen, dass das miniaturisierte Modell trotz seiner Einfachheit eine gute Leistung erbringen konnte, was die Wichtigkeit einer ausgewogenen Modellkomplexität in der Modellentwicklung hervorhebt.

Insgesamt legen diese Beobachtungen nahe, dass es vorteilhaft sein kann, sich auf die Optimierung von Hyperparametern zu konzentrieren, um kleinere, aber leistungsfähige Modelle zu entwickeln.

Chapter 5

Diskussion

In dieser Studie wurde der Deep Deterministic Policy Gradient (DDPG) Algorithmus zur Optimierung von PID-Reglern in DC-DC-Wandlersystemen erfolgreich angewendet. Die Ergebnisse heben die Effektivität des DDPG-Ansatzes hervor, insbesondere im Hinblick auf die Sensibilität bezüglich der Hyperparameterwahl und der Gestaltung der Belohnungsfunktion. Die systematische Untersuchung und das Training des DDPG-Algorithmus über mehrere Phasen haben zu einem tiefgreifenden Verständnis der Regelungsdynamik eines PID-regulierten DC-DC-Konverters geführt. Die schrittweise Entwicklung vom ersten Erlernen der Umgebungsbedingungen über die Fehlleitung durch einen untrainierten Kritiker bis hin zur Feinabstimmung und nahezu optimalen Regelungsleistung hat mehrere Schlüsselemente des maschinellen Lernens hervorgehoben.

Vergleich mit Bayesianischer Optimierung Ein interessanter Aspekt der Untersuchung war der Vergleich des DDPG-Ansatzes mit der Bayesianischen Optimierung. Obwohl es keine direkte Baseline-Methode gab, ermöglichte dieser Vergleich eine Einschätzung der Effektivität beider Ansätze. Beide Verfahren führten zu ähnlichen Ergebnissen, was darauf hindeutet, dass der DDPG-Algorithmus eine gleichwertige, wenn nicht sogar überlegene Alternative in bestimmten Szenarien sein kann.

Phasen der Untersuchung Im Verlauf der Studie wurden verschiedene Phasen betrachtet, von der initialen Anwendung des DDPG-Algorithmus über die Analyse verschiedener Netzwerkgrößen bis hin zur Miniaturisierung der Modelle. Die Robustheit des DDPG-Algorithmus wurde dabei deutlich, insbesondere seine Fähigkeit, auch bei reduzierten Modellgrößen effektiv zu konvergieren.

Miniaturisierung und Leistung Die Ergebnisse aus der Phase der Miniaturisierung sind besonders bemerkenswert. Trotz der signifikanten Reduktion in der Größe des Netzwerks blieb die Leistung der Modelle hoch. Dies unterstreicht das Potenzial des DDPG-Ansatzes für Anwendungen, bei denen eine kompakte und effiziente Modellarchitektur erforderlich ist.

Chapter 6

Zukünftige Arbeit

Die Ergebnisse dieser Studie eröffnen verschiedene Wege für zukünftige Forschungen im Bereich der Regelungstechnik und des maschinellen Lernens. Die Modularität und Generalisierbarkeit des verwendeten Ansatzes bieten zahlreiche Möglichkeiten für weiterführende Untersuchungen.

Anwendung auf Verschiedene Schaltungen Ein vielversprechender Bereich für zukünftige Arbeiten ist die Anwendung des entwickelten Reinforcement Learning-Algorithmus auf eine Vielzahl von Schaltungen. Die Flexibilität des Systems ermöglicht es, mit unterschiedlichen Schaltungsdesigns und Belohnungsfunktionen zu experimentieren, um ihre Wirksamkeit in verschiedenen Kontexten zu testen.

Optimierung der Netzwerkarchitektur Die Ergebnisse hinsichtlich der Miniaturisierung der Netzwerke legen nahe, dass eine weitere Optimierung der Netzwerkarchitektur möglich ist. Insbesondere könnte die Kombination eines komplexeren Kritikers mit einem minimalisierten Akteur untersucht werden. Zusätzlich wäre die Anwendung von spezialisierten Netzwerkoptimierungsalgorithmen, die auf die Reduktion der Netzwerkgröße abzielen, ein interessantes Forschungsfeld.

Validierung in der Realen Welt Ein kritischer Schritt für die Zukunft ist die Überprüfung der Simulationsergebnisse in realen Anwendungen. Es ist unerlässlich, die in der simulierten Umgebung erzielten Ergebnisse in der realen Welt zu validieren und dabei zusätzliche Variablen wie Rauschen und unvorhersehbare Betriebsbedingungen einzubeziehen. Dies würde nicht nur die Zuverlässigkeit des Algorithmus bestätigen, sondern auch seine Anwendbarkeit in praktischen Szenarien.

Erweiterte Reinforcement Learning Strategien Es wäre lohnenswert, alternative Reinforcement Learning-Strategien und -Architekturen zu erforschen, um die Effektivität und Effizienz des Lernprozesses weiter zu verbessern. Insbesondere könnten Anpassungen der Lernrate oder die Integration von Techniken zur Vermeidung von Overfitting den Algorithmus robuster und vielseitiger machen.

Parallele Architekturen und Simulationen Eine mögliche Erweiterung des aktuellen Ansatzes wäre die Nutzung paralleler Architekturen mit mehreren Kritikern und Akteuren. Dies könnte die Effizienz und Performance in komplexen Systemen steigern, insbesondere durch die Reduktion des benötigten Speichers und der Rechenzeit. Solche parallelen Strukturen könnten insbesondere bei größeren Schaltungen von Vorteil sein, um den Lernprozess zu beschleunigen.

Anpassung der Belohnungsfunktion Die Bedeutung einer maßgeschneiderten Belohnungsfunktion ist im Rahmen dieser Studie deutlich hervorgetreten. Zukünftige Forschungen könnten sich darauf konzentrieren, die Belohnungsfunktion speziell auf die erwartete Nutzung des Systems abzustimmen, wie z.B. die Anpassung an bestimmte Verhaltensmuster oder Betriebsbedingungen.

Optimales Schaltungsentwurf Ein weiterer Ansatz könnte darin bestehen, die Architektur des Algorithmus zur Entwicklung optimaler Schaltungsentwürfe zu nutzen. Anstatt sich lediglich auf die Regelung bestehender Schaltungen zu konzentrieren, könnte der Algorithmus dazu verwendet werden, Parameter für den Entwurf effizienterer und leistungsfähigerer Schaltungen zu ermitteln.

Erweiterung der Anwendbarkeit Die Verallgemeinerungsfähigkeit des Ansatzes legt nahe, dass er auch auf andere Arten von Schaltungen und Regelungsproblemen anwendbar sein könnte. Die Möglichkeit, den Algorithmus für eine breite Palette von Anwendungen zu nutzen, eröffnet ein weites Feld für zukünftige Experimente und Entwicklungen.

Insgesamt bieten die Ergebnisse dieser Arbeit eine solide Basis für eine Vielzahl von zukünftigen Forschungsprojekten, die darauf abzielen, die Leistungsfähigkeit und Effizienz von Regelungssystemen durch maschinelles Lernen weiter zu verbessern.

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter verfasst habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, June 19, 2024

Bibliography

- [1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Library of Congress Control Number: 2018947636. Yorktown Heights, NY, USA: Springer International Publishing AG, part of Springer Nature, 2018. ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.
- [2] Muhanad D. Hashim Almawlawe, Muhammad Al-badri, and Issam Hayder Alsakini. “Performance Improvement of a DC/DC Converter Using Neural Network Controller in comparison with Different Controllers”. In: (2023).
- [3] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.
- [4] François Chollet. *Deep Learning with Python, Second Edition*. Covers topics from the fundamentals of machine learning and deep learning to advanced topics in computer vision and text processing. Shelter Island, NY: Manning Publications Co., 2021. ISBN: 9781617296864. URL: <https://www.manning.com/>.
- [5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [6] Jeff Heaton. *Introduction to the Math of Neural Networks*. Heaton Research Inc., 2012. ISBN: 978-1475190878.
- [7] Dan Klein and Pieter Abbeel. *Markov Decision Processes II - CS 188: Artificial Intelligence*. Online. Lecture notes available: <http://ai.berkeley.edu>. University of California, Berkeley, Year of Lecture, e.g., 2021.
- [8] Timothy P. Lillicrap et al. “Continuous Control with Deep Reinforcement Learning”. In: (2016).
- [9] Kevin Sebastian Luck et al. “Improved Exploration through Latent Trajectory Optimization in Deep Deterministic Policy Gradient”. In: *arXiv preprint arXiv:1911.06833* (2019).
- [10] Miguel Morales. *Grokking Deep Reinforcement Learning*. Manning Publications Co., 2020.
- [11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson Education, Inc., 2021.
- [12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. Adaptive Computation and Machine Learning. Licensed under Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic License. Cambridge, Massachusetts; London, England: The MIT Press, 2018. ISBN: 9780262039246. URL: <https://lccn.loc.gov/2018023826>.
- [13] Junta Wu and Huiyun Li. “Aggregated Multi-deep Deterministic Policy Gradient for Self-driving Policy”. In: (2018).