

BACHELORARBEIT

Maja Franz

Reinforcement Learning mit parametrisierten Quantenschaltkreisen

31. August 2021

Fakultät:	Informatik und Mathematik
Studiengang:	Bachelor Informatik
Abgabefrist:	31. August 2021
Betreuung:	Prof. Dr. Wolfgang Mauerer
Zweitbegutachtung:	Prof. Dr. Kai Selgrad

Kurzfassung

In dem letzten Jahrzehnt konnten in den Forschungsgebieten des Reinforcement Learnings (RLs) und *Quantencomputings* große Fortschritte erzielt werden. Dabei haben sich parametrisierte Quantenschaltkreise (PQCs) als Modell des quantenbasierten maschinellen Lernens etabliert. Diese haben das Potenzial, in naher Zukunft eine Überlegenheit von Quantencomputern gegenüber klassischen zu zeigen.

In dieser Arbeit werden Verfahren und Konzepte des quantenbasierten RLs vorgestellt und daraus entwickelte Technologien experimentell untersucht. Die Ergebnisse zeigen, dass quantenbasierte Ansätze, ähnlich zu klassischen, einfache RL-Anwendungen optimal lösen können. Es wird vermutet, dass diese Verfahren für auch auf realen Quantensystemen mit geringem Fehlereinfluss durchgeführt werden können, wenn die verwendeten Quantenschaltkreise eine vergleichbar geringe Größe haben.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung	2
1.3. Überblick	2
2. Theoretische Grundlagen	3
2.1. Reinforcement Learning (RL)	3
2.1.1. Markov-Entscheidungsprozess (MDP)	3
2.1.2. Q-Learning	6
2.1.3. Deep Q-Learning	7
2.2. Quantencomputer und deren Berechnungen	11
2.2.1. Qubits und Superposition	11
2.2.2. Verschränkung mehrerer Qubits	11
2.2.3. Quantenschaltkreise	13
2.2.4. Quantengatter	13
2.2.5. Messungen in der Standardbasis	17
3. Maschinelles Lernen mit parametrisierten Quantenschaltkreisen (PQCs)	19
3.1. Hybrider Ansatz für Quanten-RL	19
3.2. Struktur des PQCs für maschinelles Lernen	21
3.3. Kodierung klassischer Daten	22
3.3.1. Binäre Kodierung	23
3.3.2. Komponentenweise Kodierung	24
3.4. Extrahieren von Quantendaten	25
3.5. Gradientenberechnung	26
4. Reproduktion verwandter Arbeiten	29
4.1. Beschreibung der Lernumgebungen	29
4.1.1. <i>CartPole</i>	29
4.1.2. <i>Blackjack</i>	31
4.2. Umsetzung von quantenbasiertem RL	32
4.3. Resultate eines klassischen RL-Verfahrens	32
4.4. Resultate eines Quanten-RL-Verfahrens	35
4.4.1. Architektur des Quantenschaltkreises	35
4.4.2. Resultate in <i>CartPole</i>	36
4.4.3. Resultate in <i>Blackjack</i>	38
4.4.4. Diskussion	40

4.5. Resultate einer alternativen PQC-Struktur	41
4.5.1. Architektur des Quantenschaltkreises	41
4.5.2. Resultate in <i>CartPole</i>	41
4.5.3. Diskussion	44
5. Weiterführende experimentelle Untersuchungen	47
5.1. Initiale Ergebnisse mit Quantencomputern	47
5.2. Nichtlineare Parametrisierung von PQC's	48
5.3. Schichtweises Lernen für Quanten-RL	51
5.4. Diskussion	53
6. Fazit und Erkenntnisse	55
6.1. Zusammenfassung	55
6.2. Ausblick	55
A. Anhang: Liste der Quantengatter	57
Abbildungsverzeichnis	60
Literatur	61

Abkürzungsverzeichnis

RL Reinforcement Learning

NISQ Noisy Intermediate Scale Quantum

PQC parametrisierter Quantenschaltkreis

MDP Markov-Entscheidungsprozess

TD Temporal-Difference

DQN Deep Q-Network

NN neuronales Netz

1. Einleitung

Dieses Kapitel bietet eine Einführung in die Thematik. Es beschreibt die Möglichkeiten von Quantencomputern hinsichtlich maschinellen Lernens. Speziell wird dabei auf Reinforcement Learning (RL) (dt. etwa „bestärkendes Lernen“) eingegangen. Ziel dieser Arbeit ist eine Analyse und Evaluation von quantenbasierten Verfahren für RL.

1.1. Motivation

Mit Hilfe von RL konnten bereits übermenschliche Leistungen in komplexen Spielen, wie Schach, Go [Sil+18], Dota [Ope+19] oder StarCraft [Vin+19], erzielt werden. Außerdem führte RL in spezifischen Themengebieten, wie beispielsweise der automatischen Indizierung von Datenbanken [LM20] oder dem Entwurf von Chips [Mir+20], zu nahezu optimalen Ergebnissen.

Parallel zu den Verbesserungen im Gebiet des RLs gewann auch *Quantencomputing* im letzten Jahrzehnt an Bedeutung. Der Stand der Technik sind sogenannte *Noisy Intermediate Scale Quantum (NISQ)*-Geräte. Das sind die ersten Quantencomputer, die potenziell klassischen Computern überlegen sind. Eine Überlegenheit von Quantencomputern ist erreicht, sobald Quantencomputer eine Aufgabe in realistischer Zeit lösen können, während klassische Computer eine nicht realisierbare Rechenzeit benötigen [Pre18]. Erste Versuche, die diese Überlegenheit belegen können, wurden 2019 von Arute et al. [Aru+19] unternommen. Aktuelle Geräte sind allerdings in der Größe ihrer Quantenschaltkreise noch durch Fehler beschränkt, die auf elektromagnetisches Rauschen (*engl. noise*) zurückzuführen sind. Ungeachtet dessen gibt es bereits spezifische Entwürfe von Anwendungen für NISQ-Geräte, von denen vermutet wird, dass diese bereits in naher Zukunft eingesetzt werden können [Pre18; Moh+17].

Ein Modell, mit dem konkrete Quantenalgorithmen für NISQ-Geräte definiert werden können, sind parametrisierte Quantenschaltkreise (PQCs) [Ben+19]. Diese können, ähnlich zu klassischen Neuronalen Netzen (NNs) in maschinellem Lernen und damit auch in RL eingesetzt werden [Mit+18]. Da PQCs bereits auf realer Hardware angewandt werden können, sind sie vielversprechende Kandidaten, um in naher Zukunft eine Überlegenheit von quantenbasierten Ansätzen in komplexen Aufgaben, wie RL, zu zeigen [SJD21].

1.2. Zielsetzung

In dieser Arbeit werden verschiedene Konzepte für RL untersucht, welche auf einem *hybriden* Ansatz aus quantenbasierten und klassischen Elementen basieren. Dazu werden PQCs aus verwandten Arbeiten reproduziert und mit klassischen NNs hinsichtlich ihrer Leistung in definierten RL-Umgebungen verglichen. Das Ziel dieser Arbeit ist eine Analyse und Evaluation der betrachteten Verfahren für quantenbasiertes RL.

1.3. Überblick

Diese Arbeit ist folgendermaßen gegliedert: Das nächste Kapitel vermittelt die theoretischen Grundlagen von RL und *Quantencomputing*. Kapitel 3 beschreibt Verfahren und Konzepte, die für quantenbasiertes maschinelles Lernen verwendet werden, wonach diese in Kapitel 4 anhand einer Reproduktion verwandter Arbeiten experimentell untersucht und evaluiert werden. Kapitel 5 bietet einen Überblick über zusätzlich durchgeführte Experimente, in denen mögliche Erweiterungen der bestehenden Verfahren angewandt wurden. Zum Abschluss fasst Kapitel 6 die Arbeit zusammen und bietet einen Ausblick auf weitere Forschung.

2. Theoretische Grundlagen

In diesem Kapitel werden die zum Verständnis der Arbeit notwendigen theoretischen Grundlagen von RL und Quantencomputern beschrieben. Die Darstellung von RL ist überwiegend aus dem Werk „Reinforcement Learning - An Introduction“ [SB18] von Sutton und Barto übernommen.

2.1. Reinforcement Learning (RL)

RL ist eine Form des maschinellen Lernens, bei der durch Interaktion mit einer Umgebung ein Verhalten gelernt wird, das eine möglichst hohe Belohnung herbeiführt. Es ähnelt damit dem natürlichen Lernprozess von Lebewesen. Im Gegensatz zu anderen Formen des maschinellen Lernens, wie *Supervised Learning* und *Unsupervised Learning*, wird kein fester Datensatz zum Trainieren benötigt [SB18].

2.1.1. Markov-Entscheidungsprozess (MDP)

Der Lernprozess des RL kann durch einen *Markov-Entscheidungsprozess (MDP)* (engl. *Markov decision process*) formalisiert werden. Ein MDP umfasst zwei primäre Einheiten:

- Der *Agent* bezeichnet den Lernenden in dem Lernprozess. Er entscheidet welche Aktion aus einer Menge möglicher Aktionen \mathcal{A} durchgeführt wird.
- Die *Umgebung* bezeichnet die Einheit, mit der der Agent interagiert. Sie umfasst den Lernprozess außerhalb des Agenten. Die Umgebung enthält eine Menge an Repräsentationen ihrer möglichen Zustände (engl. *states*), \mathcal{S} . Außerdem verwaltet die Umgebung eine Menge an Belohnungen (engl. *rewards*) $\mathcal{R} \subset \mathbb{R}$.

Transitionen

Der Agent und die Umgebung interagieren in jedem diskreten Zeitschritt $t = 0, 1, 2, 3, \dots$ einer Sequenz. In jedem Zeitschritt t erhält der Agent eine Repräsentation des Umgebungszustandes $S_t \in \mathcal{S}$. Auf Basis von S_t wählt der Agent eine

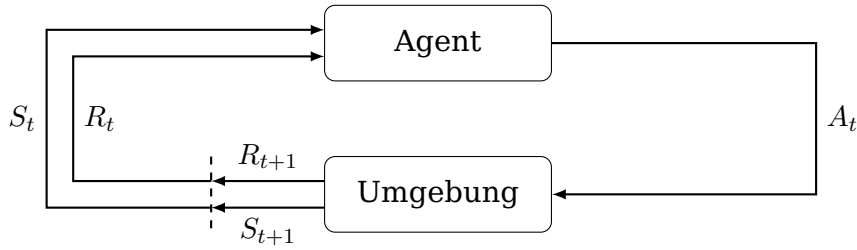


Abbildung 2.1.: Transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in einem MDP für den Zeitschritt t mit den Belohnungen R_t, R_{t+1} , den Zuständen S_t, S_{t+1} und der Aktion A_t . Quelle: [SB18], abgeändert

Aktion $A_t \in \mathcal{A}$. Die Umgebung bestimmt auf Grundlage von A_t eine numerische Belohnung $R_{t+1} \in \mathcal{R}$ und einen neuen Zustand $S_{t+1} \in \mathcal{S}$, die der Agent im nächsten Zeitschritt $t + 1$ erhält [SB18].

Eine solche *Transition* kann als Tupel $(S_t, A_t, R_{t+1}, S_{t+1})$ formuliert werden und ist in Abbildung 2.1 graphisch dargestellt. Die Wahrscheinlichkeit, dass die Umgebung von dem Zustand $s \in \mathcal{S}$ in den Zustand $s' \in \mathcal{S}$ übergeht, nachdem der Agent die Aktion $a \in \mathcal{A}$ ausführt, ist durch die folgende *Transitionsfunktion* gegeben [SB18]:

$$p(s'|s, a) = P\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} \quad (2.1)$$

Belohnungen

Der Erwartungswert der Belohnung r der bei dieser Transition erreicht wird, lässt sich durch eine *Belohnungsfunktion* bestimmen [SB18]:

$$r = r(s, a, s') = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] \quad (2.2)$$

Das Ziel des Agenten ist die langfristige Maximierung der kumulierten Belohnungen. Bei einer Sequenz an Belohnungen nach einem Zeitschritt t

$$R_{t+1}, R_{t+2}, R_{t+3}, \dots, R_n$$

wird versucht, den *erwarteten Rückgabewert* G_t zu maximieren, wobei T der finale Zeitschritt ist. Im einfachsten Fall ist G_t die Summe der Belohnungen:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T = \sum_{i=t+1}^T R_i \quad (2.3)$$

Bei dieser Formulierung werden zukünftige Belohnungen gleich gewertet, wie Aktuelle. RL-Anwendungen sind häufig so konzipiert, dass die Interaktion zwischen Umgebung und Agent in *Episoden* unterteilt werden kann. Darin ist $T <$

∞ und G_t konvergiert. In Fällen, in denen kontinuierliche Aufgaben gelöst werden sollen, ist der finale Zeitschritt $T = \infty$ und damit ein optimaler erwarteter Rückgabewert ebenfalls $G_t = \infty$.

Um diese Divergenz zu vermeiden, wird eine *Discount-Rate* $\gamma \in [0, 1]$ eingeführt, die angibt, wie stark der Agent zukünftige Belohnungen bei Maximierung des erwarteten Rückgabewertes beachten soll. Folgende Definition von G_t berücksichtigt die *Discount-Rate*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \quad (2.4)$$

R_t und damit auch G_t sind Zufallsvariablen [SB18].

Strategien

Das Verhalten des Agenten wird durch seine Strategie (*engl. policy*), $\pi(a|s)$, definiert, wobei $S_t = s$ und $A_t = a$ [SB18]. Die Strategie gibt die Wahrscheinlichkeit an, dass der Agent die Aktion a wählt, wenn sich die Umgebung in Zustand s befindet. RL-Methoden spezifizieren, wie sich die Strategie des Agenten durch Sammeln von Erfahrung ändert.

Die Strategie kann mit Hilfe der *Zustandswertfunktion*

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s], \forall s \in \mathcal{S} \quad (2.5)$$

evaluiert werden. $v_\pi(s)$ gibt den erwarteten Rückgabewert, ausgehend von Zustand s an, wenn die Strategie π verfolgt wird.

Eine ähnliche Möglichkeit zur Strategieevaluation bietet die *Aktionswertfunktion*:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad (2.6)$$

$q_\pi(s, a)$ liefert ebenso, wie Gleichung 2.5, den erwarteten Rückgabewert, wenn die Strategie π verfolgt wird, allerdings zusätzlich unter der Bedingung, dass Aktion a ausgeführt wurde. Die Ergebnisse dieser Funktion werden auch *Q-Werte* genannt.

Um eine RL-Aufgabe zu lösen, muss eine Strategie gefunden werden, die eine möglichst hohe Belohnung im Laufe der Zeit erzielt. Mit Hilfe der Wertfunktionen aus Gleichung 2.5 oder Gleichung 2.6 kann eine Ordnung möglicher Strategien hergestellt werden. Eine Strategie π ist besser, als eine Strategie π' , wenn $q_\pi(s, a) > q_{\pi'}(s, a)$ oder $v_\pi(s) > v_{\pi'}(s)$ für alle $s \in \mathcal{S}$ und alle $a \in \mathcal{A}$. Eine optimale Strategie wird mit π_* denotiert. Die *optimale Zustandswertfunktion*

$$v_*(s) = \max_{\pi} v_\pi(s), \forall s \in \mathcal{S} \quad (2.7)$$

und die *optimale Aktionswertfunktion*

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a), \forall s \in \mathcal{S} \wedge \forall a \in \mathcal{A} \quad (2.8)$$

sind für π_* maximal.

q_* lässt sich auch formulieren als:

$$\begin{aligned} q_*(s, a) &= \max_{\pi} \mathbb{E}_{\pi} [R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E} [R_{t+1} + \gamma v_*(s') | S_t = s, S_{t+1} = s', A_t = a] \end{aligned} \quad (2.9)$$

Hier wird deutlich, dass der Agent bei den Verfolgen einer optimalen Strategie den erwarteten Rückgabewert für Aktion a in Zustand s erhält und anschließend weiterhin optimal agiert. Sobald q_* bekannt ist, kann daraus π_* abgeleitet werden mit [SJD21]:

$$\pi_*(a|s) = \operatorname{argmax}_a q_*(s, a) \quad (2.10)$$

Das Ziel des RL-Lernprozesses ist es, eine optimale Strategie zu erreichen oder sich dieser anzunähern. Diese Arbeit konzentriert sich auf das *Q-Learning*-Verfahren.

2.1.2. Q-Learning

Q-Learning [Wat89] hat zum Ziel, eine Schätzung der *optimalen Aktionsfunktion* aus Gleichung 2.9 zu lernen. Es ist eine Form des *Temporal-Difference (TD)-Lernens* und verwendet Aspekte der *dynamischen Programmierung* [SB18].

Dynamische Programmierung

Eine wichtige Eigenschaft der Wertfunktionen ist die Rekursivität. Für die optimale Strategie π_* gilt in jedem Zustand s für jede Aktion a :

$$\begin{aligned} q_*(s, a) &= \max_{\pi} \mathbb{E}_{\pi} [R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E} [R_{t+1} + \gamma v_*(s') | S_t = s, S_{t+1} = s', A_t = a] \text{ (aus 2.9)} \\ &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(s', a') | S_t = s, S_{t+1} = s', A_t = a \right] \end{aligned} \quad (2.11)$$

Gleichung 2.11 ist eine *Bellman-Gleichung* für q_* . Sie stellt den Zusammenhang des Werts eines Zustands-Aktions-Paares mit dem seines Nachfolgers her. Ähnlich zu Gleichung 2.11 lässt sich auch für v_* eine Bellman-Gleichung formulieren [SB18]. Da *Q-Learning* allerdings hauptsächlich mit q_* arbeitet, werden im weiteren Verlauf der Arbeit die Formulierungen für v_* vernachlässigt.

In der dynamischen Programmierung werden Bellman-Gleichungen als Aktualisierungsregeln verwendet, um die gewünschte Wertfunktion zu verbessern [SB18].

In einer Sequenz an geschätzten Aktionswertfunktionen q_k (q_0, q_1, q_2, \dots) wird für jedes Zustandspaar (s, a) q_0 willkürlich initialisiert und anschließend jede weitere geschätzte *Aktionswertfunktion* q_{k+1} bestimmt durch:

$$\begin{aligned} q_{k+1}(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_k(s', a') \mid S_t = s, S_{t+1} = s', A_t = a \right] \\ &= \sum_{S_{t+1}} p(s' \mid s, a) \left(r(s, a, s') + \gamma \max_{a'} q_k(s', a') \mid S_t = s, A_t = a \right) \end{aligned} \quad (2.12)$$

Aus q_k lässt sich die Strategie π_k ableiten. π_k konvergiert gegen π_* für $k \rightarrow \infty$. Diese Art der Aktualisierung nennt sich *Q-Wert Iteration* [Mni+13].

TD-Lernverfahren

Ein Nachteil der reinen *Q-Wert Iteration*, oder der dynamischen Programmierung im Allgemeinen ist, dass ein vollständiges Modell des MDPs für alle möglichen Transitionen vorhanden sein muss. Dies ist für die meisten Umgebungen nicht praktikabel. Im Gegensatz dazu kann in TD-Lernalgorithmen ein unbekanntes Modell erforscht werden, indem Aktionen ausprobiert werden, was zu einer Abschätzung der Transitionen und der Wertfunktionen führt. Neue Schätzungen der Wertfunktion werden auf Grundlage vergangener Schätzungen aufgebaut [SB18].

Q-Learning ist eine Variante des TD-Verfahrens. Ursprünglich handelt es sich dabei um einen tabularen Lernalgorithmus. Das bedeutet, dass die *Q-Werte* für alle Zustands-Aktionspaare, die während des Lernprozesses auftreten, in einer Tabelle abgespeichert werden. Im ersten Schritt des Algorithmus wird die Tabelle willkürlich befüllt. Danach erforscht der Agent die Umgebung und aktualisiert jeden *Q-Wert* eines entdeckten Zustands-Aktions-Paares, folgendermaßen [SB18]:

$$q(S_t, A_t) \leftarrow (1 - \alpha) \underbrace{q(S_t, A_t)}_{\text{alter Wert}} + \alpha \underbrace{\left(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, A_{t+1}) \right)}_{\text{neuer Wert}} \quad (2.13)$$

Dabei ist $\alpha \in [0, 1]$ eine Schrittweite, die bestimmt, wie stark der alte Wert dem neuen angenähert werden soll. In [Mel01] wurde gezeigt, dass $q(s, a)$ mit diesem Verfahren gegen die optimalen *Q-Werte* $q_*(s, a)$ konvergiert, wenn alle Zustands-Aktions-Paare unendlich oft besucht werden.

2.1.3. Deep Q-Learning

In Anwendungen mit vielen möglichen Transitionen ist das Speichern des *Q-Wertes* für alle Zustände und Aktionen allerdings ineffizient hinsichtlich des Speicherbedarfs und nur schwer realisierbar. Deshalb wird im sogenannten

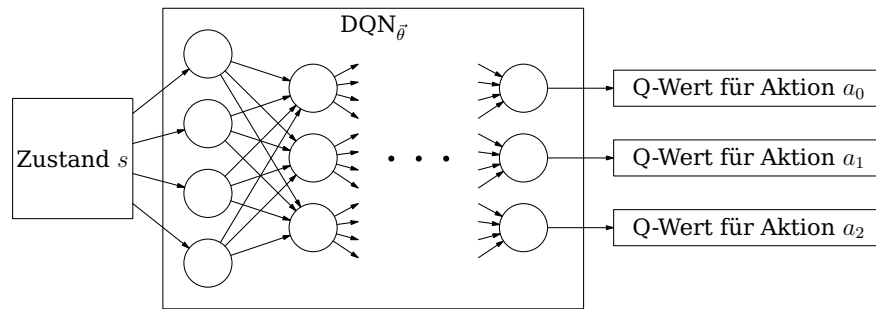


Abbildung 2.2.: Darstellung eines DQNs. Das mit $\vec{\theta}$ parametrisierte DQN besteht aus mehreren Schichten. Es erhält die Kodierung eines Zustandes s als Eingabe und gibt für jede mögliche Aktion a_i einen Q -Wert zurück.

Deep Q-Learning ein NN zur Approximation der *Aktionswertfunktion* verwendet. Dieses mehrschichtige, mit den Gewichten $\vec{\theta}$ parametrisierte Deep Q-Network (DQN), wird mit der Funktion $q(s, a; \theta)$ denotiert.

Das DQN ist eigentlich so strukturiert, dass es einen Zustand s als Eingabe erhält und einen Vektor an möglichen Q -Werten zurückgibt:

$$\vec{q} = q_{\vec{\theta}}(s) \quad (2.14)$$

Die Größe dieses Vektors ist die Anzahl der möglichen Aktionen. Damit das DQN die Aktionswertfunktion $q(s, a; \theta)$ berechnen kann, wird der Q -Wert an der Indexposition der Aktion a des Vektors \vec{q} als Rückgabewert bestimmt [HGS15]. In Abbildung 2.2 ist ein DQN visualisiert.

Epsilon-greedy Strategie

Diese Arbeit orientiert sich an dem RL-Algorithmus von Mnih et al. [Mni+13] und verwendet eine ϵ -greedy-Strategie. Hierbei wird mit einer Wahrscheinlichkeit $\epsilon \in [0, 1]$ eine zufällige Aktion und mit der Wahrscheinlichkeit $\epsilon - 1$ die Aktion gewählt, welche die größten Q -Werte im aktuellen Zustand liefert. Zufällige Aktionen sind vor allem zu Beginn des Lernprozesses sinnvoll, damit der Agent die Umgebung möglichst umfangreich erforschen kann. Zu Beginn sollte damit $\epsilon = 1$ sein und im Verlauf immer kleiner werden, bis zum Ende $\epsilon = 0$ beträgt und nur noch die möglichst beste Strategie verfolgt wird.

Experience replay

Um den Lernprozess zu beschleunigen, wird die Technik *experience replay* [Lin91] verwendet. Dabei werden Erfahrungen des Agenten bei Erforschung der Umgebung nicht verworfen, sondern in jedem Zeitschritt t als Tupel $e_t = (S_t, A_t, S_{t+1}, R_{t+1})$ zwischengespeichert. Während des Lernvorganges wird eine

kleine Menge (ein *Batch*) der gespeicherten Werte zufällig geladen und für die Aktualisierung der Gewichte des DQN verwendet. Dieses Aktualisieren wird im folgenden Absatz weiter ausgeführt.

Aktualisierung des DQNs

Im Gegensatz zu tabularen *Q-Learning* wird in *Deep Q-Learning* nicht der *Q-Wert* direkt aktualisiert, sondern das DQN, welches die *Aktionswertfunktion* abschätzt. Genauer gesagt werden die Gewichte $\vec{\theta}$ aktualisiert. Diese Aktualisierung erfolgt ähnlich wie beim *supervised learning* über das stochastische *gradient descent*-Verfahren [Mni+13].

Das stochastische *gradient descent*-Verfahren [Ama93] aktualisiert die Gewichte $\vec{\theta}$ eines NNs durch folgende Formel:

$$\theta_{i+1}^k = \theta_i^k - \alpha \frac{\partial}{\partial \theta_i^k} \mathbb{L}_i(\theta_i) \quad (2.15)$$

Dabei $\alpha \in [0, 1]$ eine Lernrate, θ_i^k ein Gewicht des DQNs zum Aktualisierungsschritt i und $\mathbb{L}_i(\theta_i)$ die Fehlerfunktion. $\frac{\partial}{\partial \theta_i^k} \mathbb{L}_i(\theta_i)$ ist ein sogenannter *Gradient*. Die Berechnung eines Gradienten erfolgt durch Differenzierung mit dem *Backpropagation*-Verfahren [Ama93]. Für den Fall des quantenbasierten, maschinellen Lernens wird dies in Abschnitt 3.5 näher beschrieben.

In jedem Aktualisierungsschritt i wird ein *Batch* an Erfahrungen \mathbb{B} gleichverteilt aus dem Zwischenspeicher geladen. Jeder Wert des *Batches* hat die Form $e = (s, a, s', r)$. Der Fehler wird beispielsweise über die *Mean Squared Loss*-Funktion [SJD21; Mni+15] berechnet:

$$\mathbb{L}_i(\vec{\theta}_i) = \frac{1}{|\mathbb{B}|} \sum_{e \in \mathbb{B}} \left(\underbrace{r + \gamma \max_{a'} q(s', a'; \vec{\theta}_i)}_{\text{Zielwert}} - \underbrace{q(s, a, \vec{\theta}_i)}_{\text{berechneter Aktionswert}} \right)^2 \quad (2.16)$$

Dabei sind $\vec{\theta}_i$ die Gewichte des DQNs zur Iteration i .

Ziel-DQN

In [VGS16] wurde gezeigt, dass der Lernvorgang in vielen Anwendungen stabiler abläuft, wenn für die Zielwertberechnung ein anderes DQN, als für die Berechnung des Aktionswertes, verwendet wird. Es wird somit auch in dieser Arbeit ein *Ziel-DQN* verwendet, welches eine Kopie des regulären *Strategie-DQNs* ist. Die Gewichte des *Ziel-DQNs* $\vec{\theta}$ werden allerdings lediglich alle $M \in \mathbb{N}$

Zeitschritte aktualisiert und den Gewichten des *Strategie-DQNs* gleichgesetzt. Die Berechnung des Fehlers ändert sich folgendermaßen:

$$\mathbb{L}_i(\vec{\theta}_i) = \frac{1}{|\mathbb{B}|} \sum_{e \in \mathbb{B}} \left(\underbrace{r + \gamma \max_{a'} q(s', a'; \vec{\theta}_i^-)}_{\text{Zielwert}} - \underbrace{q(s, a, \vec{\theta}_i)}_{\text{berechneter Aktionswert}} \right)^2 \quad (2.17)$$

Eine vollständige Beschreibung des in der Arbeit verwendeten *Deep Q-Learning* Algorithmus ist in Algorithmus 1 beschrieben, wobei die Variable N bestimmt, wie viele Schritte für den Lernprozess durchlaufen werden.

Algorithmus 1 Deep Q-Learning Algorithmus nach Mnih et al. [Mni+13]

```

initialisiere Zwischenspeicher  $\mathbb{D}$  mit Kapazität  $C$ 
initialisiere die Gewichte des Strategie-DQNs,  $\vec{\theta}$ 
initialisiere die Gewichte des Ziel-DQNs,  $\vec{\theta}^- \leftarrow \vec{\theta}$ 
 $i \leftarrow 0$ 
while  $i < N$  do
   $t \leftarrow 0$ 
  while  $S_t$  ist kein Endzustand do
    Mit der Wahrscheinlichkeit  $\epsilon$  wähle eine zufällige Aktion  $A_t$ 
    Ansonsten wähle  $A_t \leftarrow \operatorname{argmax}_A q(S_t, A; \vec{\theta})$ 
    Führe Aktion  $A_t$  aus und erhalte Belohnung  $R_{t+1}$  und Folgezustand  $S_{t+1}$ 
    Speichere Transition  $(S_t, A_t, S_{t+1}, R_{t+1})$  in  $\mathbb{D}$ 
    Lade einen Batch  $\mathbb{B}$  an Transitionen  $e_j = (S_j, A_j, S_{j+1}, R_{j+1})$  aus  $\mathbb{D}$ 
    for all  $e_j \in \mathbb{B}$  do
       $y_j \leftarrow \begin{cases} R_j & \text{falls } S_{j+1} \text{ Endzustand ist} \\ R_j + \gamma \max_A Q(S_{j+1}, A; \vec{\theta}^-) & \text{sonst} \end{cases}$ 
      Berechne Fehler  $\mathbb{L}^j(\vec{\theta}) \leftarrow (y_j - q(S_j, A_t; \vec{\theta}))^2$ 
    end for
    Berechne Durchschnitt des Fehlers  $\mathbb{L}(\vec{\theta})$  über alle  $\mathbb{L}^j(\vec{\theta})$ 
    Berechne die Gradienten über  $\mathbb{L}(\vec{\theta})$ 
    Aktualisiere  $\vec{\theta}$  mit den Gradienten analog zu Gleichung 2.15
    if  $i \bmod M = 0$  then
       $\vec{\theta}^- \leftarrow \vec{\theta}$ 
    end if
     $t \leftarrow t + 1, i \leftarrow i + 1$ 
  end while
end while

```

2.2. Quantencomputer und deren Berechnungen

Quantencomputing bezeichnet weitestgehend das Forschungsgebiet der Informationsverarbeitung mittels quantenmechanischer Systeme. Das fundamentale Konzept des Quantencomputings ist das *Quantenbit*, auch *Qubit* genannt [NC10].

2.2.1. Qubits und Superposition

In der klassischen Informationsverarbeitung hat ein Bit entweder den Zustand 0 oder 1. Der Zustand eines Qubits unterscheidet sich dahingehend, dass er mehr als zwei feste Werte annehmen kann. Ein Qubit kann die *Quantenzustände* $|0\rangle$, $|1\rangle$ oder eine Linearkombination dieser beiden annehmen:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.18)$$

Dabei sind $\alpha, \beta \in \mathbb{C}$. Bei diesen Zuständen handelt es sich um Vektoren in einem zweidimensionalen komplexen Vektorraum, wobei die beiden Zustände $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ und $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ eine Orthonormalbasis bilden [NC10].

Wenn sich beide Koeffizienten α und β der Linearkombination aus Gleichung 2.18 von 0 unterscheiden, wird der Zustand als **Superposition** bezeichnet.

Wird eine Superposition gemessen, sind die genauen Werte für α und β nicht eindeutig bestimmbar. Stattdessen wird mit der Wahrscheinlichkeit $|\alpha|^2$ der Wert 0 und mit der Wahrscheinlichkeit $|\beta|^2$ der Wert 1 gemessen. Es gilt $|\alpha|^2 + |\beta|^2 = 1$. Eine Messung verändert den Quantenzustand, da ein Messergebnis diesen eindeutig als $|0\rangle$ oder $|1\rangle$ festlegt [NC10].

Gleichung 2.18 ist eine quantenmechanische Wellenfunktion. Wie in [NC10] beschrieben, kann diese auch formuliert werden als:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle \quad (2.19)$$

Die Werte $\theta, \varphi \in \mathbb{R}$ können graphisch als ein Punkt auf der Oberfläche der Einheitskugel, auch *Bloch-Kugel* genannt, interpretiert werden. Abbildung 2.3 zeigt einen Quantenzustand $|\psi\rangle$ auf der *Bloch-Kugel*.

2.2.2. Verschränkung mehrerer Qubits

Ein Quantensystem besteht aus einer Folge mehrerer Qubits. Der Gesamtzustand eines Quantensystems kann ermittelt werden, indem die Quantenzustände der einzelnen Qubits durch das sogenannte *Tensorprodukt* \otimes verknüpft werden. Der Zustand eines Zwei-Qubit-Quantensystems mit $|\psi_0\rangle = \beta_0|0\rangle + \beta_1|1\rangle$

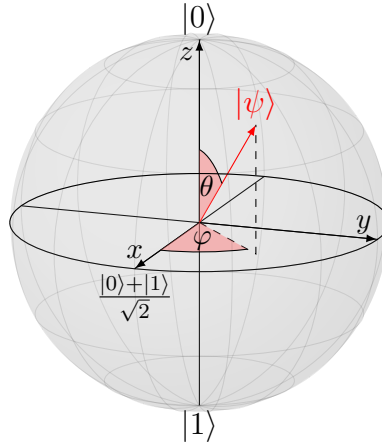


Abbildung 2.3.: Darstellung des Quantenzustands $|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle$ auf der Bloch-Kugel

und $|\psi_1\rangle = \gamma_0|0\rangle + \gamma_1|1\rangle$ wird durch die folgende Berechnung festgestellt, wobei $|\psi_0\rangle \otimes |\psi_1\rangle$ auch als $|\psi_0\psi_1\rangle$ formuliert werden kann und $\beta_i, \gamma_i \in \mathbb{C}$ [Hom18].

$$\begin{aligned}
 |\psi_0\psi_1\rangle &= |\psi_0\rangle \otimes |\psi_1\rangle \\
 &= (\beta_0|0\rangle + \beta_1|1\rangle) \otimes (\gamma_0|0\rangle + \gamma_1|1\rangle) \\
 &= \beta_0\gamma_0(|0\rangle \otimes |0\rangle) + \beta_0\gamma_1(|0\rangle \otimes |1\rangle) + \beta_1\gamma_0(|1\rangle \otimes |0\rangle) + \beta_1\gamma_1(|1\rangle \otimes |1\rangle) \\
 &= \beta_0\gamma_0|00\rangle + \beta_0\gamma_1|01\rangle + \beta_1\gamma_0|10\rangle + \beta_1\gamma_1|11\rangle
 \end{aligned} \tag{2.20}$$

Wird das Produkt der Koeffizienten $\beta_i\gamma_j$ durch einen einzelnen Koeffizienten $\alpha_{ij} \in \mathbb{C}$ ersetzt, ergibt sich folgende Formulierung:

$$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.21}$$

Der Zustand eines Zwei-Qubit-Quantensystems kann, analog zu Gleichung 2.18, als eine Linearkombination der folgenden vier Basiszustände formuliert werden:

$$\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\} = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\} \tag{2.22}$$

Wie bei der Messung des Zustand eines einzelnen Qubits, treten die Messergebnisse x ($= 00, 01, 10$ oder 11) mit der Wahrscheinlichkeit $|\alpha_x|^2$ auf. Dabei gilt die Normalisierungsbedingung $\sum_{x \in \{0,1\}^2} |\alpha_x|^2 = 1$.¹ Die Koeffizienten α_x werden auch *Amplituden* genannt [NC10].

Ein besonderer Zustand zweier Qubits ist der *Bell-Zustand*:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \sqrt{\frac{1}{2}}|00\rangle + 0|01\rangle + 0|10\rangle + \sqrt{\frac{1}{2}}|11\rangle \tag{2.23}$$

¹Die Schreibweise $\{0,1\}^n$ bezeichnet sämtliche mögliche Kombinationen aus 0 und 1 mit der Länge n .

Bei Untersuchung des Quantensystems im Bell-Zustand wird jeweils mit der Wahrscheinlichkeit $\frac{1}{2}$ das Ergebnis 00 oder 11 gemessen. Entscheidend ist, dass ein Messergebnis von 0 an dem ersten Qubit ein Messergebnis von 0 an dem zweiten Qubit nach sich zieht und vice versa, auch wenn die Messungen der beiden Qubits unter unterschiedlichen Bedingungen, wie beispielsweise zu verschiedenen Zeitpunkten, vorgenommen werden. Das Gleiche gilt auch für das Messen einer 1. Die Messergebnisse der beiden Qubits korrelieren. Dieses Phänomen ist quantenspezifisch und wird **Verschränkung** genannt [NC10].

Im allgemeinen Fall kann ein Quantensystem, das aus n Qubits besteht, formuliert werden mit $|\psi_0\psi_1 \dots \psi_{n-1}\rangle$. Der Zustand dieses Registers kann als Superposition von 2^n Basisvektoren mit den zugehörigen Amplituden spezifiziert werden [NC10]. Der Informationsgehalt bei n Qubits skaliert somit mit $\mathcal{O}(2^n)$ und nicht, wie im klassischen Fall, mit $\mathcal{O}(n)$ [LS20].

2.2.3. Quantenschaltkreise

Klassische Computer sind aus elektrischen Schaltkreisen aufgebaut, welche aus Leitungen und logischen Gattern bestehen, um Informationen zu übertragen und abzuändern. Analog dazu, verwenden Quantencomputer dafür Quantenschaltkreise, welche Quantenleitungen und Quantengatter enthalten [NC10]. Ein Quantenschaltkreis stellt das Berechnungsmodell eines Quantencomputers, womit Quanten-Algorithmen abgebildet werden können [Hom18].

Ein Quantenschaltkreis wird von links nach rechts gelesen. Jede Linie repräsentiert eine Quantenleitung, wobei diese nicht physikalisch sein muss, sondern auch beispielsweise einen zeitlichen Unterschied repräsentieren kann [NC10]. Dabei entspricht jede waagrechte Quantenleitung einem Qubit. Quantenoperationen werden durch Gatter repräsentiert, die als Symbole auf den Leitungen der Qubits dargestellt sind [Hom18]. Ein exemplarischer Quantenschaltkreis, welcher auch in dieser Arbeit verwendet wird, ist in Kapitel 4 in Abbildung 4.2 dargestellt. Per Konvention wird angenommen, dass der Eingabezustand für jedes Qubit in einem Quantenschaltkreis $|0\rangle$ ist [NC10].

2.2.4. Quantengatter

Mit einem Quantengatter kann der Zustand eines Quantensystems verändert werden.

Das NOT-Gatter

Ein Beispiel für ein solches Quantengatter ist das *NOT*-Gatter, welches den Zustand eines einzelnen Qubits invertiert. Das *NOT*-Gatter ändert beispielsweise

den Quantenzustand

$$\alpha |0\rangle + \beta |1\rangle \quad (2.24)$$

zu

$$\beta |0\rangle + \alpha |1\rangle. \quad (2.25)$$

Quantengatter können durch unitäre Matrizen repräsentiert und mit Matrixmultiplikation auf die Vektordarstellung eines Zustandes angewandt werden. Eine Matrix U gilt als unitär, wenn die Multiplikation mit seiner adjungierten Matrix U^\dagger die Einheitsmatrix I liefert, im Ergebnis damit $U^\dagger U = I$ gilt. Eine adjungierte Matrix U^\dagger ist die transponierte, komplex konjugierte Matrix von U [NC10]. Die Matrixdarstellung des *NOT*-Gatters ist:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.26)$$

Wird die Matrix auf den Zustand aus Gleichung 2.24 multipliziert, ergibt sich folgende Berechnung mit dem resultierenden Zustand aus Gleichung 2.26 als Ergebnis [NC10]:

$$\begin{aligned} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} (\alpha |0\rangle + \beta |1\rangle) &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \left(\alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \\ &= \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = \beta \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \beta |0\rangle + \alpha |1\rangle \end{aligned} \quad (2.27)$$

Das CNOT-Gatter

Quantengatter lassen sich auch für Folgen mehrerer Qubits definieren. In einem Quantensystem zweier Qubits kann beispielsweise das *kontrollierte NOT*-Gatter, auch *CNOT*-Gatter genannt, verwendet werden.

Das *CNOT*-Gatter hat zwei Eingabe-Qubits. Eines der Qubits ist das *Kontroll-Qubit*, das Andere das *Ziel-Qubit*. Wenn das Kontroll-Qubit 0 ist, ändert sich das Ziel-Qubit nicht. Ist das Kontroll-Qubit hingegen 1, wird das Ziel-Qubit invertiert. Das *CNOT*-Gatter führt also folgende Transformationen auf den Standardbasiszuständen eines Zwei-Qubit-Systems durch:

$$\begin{aligned} |00\rangle &\rightarrow |00\rangle & |10\rangle &\rightarrow |11\rangle \\ |01\rangle &\rightarrow |01\rangle & |11\rangle &\rightarrow |10\rangle \end{aligned} \quad (2.28)$$

Diese Operation entspricht auch der Anwendung des *exklusiven Oder*-Gatters *XOR* \oplus auf das zweite Qubit: $|x\rangle \otimes |y\rangle \rightarrow |x\rangle \otimes |x \oplus y\rangle$. Das Schaltsymbol des *CNOT* Gatters für einen Quantenschaltkreis ist in Abbildung 2.4 dargestellt [NC10; Hom18].

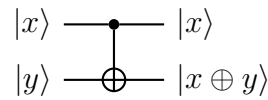


Abbildung 2.4.: Wirkung des *CNOT*-Gatters auf zwei Qubits mit den Zuständen $|x\rangle$ und $|y\rangle$

Quantengatter für mehrere Qubits lassen sich, genau wie Quantengatter für einzelne Qubits, als unitäre Matrizen repräsentieren. Die Matrixdarstellung des *CNOT*-Gatters ist:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.29)$$

Im Folgenden wird ein Überblick über weitere, für diese Arbeit relevante Gatter gegeben. In Anhang A ist eine tabellarische Zusammenfassung der beschriebenen Gatter zu finden.

Pauli-Gatter

Die Operation des *NOT*-Gatters aus Gleichung 2.27 ist die *X*-Matrix der drei *Pauli-Matrizen* [NC10]:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2.30)$$

Auch die beiden anderen Pauli-Matrizen, *Y* und *Z* sind Quantengatter [NC10; Hom18]. Die Wirkung der Pauli-Gatter auf einen Zustand $\alpha|0\rangle + \beta|1\rangle$ wird in Abbildung 2.5 beschrieben. Graphisch lässt sich eine Pauli-Operation in der Bloch-Kugel als Rotation um die entsprechende Achse mit dem Winkel π darstellen [LS20]. Dies ist in Abbildung 2.6 visualisiert.

$$\begin{aligned} \alpha|0\rangle + \beta|1\rangle &\xrightarrow{\mathbf{X}} \beta|0\rangle + \alpha|1\rangle \\ \alpha|0\rangle + \beta|1\rangle &\xrightarrow{\mathbf{Y}} -i\beta|0\rangle + i\alpha|1\rangle \\ \alpha|0\rangle + \beta|1\rangle &\xrightarrow{\mathbf{Z}} \alpha|0\rangle - \beta|1\rangle \end{aligned}$$

Abbildung 2.5.: Wirkung der Pauli-Gatter auf einen Quantenzustand $\alpha|0\rangle + \beta|1\rangle$ [Bab+18]

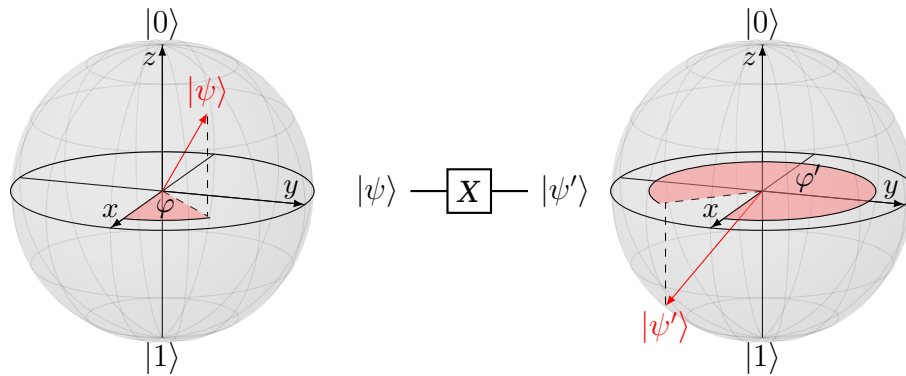


Abbildung 2.6.: Wirkung des Pauli-X-Gatters auf einen Quantenzustand $|\psi\rangle$ in der Bloch-Kugel. Analog werden das Y-Gatter und das Z-Gatter als Spiegelung an der y - oder z -Achse aufgefasst.

Kontrollierte Gatter

Die Idee des *CNOT*-Gatters lässt sich verallgemeinern. So kann jedes mögliche Quantengatter für ein einzelnes Qubit auch durch ein Kontroll-Qubit gesteuert werden. Wenn das Kontroll-Qubit 1 ist, wird das Gatter auf das Ziel-Qubits angewandt, hingegen nicht, wenn es 0 ist. Diese Operation wird durch folgende Matrix beschrieben:

$$C_U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & & U \\ 0 & 0 & & \end{pmatrix} \quad (2.31)$$

Dabei ist U eine beliebige unitäre 2×2 Matrix. Kontrollierte Gatter, oder allgemein Gatter über mehrere Qubits, können nicht verschränkte Zustände verschränken [Hom18].

Neben dem *CNOT*-Gatter wird auch das *kontrollierte Pauli-Z-Gatter* (*CZ*) in dieser Arbeit verwendet. Das *CZ*-Gatter wird durch folgende Matrix beschrieben:

$$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad (2.32)$$

Dieses Gatter führt die Pauli-Z-Operation auf dem Ziel Qubit aus, wenn das Kontroll-Qubit den Wert 1 hat. Es kann auch so beschrieben werden, dass es

die Amplitude des Basiszustands $|11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ negiert [GC11]. Wie in [GC11]

gezeigt wird, ist bei einem *CZ*-Gatter das Kontroll-Qubit und das Ziel-Qubit austauschbar.

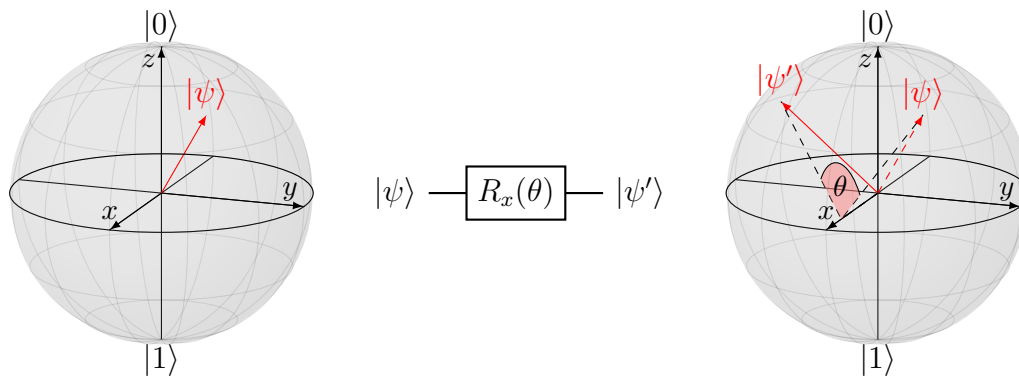


Abbildung 2.7.: Wirkung des mit θ parametrisierten X-Rotationsgatters auf einen Quantenzustand in der Bloch-Kugel

Pauli-Rotationsgatter

Die Pauli-Rotationsgatter $R_x(\theta)$, $R_y(\theta)$, $R_z(\theta)$ sind spezialisierte Versionen der Pauli-Gatter. Mit ihnen kann ein Quantenzustand in der Bloch-Kugel um einen spezifizierten Winkel θ rotiert werden [NC10]. Ein Beispiel, wie dadurch ein Zustand transformiert wird, ist in Abbildung 2.7 dargestellt. Für die Matrixrepräsentationen wird auf die Liste aller Gatter in ?? verwiesen. Rotationsgatter sind elementare Operationen in parametrisierten Quantenschaltkreisen (PQCs), welche in Abschnitt 3.1 genauer beschrieben werden.

2.2.5. Messungen in der Standardbasis

Die Messung des Zustandes eines Quantensystems führt den Quantenzustand in einen eindeutigen, klassischen Wert über. In dieser Arbeit wird lediglich die Messung in der Standardbasis $\{|0\rangle, |1\rangle\}$ betrachtet. Ein Messen ist jedoch auch in allen anderen Orthonormalbasen möglich [NC10].

Bei Messung eines Quantenzustandes $\alpha|0\rangle + \beta|1\rangle$ in der Standardbasis wird, wie bereits in Unterabschnitt 2.2.1 beschrieben, ein eindeutiger Wert basierend auf den Wahrscheinlichkeiten $|\alpha|^2$ und $|\beta|^2$ gemessen.

Der Erwartungswert einer Messung in der Standardbasis auf einem Qubit lässt sich durch folgende Formel [Asf+20] berechnen:

$$\mathbb{E}[\alpha|0\rangle + \beta|1\rangle] = |\alpha|^2 - |\beta|^2 \quad (2.33)$$

Wird der Wert 0 gemessen, ist der Erwartungswert somit 1. Bei Messen einer 1 ist der Erwartungswert -1 . Um einen aussagekräftigen Erwartungswert ermitteln zu können, werden Quantenschaltkreise mehrfach durchlaufen und ein Durchschnitt bestimmt.

Eine Erwartungswertberechnung ist auch über mehrere Qubits möglich. So wird diese in einem Zwei-Qubit-System folgendermaßen bestimmt [Asf+20]:

$$\mathbb{E}[\alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle] = |\alpha_{00}|^2 - |\alpha_{01}|^2 - |\alpha_{10}|^2 + |\alpha_{11}|^2 \quad (2.34)$$

3. Maschinelles Lernen mit parametrisierten Quantenschaltkreisen (PQCs)

Folgendes Kapitel bietet dem Leser Hintergrundwissen über die Konzepte und Verfahren, die für quantenbasiertes RL verwendet werden. Nachfolgend werden diese Verfahren experimentell untersucht.

3.1. Hybrider Ansatz für Quanten-RL

Der Ablauf der in dieser Arbeit verwendeten Quanten-RL-Algorithmen orientiert sich an den Arbeiten [Mit+18; Ben+19; LW18; Sko+21; Zhu+19; Sch+20], in denen quantenbasierte Lernverfahren für *supervised* oder *unsupervised learning* vorgestellt werden, sowie an den Arbeiten [LS20; LS21; SJD21; Che+20], die ebenfalls Quanten-RL behandeln. Dabei werden die Algorithmen aus [LS20] und [SJD21] in Kapitel 4 genauer untersucht. PQCs bilden die Basis der in den verwandten Arbeiten verwendeten Verfahren.

Ein PQC ist ein Quantenschaltkreis, welcher Gatter enthält, die durch Parameter angepasst werden können. Ähnlich, wie mit einem klassischen NN, kann mit einem PQC eine Funktion approximiert werden [Ben+19; Che+20; Mit+18]. Die Parameter werden durch Optimierungsverfahren, wie beispielsweise *gradient descent*, iterativ aktualisiert [Che+20].

Quanten-RL verfolgt einen hybriden Ansatz aus klassischen Berechnungen und Quantencomputing [Mit+18]. Algorithmus 1 wird mit wenigen Anpassungen auch in Quanten-RL verwendet. Die hauptsächliche Änderung zum klassischen Algorithmus ist das Ersetzen des in Unterabschnitt 2.1.3 beschriebenen DQN durch einen PQC.

Die einzelnen Schritte des RL-Lernprozesses werden auf einen klassischen Computer und einem Quantencomputer aufgeteilt [Che+20]. Eine Übersicht der Aufteilung ist in Abbildung 3.1 visualisiert.

Hyperparameter, die Umgebung und der Zwischenspeicher werden weiterhin klassisch verwaltet. Auch die Wahl der Aktion mit der ϵ -greedy-Strategie, die Fehlerberechnung und das Optimierungsverfahren der Gewichte werden klassisch durchgeführt. Dagegen wird die Aktionswertfunktion $Q(s, a; \vec{\theta})$ mit $s \in$

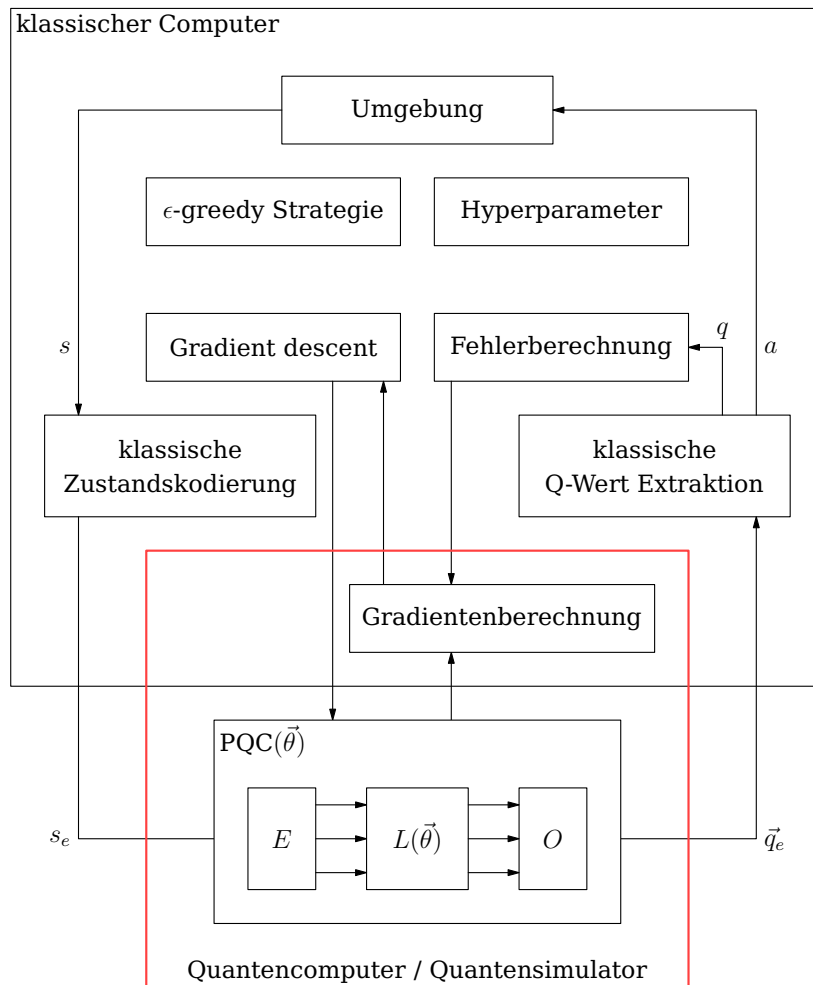


Abbildung 3.1.: Übersicht über die beteiligten Komponenten bei Quanten-RL. Der klassische Computer bearbeitet weiterhin einen großen Teil der Schritte von Algorithmus 1. Der Quantencomputer erhält einen kodierten Quantenzustand als Eingabe und berechnet mit dem PQC die Q-Werte, welche klassisch dekodiert werden müssen. Mit dem *gradient descent* Verfahren werden die Parameter $\vec{\theta}$ des PQC's aktualisiert. Dafür werden die Gradienten häufig durch eine hybride Berechnung des Quantencomputers und des klassischen Computers ermittelt.

$S, a \in \mathcal{A}$, durch einen PQC auf einem Quantencomputer oder Quantensimulator berechnet [Che+20], wobei die Struktur des PQCs wird im nächsten Abschnitt beschrieben wird. Jeder Aufruf von $Q(s, a; \vec{\theta})$ in Algorithmus 1 wird in Quanten-RL durch die folgenden Schritte ersetzt:

1. Die Repräsentation des Eingabezustandes s wird von einem klassischen Datum in einen Quantenzustand konvertiert. Möglichkeiten zur Kodierung von klassischen Daten werden in Abschnitt 3.3 vorgestellt.
2. Der mit $\vec{\theta}$ parametrisierte PQC wird durchlaufen.
3. Der Zustand des PQCs wird gemessen und in einen Vektor von möglichen Q-Werten \vec{q} konvertiert. Dieser wird, wie in Unterabschnitt 2.1.3 dargestellt, weiter verwendet. Das Extrahieren von Quantenzuständen wird in Abschnitt 3.4 näher beschrieben.

Für die Gradientenberechnung in quantenbasierten, maschinellen Lernen wird in dieser Arbeit das *parameter shift*-Verfahren [Sch+19; Mit+18] verwendet. Dieses wird klassisch durchgeführt, enthält allerdings Berechnungen, welche von einem PQC durchgeführt werden. Daher ist die Gradientenberechnung in Abbildung 3.1 als hybrid markiert. In Abschnitt 3.5 wird dies näher erläutert.

3.2. Struktur des PQCs für maschinelles Lernen

Der PQC für maschinelles Lernen wird in verwandten Arbeiten [Zhu+19; Che+20; LS20; LS21; Sko+21; SJD21; Sch+20] in Form von Schichten aufgebaut. Dies hat den Vorteil, dass die Tiefe des Schaltkreises einfach skalierbar ist [Che+20]. Der Begriff „Schicht“ wird dabei lediglich zu Visualisierungszwecken verwendet. Die mathematischen Operationen einer PQC-Schicht unterscheiden sich von denen einer Schicht eines klassischen NN [LS20].

Alle Schichten des PQC verfolgen einen hardwareeffizienten Ansatz, wie es in [Kan+17] beschrieben wird. Sie bestehen aus Rotationsgattern auf einzelnen Qubits und Gattern, welche jeweils zwei Qubits miteinander verschränken, wie beispielsweise *CNOT*- oder *CZ*-Gatter. Die erste Schicht des PQCs ist für die Kodierung der klassischen Eingabedaten zuständig (siehe Abschnitt 3.3). Die anderen Schichten stellen den mit $\vec{\theta}$ parametrisierten Teil des Schaltkreises dar. Jede parametrisierte Schicht L_l hat hierbei zwei Bestandteile:

- Eine Menge an Rotationsgattern U_l , die mit den Rotationswinkeln $\vec{\theta}_l$ parametrisiert werden
- Eine Menge an Gattern V_l , welche die Qubits des Schaltkreises miteinander verschränken

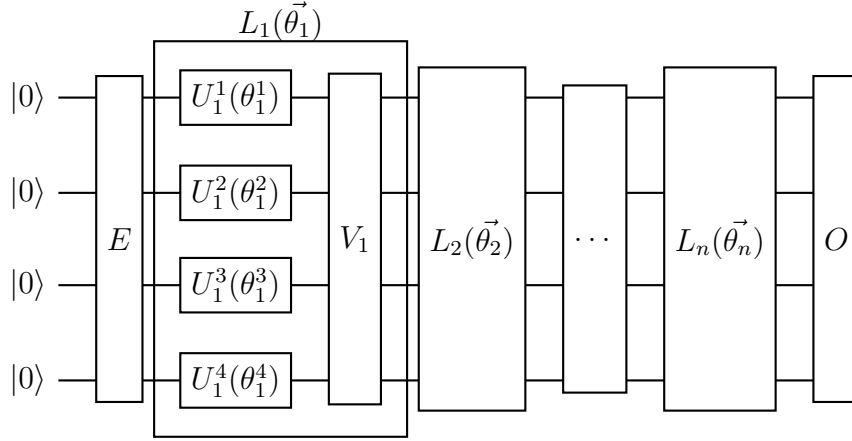


Abbildung 3.2.: Struktureller Aufbau eines mit $\vec{\theta}$ parametrisierten PQC. Dabei bezeichnet E die Sammlung an Gattern, welche klassische Eingaben kodieren, $L(\vec{\theta}) = \{L_1(\vec{\theta}_1), L_2(\vec{\theta}_2), \dots, L_n(\vec{\theta}_n)\}$ die parametrisierten Schichten und O eine Messoperation. Im Fall der in Abschnitt 3.4 beschriebenen Quanten-Bündelungsoperation kann O auch aus mehreren Gattern bestehen. Eine Schicht L_l enthält Rotationsgatter U_l auf jedem Qubit und eine Sammlung an kontrollierten Gattern V_l zur Verschränkung.

Lockwood und Si [LS20; LS21] definieren eine weitere Schicht am Ende des Schaltkreises, welche speziell den Quantenzustand für den Rückgabewert anpasst. Dies wird in Abschnitt 3.4 weiter ausgeführt.

Zusammengefasst besteht ein PQC aus einer Eingabeschicht, mehreren parametrisierten Schichten $L(\vec{\theta})$ und einer optionalen Ausgabeschicht O . Eine Übersicht dieser Bestandteile ist in Abbildung 3.2 dargestellt. Beispiele für PQCs werden in Kapitel 4 vorgestellt.

3.3. Kodierung klassischer Daten

Ein Quantenschaltkreis kann lediglich Quantenzustände interpretieren. In der Arbeit wird angenommen, dass der initiale Zustand eines Quantenschaltkreises $\otimes_{i=1}^n |0\rangle$ ist. Dieser Zustand wird in einem PQC angepasst, sodass er die Informationen der Eingabedaten enthält. Bei RL ist ein Eingabedatum die Repräsentation eines Umgebungszustandes $s \in \mathcal{S}$.

Die erste Schicht des PQC transformiert den initialen Zustand $\otimes_{i=1}^n |0\rangle$ zu dem kodierten Zustand $|\psi\rangle$, welcher die Informationen aus s enthält. s wird dabei klassisch in eine Repräsentation s_e umgewandelt, die von Quantengattern interpretiert werden kann. Anschließend wird eine Sammlung an mit s_e parametrisierten

trisierten Gattern $E(s_e)$ auf den Eingabezustand angewandt:

$$|\psi\rangle = E(s_e)(\otimes_{i=1}^n |0\rangle) = E(s_e) \left| \underbrace{00 \dots 0}_{n\text{-Mal}} \right\rangle \quad (3.1)$$

Die Eingabeschicht $E(s_e)$ besteht in verwandten Arbeiten ([SJD21; LS20; LS21; Che+20]) aus einzelnen oder mehreren Rotationsgattern auf jedem Qubit. Genaue Architekturen der Eingabeschicht werden in Kapitel 4 vorgestellt.

Das Verfahren, welches zur klassischen Kodierung eines originalen Eingabedatum s verwendet wird, bestimmt, um welchen Winkel rotiert wird und wie viele Rotationsgatter und damit auch wie viele Qubits in dem PQC benötigt werden [LS21].

3.3.1. Binäre Kodierung

Auf klassischen Computern werden Daten binär dargestellt. Im Folgenden werden Verfahren beschrieben, mit welchen eine binäre Kodierung auch auf Quantencomputern durchgeführt werden kann.

Diskrete Kodierung

In RL-Umgebungen mit einer endlichen Menge an diskreten Zuständen kann jeder Zustand mit einer Zahl eindeutig spezifiziert und durch eine klassische Binärzahl dargestellt werden. Jede Stelle b_i der Binärzahl $b_n b_{n-1} \dots b_1$ eines Eingabezustandes kann als Quantenzustand eines einzelnen Qubits interpretiert werden, indem für jedes Qubit der initiale Zustand $|0\rangle$ erhalten bleibt, wenn $b_i = 0$ und auf $|1\rangle$ geändert wird, wenn $b_i = 1$ [Che+20]. Dies kann erreicht werden, indem die Rotationsgatter $R_x(b_i * \pi)$ und $R_z(b_i * \pi)$ auf jedem Qubit angewandt werden [Che+20]. Mit diesem Verfahren wird für jede Binärstelle ein Qubit verwendet.

Bitweise Kodierung

In Umgebungen mit nicht endlicher Zustandsmenge kann das diskrete Kodierungsverfahren nicht angewandt werden. Hier könnte die Repräsentation eines Eingabezustand direkt binär kodiert werden, da klassische Daten durch mehrere Bits repräsentiert sind. Die Anzahl der Bits hängt von dem Datentyp des Eingabedatums ab. So besteht beispielsweise ein *float*-Wert, also eine Gleitkommazahl mit einfacher Genauigkeit, aus 32 Bits [LS21].

Es wird die gleiche Anzahl an Qubits zur Kodierung benötigt, wie das Eingabedatum Bits hat. Im Fall der simplen *Cartpole*-Umgebung, welche in Abschnitt 4.1

genauer beschrieben wird, besteht ein Eingabezustand aus vier *float*-Werten. Somit werden für die binäre Kodierung $4 * 32 = 128$ Qubits benötigt, was die Anzahl von maximal 50 bis 100 Qubits auf aktuellen NISQ-Geräten übersteigt [Pre18].

Binäre Kodierung ist somit ungeeignet für RL-Umgebungen mit nicht endlicher Zustandsmenge. Daher werden in dieser Arbeit andere Kodierungsverfahren verwendet. In [LS20] und [SJD21] werden Verfahren vorgestellt, welche für jede Komponente des Eingabezustandes ein Qubit verwenden. Dies ist effizienter hinsichtlich der Anzahl der Qubits [LS20].

3.3.2. Komponentenweise Kodierung

Im Folgenden werden die in der Arbeit verwendeten Kodierungsverfahren beschrieben. Ein Vergleich der Verfahren hinsichtlich der Leistung eines PQC wird in Kapitel 4 aufgestellt.

Kontinuierliche Kodierung

Skolik et al. [SJD21] stellen ein Kodierungsverfahren vor, mit dem jede Komponente x des Eingabezustandes mit $\arctan(x)$ in einen Bereich $[-\frac{\pi}{2}, \frac{\pi}{2}]$ skaliert wird. Der skalierte Wert kann als Winkel für ein oder mehrere Rotationsgatter auf einem Qubit in der Eingabeschicht verwendet werden.

Skalierte Kodierung

Die skalierte Kodierung [LS20] kann auf Komponenten eines Eingabezustandes angewandt werden, welche in einem definierten Bereich liegen. Dabei wird ein Wert auf den Bereich $[0, 2\pi]$ skaliert. Dieser skalierte Wert wird, wie in der kontinuierlichen Kodierung, als Rotationswinkel verwendet.

Gerichtete Kodierung

Lockwood und Si [LS20] definieren ein Verfahren, welches die Rotationsgatter der Eingabeschicht auf einem Qubit um π rotiert, wenn die Eingabekomponente größer als 0 ist. Ansonsten wird mit 0 rotiert. Mit diesem gerichteten Verfahren ist ein Informationsverlust verbunden, da ein unendlicher Wertebereich auf zwei Werte reduziert wird. Dennoch hat sich dieses Verfahren in Kombination mit der skalierten Kodierung in [LS20] experimentell als praktikabel erwiesen.

3.4. Extrahieren von Quantendaten

Die Rückgabe eines PQC wird durch Messungen bestimmt. Der Erwartungswert eines einzelnen Qubits kann durch mehrmaliges Durchlaufen und Messen eines Quantenschaltkreises analog zu Gleichung 2.33 berechnet werden. Die Anzahl der Rückgabewerte in einem PQC ist somit durch die Menge der Qubits festgelegt und nicht wie in einem NN manuell regulierbar [SJD21].

In einer Q-Learning-Anwendung ist dieser Unterschied entscheidend, da die Rückgabe eines DQNs ein Vektor von *Q-Werten* für jede mögliche Aktion ist [SJD21]. Die Anzahl der Aktionen und somit auch die Dimension des Rückgabektors sind durch die Umgebung festgelegt. Die gemessenen Ergebnisse eines PQC müssen somit je nach Umgebung ausgewertet und der Anzahl der möglichen Aktionen angepasst werden.

Erwartungswert über mehrere Qubits

Ist die Anzahl der möglichen Aktionen kleiner als die Anzahl der Qubits, werden die Informationen mehrerer Qubits zusammengefasst. Dies kann erreicht werden, indem der Erwartungswert über die Zustände mehrerer Qubits berechnet wird. So kann beispielsweise mit Gleichung 2.34 der Erwartungswert über zwei Qubits berechnet werden [Asf+20].

Lockwood und Si [LS20] stellen zwei andere Möglichkeiten vor, die Dimension der Rückgabewerte ohne Informationsverlust zu reduzieren.

Hybrides Modell

In dem ersten Verfahren werden die Erwartungswerte über die Qubits einzeln berechnet und an die Schicht eines NNs weitergegeben. Diese Schicht enthält Gewichte, welche im Verlauf des Lernprozesses angepasst werden. Die Dimension der Rückgabewerte der Schicht wird nach der Anzahl der möglichen Aktionen definiert. Die Autoren bezeichnen dieses Modell als *hybrid*, wobei beachtet werden sollte, dass Quanten-RL generell ein hybrides Verfahren aus klassischen und quantenbasierten Operationen ist (siehe Abschnitt 3.1).

Quanten-Bündelungsoperation

In dem zweiten Verfahren werden Informationen zusammengefasst, indem die Quantenzustände zweier Qubits zu einem gebündelt werden. Die Operation, welche an den beiden Qubits vorgenommen wird, ist durch eine Sammlung von Quantengattern definiert, welche in Abbildung 3.3 dargestellt wird. Dabei werden die Zustände der beiden Qubits durch Rotationsgatter R_x, R_y, R_z verändert und durch ein *CNOT*-Gatter verschränkt. Auf dem *Ziel-Qubit* werden inverse

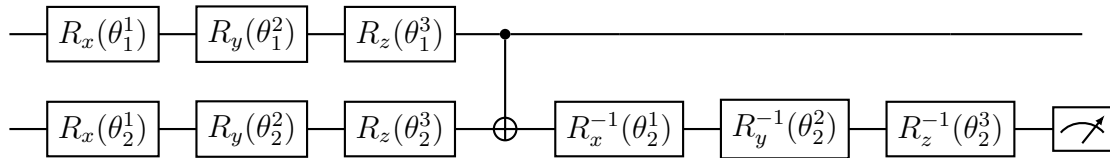


Abbildung 3.3.: Darstellung der Quanten-Bündelungsoperation nach Lockwood und Si

Rotationsgatter $R_x^{-1}, R_y^{-1}, R_z^{-1}$ angewendet. Der Erwartungswert wird für das Ziel-Qubit berechnet. Diese Operation kann so oft wie nötig auf jeweils zwei Qubits durchgeführt werden.

Skalierung der Ausgabewerte

Die beschriebenen Verfahren zum Extrahieren der Quantendaten werden in Kapitel 4 hinsichtlich der Leistung eines PQC's evaluiert und verglichen.

Die berechneten Erwartungswerte liegen in $[-1, 1]$. Ein *Q-Wert* ist, wie in Unterabschnitt 2.1.1 beschrieben, ein Abschätzung der zukünftigen Belohnung in einem MDP. In Umgebungen, in denen Belohnungen einen hohen Wert annehmen können, reicht der Bereich $[-1, 1]$ in vielen Fällen nicht aus, um *Q-Werte* ausreichend abzubilden [SJD21]. Daher werden in dieser Arbeit die gemessenen Erwartungswerte mit einem Koeffizienten versehen, welcher in Verlauf des Lernprozesses angepasst werden kann [SJD21].

3.5. Gradientenberechnung

Gradienten werden im *gradient descent*-Verfahren durch partielle Ableitungen eines Gewichts θ_i nach der Fehlerfunktion $\mathbb{L}(\vec{\theta})$ ermittelt. Im klassischen Fall werden Gradienten oft durch automatische Differenzierung unter Anwendung der Kettenregel berechnet. Dabei werden partielle Ableitungen bereits während der Berechnung der Ausgabe durch Zwischenergebnisse ermittelt. Im Fall von Quantencomputern ist dies problematisch, da eine Messung innerhalb eines Quantenschaltkreises dessen gesamte Berechnung beeinflusst [Sch+19].

Stattdessen wird eine partielle Ableitung innerhalb des PQC's in dieser Arbeit mit der *parameter shift*-Regel [Sch+19; Mit+18] ermittelt. Wird der PQC als Funktion $f(\vec{\theta})$ aufgefasst, welche Erwartungswerte der Messungen bestimmt, so lässt sich die partielle Ableitung $\frac{\partial}{\partial \theta_i} f(\vec{\theta})$ durch die folgende Formel [Sko+21; Cro19] bestimmen:

$$\frac{\partial}{\partial \theta_i} f(\vec{\theta}) = r \left(f(\vec{\theta} + s\hat{\theta}_i) - f(\vec{\theta} - s\hat{\theta}_i) \right) \quad (3.2)$$

Dabei ist $\hat{\theta}_i$ ein Einheitsvektor in die Richtung der Komponente i von $\vec{\theta}$, $s = \frac{\pi}{4r}$ und r eine Konstante, welche von der Struktur des PQC's abhängt.

Die Berechnung der Gradienten kann somit auf Grundlage der gleichen Schaltkreisarchitektur durchgeführt werden, wie die zur Berechnung der *Q-Werte*.

Der PQC wird für die Gradientenberechnung in einem Aktualisierungsschritt zwei Mal für jedes Gewicht ausgeführt. Damit ist diese auf dem Quantencomputer rechenintensiver, als das eigentliche Ermitteln der *Q-Werte*.

4. Reproduktion verwandter Arbeiten

Dieses Kapitel beschreibt den Versuchsaufbau verwandter Arbeiten, welche die in Kapitel 3 dargestellten Verfahren und Konzepte verwenden. Speziell werden die Arbeiten von Lookwood, Si [LS20] und Skolik et al. [SJD21] betrachtet. Das Ziel dieser Experimente ist es, die Leistungsfähigkeit von PQCs anhand von RL-Umgebungen zu evaluieren. In der Diskussion werden Stärken und Schwächen der jeweiligen Verfahrensumsetzungen aufgezeigt.

4.1. Beschreibung der Lernumgebungen

Die Leistung eines PQCs wird anhand der Höhe des *erwarteten Rückgabewertes* (siehe Gleichung 2.4) in speziellen RL-Umgebungen gemessen. In dieser Arbeit werden die Umgebungen *CartPole* [Opea] und *Blackjack* [Opeb] von *OpenAI Gym* [Bro+16] betrachtet. Diese können durch MDPs definiert werden und sind im Folgenden näher beschrieben.

4.1.1. *CartPole*

In der *CartPole* Umgebung [Opea] ist ein Stab an einem Wagen befestigt, der sich auf einer Strecke bewegt. Die Aufgabe eines RL-Agenten ist es, den Stab senkrecht auf dem Wagen zu balancieren, ohne dass dieser umfällt. Das kann erreicht werden, indem der Agent den Wagen in eine Richtung lenkt.

Diese Umgebung lässt sich als MDP formulieren. Ein Zustand der Umgebung besteht dabei aus den in Tabelle 4.1 dargestellten Komponenten.

Komponente	Minimum	Maximum
Position des Wagens	-2,4	2,4
Geschwindigkeit des Wagens	$-\infty$	∞
Winkel des Stabs	-24°	24°
Geschwindigkeit des Stabs an seiner Spitze	$-\infty$	∞

Tabelle 4.1.: Zustandskomponenten in *CartPole*

Der Agent kann zwei diskrete Aktionen durchführen. Er kann den Wagen nach links oder rechts bewegen.

Die Belohnung ist für jeden Schritt 1. Somit hängt der *erwartete Rückgabewert* von der Länge einer Episode ab. Eine Episode terminiert, wenn der Stab umkippt, der Wagen sich zu weit vom Mittelpunkt der Strecke entfernt oder der Agent den Stab lange genug balanciert hat. Dies wird mit folgenden Bedingungen definiert:

- Der Stabwinkel ist $\geq 12^\circ$ oder $\leq -12^\circ$
- Die Position des Wagens ist $\geq 2,4$ oder $\leq -2,4$
- Die Länge der Episode ist größer als 200

Die *Transitionsfunktion* für *CartPole* aus Gleichung 2.1 ist deterministisch und somit entweder 1 oder 0 für alle $s, s' \in \mathcal{S}$. Eine Belohnung, welche erreicht wird, wenn eine Aktion in einem Zustand durchgeführt wird, ist eindeutig.

Mögliche Zustandskodierungen In Abschnitt 3.3 wurden Möglichkeiten beschrieben, wie eine Zustandskomponente zu dem Winkel eines Rotationsgatters konvertiert werden kann. Da ein Zustand in *CartPole* aus vier Komponenten besteht, werden für die quantenbasierte Repräsentation dessen ebenfalls vier Qubits verwendet. Prinzipiell lässt sich jede mögliche Komponente eines Zustandes in *CartPole* *kontinuierlich* oder *gerichtet* kodieren. Eine *skalierte* Kodierung ist lediglich für die Position des Wagens und den Winkel des Stabes möglich, da sich die Werte dieser Komponenten in einem festen Bereich befinden. In den Experimenten wurden drei Konfigurationen an klassischen Kodierungen für *CartPole* getestet:

- kontinuierliche Kodierung aller Komponenten
- skalierte Kodierung aller möglicher Komponenten (Position des Wagen, Winkel des Stabs) und gerichtete Kodierung der anderen beiden Komponenten (Geschwindigkeit Wagen/Stab)
- skalierte Kodierung aller möglicher Komponenten und kontinuierliche Kodierung der anderen beiden Komponenten

Experimente mit einer rein gerichteten Kodierung wurden wegen eines zu erwartenden Informationsverlustes durch die binäre Beschaffenheit des Kodierungsverfahrens nicht durchgeführt.

4.1.2. *Blackjack*

Blackjack [Opeb] ist ursprünglich ein Kartenspiel. Das Ziel eines Spielers ist es, Karten zu erhalten, deren Summe sich so nahe wie möglich der Zahl 21 annähert, ohne darüber hinaus zu gehen. Dabei wird gegen einen sogenannten *Dealer* gespielt.

In der *OpenAI Gym* Umgebung wird mit einem Deck an unendlich vielen Farben gespielt. Jede Farbe besitzt 13 Karten. Dabei gibt es acht Karten mit den Werten 2 – 9, vier Karten mit dem Wert 10 und eine Karte, welche als Ass bezeichnet wird. Ein Ass nimmt den Wert 11 an, wenn die Summe der Karten des Spielers oder des *Dealers* mit Wert 11 kleiner oder gleich 21 ist. In diesem Fall wird das Ass 'nutzbar' genannt. Ansonsten hat das Ass den Wert 1.

Zu Beginn eines Spiels hat der *Dealer* zwei Karten vor sich liegen, wobei eine davon aufgedeckt ist. Der Spieler hat ebenfalls zwei Karten, wovon beide aufgedeckt sind. Der Spieler zieht weitere Karten, bis er sich entscheidet aufzuhören oder die Summe seiner Karten 21 übersteigt. Hat der Spieler aufgehört weitere Karten zu nehmen, deckt der *Dealer* seine verdeckte Karte auf und zieht so lange weitere Karten, bis die Summe seiner Karten 17 übersteigt.

Ein Spieler gewinnt, wenn am Ende des Spiels die Summe der Karten des *Dealers* höher als 21 ist und die Summe des Spielers nicht, oder wenn die Summe des Spielers größer ist, als die des *Dealers*. Ansonsten verliert der Spieler.

Die Umgebung kann als MDP definiert werden. Dabei besteht ein Zustand aus den in Tabelle 4.2 gelisteten Komponenten. Da alle Zustandskomponenten diskrete Werte annehmen, sind auch alle Zustände diskret.

Komponente	Mögliche Werte
Summe des Spielers	4-21
Aufgedeckte Karte des <i>Dealers</i>	1-10
'Nutzbares' Ass auf der Hand des Spielers	Wahr oder Falsch

Tabelle 4.2.: Zustandskomponenten in *Blackjack*

Der Agent kann zwei diskrete Aktionen durchführen. Er kann entweder eine weitere Karte nehmen oder das Spiel beenden.

Die Belohnung ist +1, falls der Spieler gewinnt, -1, falls der Spieler verliert und ansonsten 0. Die *Discount*-Rate ist 1, damit der *erwartete Rückgabewert* entweder -1 oder 1 ist.

Eine Episode terminiert, unter den folgenden Bedingungen:

- Die Summe des Spielers übersteigt 21
- Der Spieler zieht keine weitere Karte

Die *Transitionsfunktion* ist in *Blackjack* nicht deterministisch, da eine weitere Karte zufällig aus dem Deck gezogen wird.

Mögliche Zustandskodierungen Ein Zustand besteht in *Blackjack* aus drei Komponenten. Somit werden für eine quantenbasierte Repräsentation im Fall von komponentenweiser Kodierung drei Qubits benötigt. In den Experimenten wurde entweder eine *skalierte* oder eine *kontinuierliche* Kodierung der Spieler-summe und der Karte des Dealers durchgeführt. Ob ein Ass 'nutzbar' ist, wird in beiden Konfigurationen binär kodiert.

4.2. Umsetzung von quantenbasiertem RL

In den Experimenten wurde Algorithmus 1, mit den in Abschnitt 3.1 beschriebenen Änderungen für PQCs, durchgeführt. Die Implementierung wurde dabei von Wolf [WM21] übernommen und den Anwendungsfällen angepasst. Um den Fortschritt des Lernprozesses verfolgen zu können, wird der PQC oder das NN nach einer definierten Anzahl an Schritten evaluiert. In einer Evaluation wird eine feste Anzahl an Schritten in einer Kopie der Umgebung durchgeführt, in denen die Gewichte des PQCs oder des NNs nicht aktualisiert werden. Ein Intervall zwischen zwei Evaluationsabschnitten wird *Epoche* genannt. Anhand des Durchschnitts der in der Evaluation erreichten Rückgabewerte werden im Folgenden verschiedene PQCs oder NNs miteinander verglichen.

Alle Experimente in diesem Kapitel wurden auf einem Quantensimulator durchgeführt und mit den Bibliotheken *Tensorflow Quantum* [Bro+20] und *Cirq* [Dev21] realisiert.

Parametereinstellungen Die verwendeten Hyperparameter sind in Tabelle 4.3 aufgelistet. Diese wurden hauptsächlich aus den verwandten Arbeiten [Sko+21; LS20] übernommen. Da allerdings nicht alle Parameter in [LS20] definiert sind und mit der Konfiguration aus [SJD21], im Vergleich zu experimentell ermittelten Werten, eine schlechtere Leistung erzielt wurde, wurden einige Parameter angepasst.

Damit ein Vergleich der Leistungen von verschiedenen PQC-Strukturen, Kodierungsmechanismen und Extraktionsverfahren durchgeführt werden kann, sind die Hyperparameter für alle durchgeführten Experimente mit PQCs gleich. Experimente für klassische NNs wurden analog durchgeführt, allerdings mit einer Lernrate von 0,01 statt 0,001. Diese Lernrate wurde in [SJD21] ebenfalls für den klassischen Fall verwendet.

4.3. Resultate eines klassischen RL-Verfahrens

In Abbildung 4.1 ist dargestellt, wie klassische NNs im Verlauf des Lernprozesses die beiden beschriebenen Umgebungen lösen. Die verwendeten NNs haben

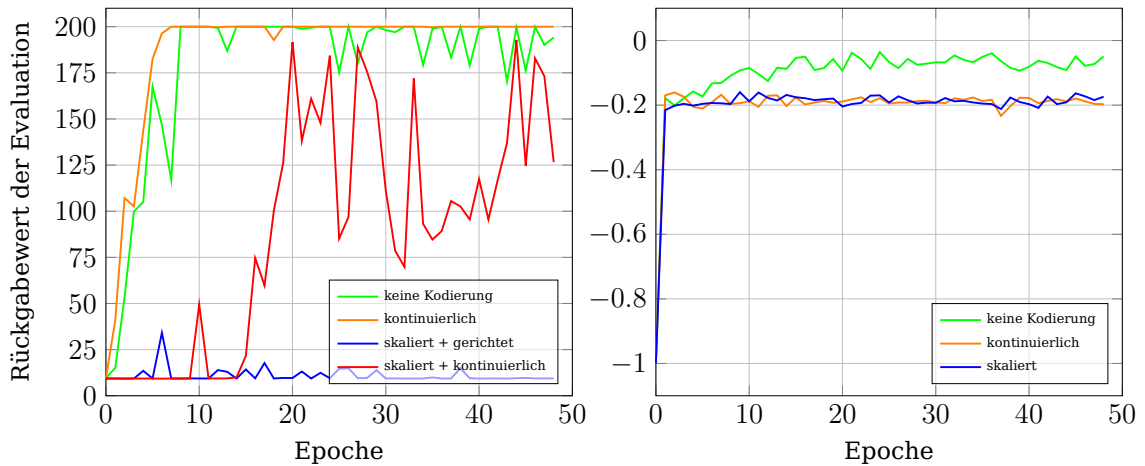
Hyperparameter	Beschreibung	Wert
Schritte	Anzahl der Schritte	50 000
Evaluationsschritte	Anzahl der Schritte zur Evaluation	5 000
Evaluationshäufigkeit	Anzahl der Schritte zwischen den Evaluationsabschnitten	500
Schichten	Anzahl der in Abbildung 4.2 dargestellten Schichten	3
γ	<i>Discount</i> -Rate	0,99
α	Lernrate	0,001
Batch-Größe	Anzahl der Elemente in einem Batch	32
ϵ_{init}	Initialer Wert für die ϵ -greedy-Strategie	1,0
ϵ_{dur}	Anzahl der Schritte bis der Faktor der ϵ -greedy-Strategie das Minimum erreicht	20 000
ϵ_{min}	Minimaler Wert für die ϵ -greedy-Strategie	0,01
Häufigkeit der Ziel-PQC-Aktualisierung	Anzahl der Schritte, zwischen der Aktualisierung des Ziel-PQC	500
Größe des Zwischenspeichers	Maximale Anzahl der Transitionen in dem Zwischenspeicher	50 000
Schritte zur Speicherinitialisierung	Anzahl der Schritte zu Beginn für die Initialisierung des Speichers. Dabei werden die Gewichte des PQC nicht aktualisiert.	1 000

Tabelle 4.3.: Verwendete Hyperparameter für die Experimente

zwei NN-Schichten mit jeweils sechs und zwei Knoten. Damit enthalten sie im Fall der *CartPole*-Umgebung 44 und im Fall der *Blackjack*-Umgebung 38 Gewichte. Die Gewichte wurden mit gleichverteilt zufälligen Werten initialisiert und sind für alle NNs der selben Struktur gleich. Um bewerten zu können, wie stark die in den Eingabezuständen enthaltenen Informationen durch Konvertierung zu Rotationswinkeln beeinflusst werden, wurden die beschriebenen Kodierungskonfigurationen auch auf die Eingaben der klassischen NNs angewandt.

Dabei wurde festgestellt, dass ein NN, auf das keine Kodierung angewandt wurde, in der *CartPole*-Umgebung nach weniger als 10 Epochen den optimalen Rückgabewert von 200 erreicht und der zugehörige Agent im weiteren Lernprozess eine nahezu optimale Strategie verfolgt. Ein NN, bei welchem die Eingabezustände kontinuierlich kodiert wurden, erzielt ähnliche Ergebnisse, wie in Abbildung 4.1a dargestellt ist. Dagegen verbleibt das NN, bei welchem die Zustandskomponenten skaliert und gerichtet kodiert wurden, im Verlauf des Lernprozesses auf einem niedrigen Rückgabewert. Dies wurde auch in [LS20] festgestellt, wo ein ähnliches NN mit der gleichen Kodierung getestet wurde. Die Kombination aus skaliertem und kontinuierlicher Kodierung lässt den Rückgabewert im Lernvorgang schwanken, wobei jedoch ein Anstieg des Rückgabewertes ab 15 Epochen zu erkennen ist. Einer optimalen Strategie wird hierbei nicht gefolgt.

Ausgehend von diesen Ergebnissen wird gefolgert, dass für klassische NNs bei



(a) Lernprozess in *CartPole*

(b) Lernprozess in *Blackjack*

Abbildung 4.1.: Verlauf des klassischen Lernprozesses in den Umgebungen *CartPole* und *Blackjack* mit verschiedenen Kodierungsstrategien. In den Graphen wird zu jeder *Epoche* der Durchschnitt der Rückgabewerte aus dem jeweils nachfolgenden Evaluationsabschnitt angegeben.

der kontinuierlichen Kodierung keine Zustandsinformationen verloren gehen, welche zu einer guten Strategie des Agenten in der *CartPole*-Umgebung führen. Die Nichtlinearität des Arkustangens scheint sich sogar günstig auf den Lernprozess auszuwirken. In der skalierten Kodierung werden die Zustandskomponenten mit einem festen Wertebereich linear skaliert, sodass die beobachtete Begünstigung durch den Arkustangens nicht eintritt. Dies könnte das Schwanken des Rückgabewertes in der Kombination mit der kontinuierlicher Kodierung erklären. Dabei handelt es sich jedoch lediglich um eine Vermutung. Der simple Ansatz der gerichteten Kodierung reicht im klassischen Fall nicht aus, um die unendlichen Zustandswertebereiche der *CartPole*-Umgebung abzubilden.

Alle getesteten Konfigurationen an NNs erzielten in der *Blackjack*-Umgebung bereits nach einer Epoche einen Rückgabewert von etwa -0,2, wie in Abbildung 4.1b dargestellt ist. Die NNs, deren Eingabezustände kodiert wurden, änderten ihren Rückgabewert im Verlauf des Lernprozesses kaum. In dem Lernvorgang mit dem NN, auf das keine Kodierung angewandt wurde, stieg der Rückgabewert noch etwas an und konvergierte gegen einen Wert knapp unter 0. Da *Blackjack* als Casino-Spiel konstruiert ist, bei dem ein Spieler häufiger verliert als gewinnt, ist dies auch der optimale durchschnittliche Rückgabewert, welcher einfacher zu erreichen ist, als in *CartPole* [LS20]. Demzufolge erzielten alle Konfigurationen an NNs nach kurzer Zeit ähnliche Ergebnisse.

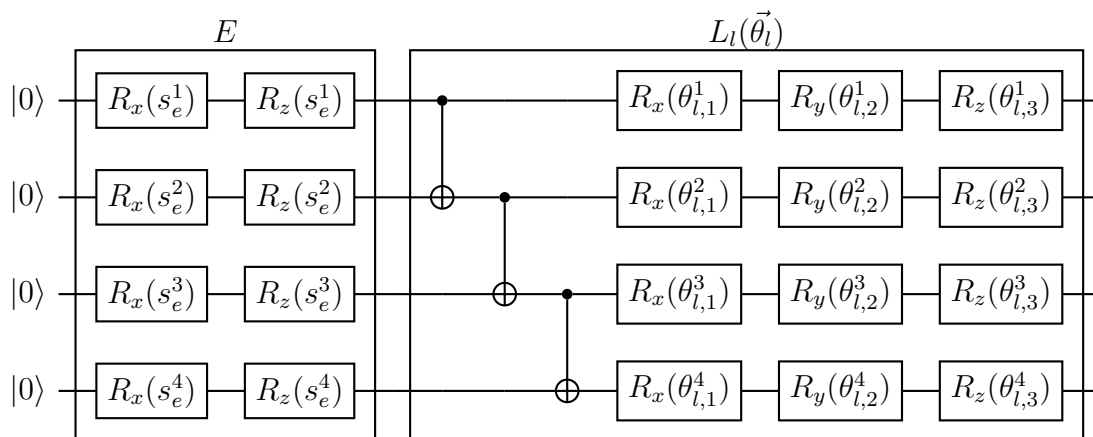


Abbildung 4.2.: PQC-Struktur nach Lockwood und Si. Dabei besteht die Eingabeschicht E aus den Gattern R_x und R_z auf jedem Qubit. Die Winkel \vec{s}_e sind die kodierten Zustandkomponenten des Eingabezustandes. In dieser Arbeit enthält der PQC drei parametrisierten Schichten $L_l(\theta_l)$. Nach diesen folgt eine Messung des Quantenzustandes oder Quanten-Bündelungsoperationen. Der dargestellte PQCs kann in der *CartPole*-Umgebung eingesetzt werden. In der *Blackjack*-Umgebung wird die gleiche Struktur mit einem Qubit weniger verwendet.

4.4. Resultate eines Quanten-RL-Verfahrens

Die im Folgenden beschriebenen Experimente basieren auf der PQC-Struktur von Lockwood und Si [LS20]. Die Schaltkreisarchitektur wird in den beschriebenen RL-Umgebungen evaluiert und hinsichtlich der Leistung mit einem klassischen NN verglichen. Die verschiedenen Kodierungs- und Extraktionsverfahren aus Abschnitt 3.3 und Abschnitt 3.4 wurden in den Experimenten getestet.

4.4.1. Architektur des Quantenschaltkreises

Die verwendete Struktur des PQCs ist in Abbildung 4.2 dargestellt. Die Eingabeschicht E setzt sich aus den Rotationsgattern R_x und R_z auf jedem Qubit zusammen. Beide Gatter werden um den Winkel rotiert, welcher durch die Kodierung einer klassischen Zustandskomponente ermittelt wurde.

Die parametrisierten Schichten L bestehen jeweils einer Sammlung an *CNOT*-Gattern, welche die Qubits verschränken und aus den drei Rotationsgattern R_x, R_y, R_z auf jedem Qubit. Die Rotationsgatter in der Schicht L_l auf dem Qubit mit dem Index i sind durch die Gewichte $\theta_{l,1}^i, \theta_{l,2}^i, \theta_{l,3}^i$ parametrisiert.

In den durchgeführten Experimenten wird, wie in [LS20], ein PQC mit drei Schichten verwendet. Der PQC hat somit im Fall von *CartPole* 36 Parameter

und im Fall von *Blackjack* 27 Parameter. Wird die Quanten-Bündelungsoperation zur Extraktion der Daten angewandt, enthält der PQC für *CartPole* insgesamt 48 und für *Blackjack* 33 Parameter. Lockwood und Si kodierten in ihren Experimenten die Zustandskomponenten, welche sich in einem festen Wertebereich befinden, skaliert und die Übrigen gerichtet. Zur Extraktion der Daten verwendeten sie die zwei in Abschnitt 3.4 beschriebenen Verfahren des hybriden Modells und der Quanten-Bündelung.

In dieser Arbeit wurden diese Experimente reproduziert, sowie zusätzliche Versuche mit der Extraktion der Daten durch den Erwartungswert durchgeführt. Zudem wurden die beschriebenen Kombinationen an Kodierungsverfahren in der jeweiligen Umgebung getestet. Dabei wurden alle Parameter der PQCs mit 0 initialisiert, um für jede Konfiguration eine äquivalente Ausgangslage zu generieren.

4.4.2. Resultate in *CartPole*

In Abbildung 4.3 sind die Verläufe der Lernprozesse abgebildet, welche mit der beschriebenen PQC-Struktur für die *CartPole*-Umgebung durchgeführt wurden. Dabei wurden alle beschriebenen Kombinationen von Kodierungs- und Extraktionsverfahren getestet.

Im Gegensatz zu den Ansätzen mit klassischen NNs, erreichte keiner der getesteten Agenten, welcher auf der beschriebenen PQC-Struktur basiert, in der durchgeführten Lernperiode einen stabilen, optimalen Rückgabewert von 200. So stiegen die Rückgabewerte der getesteten *hybriden* Modelle, deren Eingabedaten kontinuierlich (rechte Spalte, orange Kurve), sowie gerichtet und skaliert kodiert wurden (rechte Spalte, blaue Kurve), im Verlauf des Lernprozesses un- stetig auf einen durchschnittlichen Wert von etwa 40-50 an. Dies sind ähnliche Ergebnisse, wie sie in [LS20] beobachtet wurden. Ein Agent, welcher diese Werte erzielt, verfolgt jedoch keine gute Strategie. Wurden die Eingabedaten des *hybriden* Modells kombiniert skaliert und kontinuierlich kodiert (rechte Spalte, rote Kurve), erreichte der zugehörige Agent zur Mitte des Lernprozesses hohe Werte von etwa 190. Der Rückgabewert fällt in den folgenden Epochen allerdings wieder auf niedrigere Werte ab.

Die Agenten der PQCs, von welchen die *Q-Werte* durch Quanten-Bündelungsoperationen oder lediglich durch Erwartungswerte extrahiert wurden, erzielten im Verlauf des Lernvorgangs schwankende Rückgabewerte. Eine Tendenz zu einer kontinuierlichen Steigerung ist bei dem Lernvorgang zu erkennen, in dem die *Q-Werte* des PQCs mit Erwartungswerten extrahiert wurden und die Eingabedaten skaliert und gerichtet kodiert wurden (mittlere Spalte, blaue Kurve). Darin steigt der Rückgabewert auf etwa 175 an, schwankt allerdings während des Lernprozesses, was im Folgenden als *Instabilität* bezeichnet wird. In den Experimenten, in denen eine Extraktion durch Erwartungswerte mit anderen

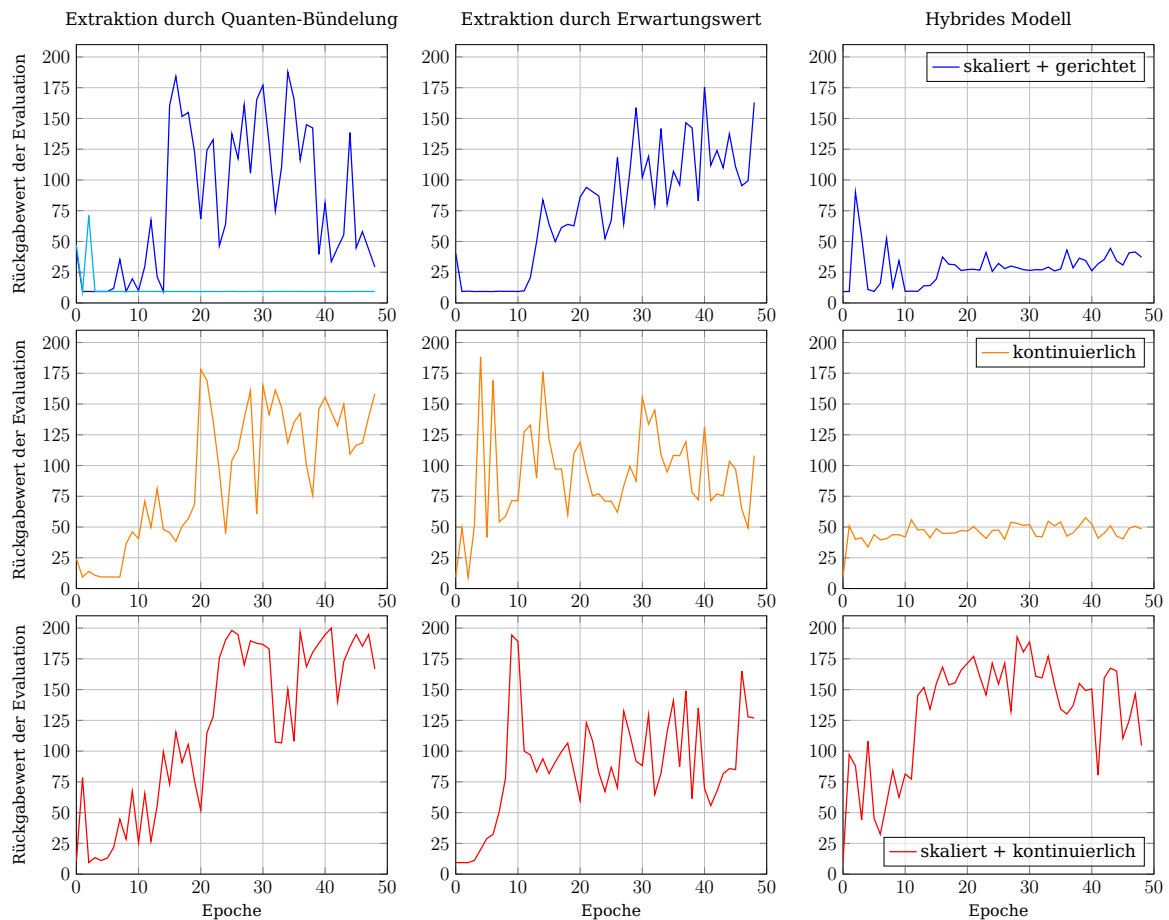


Abbildung 4.3.: Verlauf der Lernprozesse von PQC's nach der Strukturdefinition von Lockwood und Si in der *CartPole*-Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien. Die dargestellten Graphen bilden die durchschnittlichen Rückgabewerte der Evaluationsabschnitte nach jeder Epoche ab. Alle Lernvorgänge in der linken Spalte wurden mit PQC's durchgeführt, welche eine Quanten-Bündelungsoperation nutzen, um Quantendaten zu extrahieren. In der mittleren Spalte werden die Verläufe der Lernprozesse gezeigt, in denen die *Q*-Werte lediglich durch Berechnung von Erwartungswerten ermittelt werden. Die rechte Spalte stellt die Lernvorgänge des *hybriden* Modells dar. In den Lernvorgängen der oberen Reihe wurden Eingabedaten skaliert und gerichtet, in denen der mittleren Reihe kontinuierlich und in denen der unteren Reihe skaliert und kontinuierlich kodiert. Die cyan-farbige Kurve bezeichnet die Ergebnisse des Lernprozesses, in dem die Ausgabewerte des PQC's nicht zusätzlich durch einen gelernten Faktor skaliert wurden, wie es in Abschnitt 3.4 beschrieben ist.

Kodierungsstrategien getestet wurde, konnten keine Tendenzen für eine Steigerung des Rückgabewertes festgestellt werden. Stattdessen erreichen beide Agenten, welche die Eingabedaten durch die kontinuierliche Kodierung (mittlere Spalte, orange Kurve) und der Kombination aus kontinuierlicher und skaliertem Kodierung (mittlere Spalte, rote Kurve) zu Quantenzuständen konvertieren, beinahe optimale Rückgabewerte innerhalb der ersten 10 Epochen und schwankende Werte zwischen circa 75 und 150 in dem weiteren Verlauf.

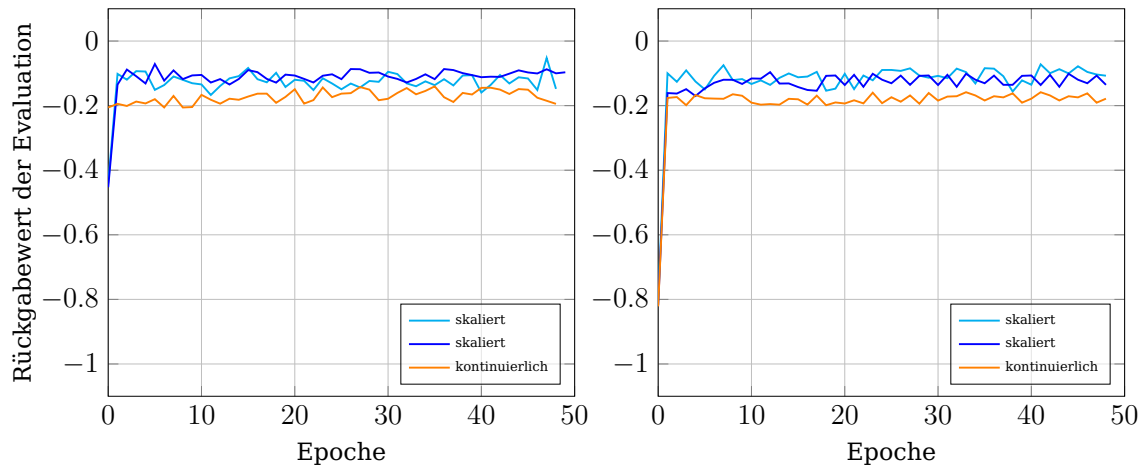
Die Rückgabewerte, die von dem Agenten erzielt werden, welcher die Eingabedaten kontinuierlich kodiert und die Quantendaten durch Quanten-Bündelungsoperationen extrahiert (linke Spalte, orange Kurve), steigen während des Lernvorgangs mit Abweichungen auf circa 175 an. Bei der Konfiguration, in der die Eingabedaten kontinuierlich und skaliert kodiert wurden (linke Spalte, rote Kurve), wird ein ähnliches, *instabiles* Verhalten festgestellt, wobei der Rückgabewert zum Ende des Lernprozesses auf das Optimum von 200 ansteigt, dieses jedoch nicht stabil beibehält. Die Ergebnisse des Agenten, welcher zur Kodierung die Kombination aus skaliertem und gerichteter Kodierung verwendet (linke Spalte, blaue Kurve), erreicht während des Lernprozesses schwankende hohe und niedrige Rückgabewerte, woraus sich nur schwer eine Strategie ableiten lässt.

Die Ausgabe der PQC's in den bisher beschriebenen Experimenten wurde skaliert, um das Spektrum der zukünftigen Belohnungen abzudecken. In der *CartPole*-Umgebung ist dies ein Wert von bis zu 200, wobei die *Discount*-Rate noch beachtet werden sollte. Der Wertebereich der Erwartungswerte eines PQC's liegt in $[-1, 1]$, was für die Abbildung der zukünftigen Belohnungen in *CartPole* nicht ausreicht. Die cyan-farbige Kurve in der linken Spalte bezeichnet einen Agenten, dessen PQC nicht durch einen gelernten Faktor skaliert wurde. Er behält über alle Epochen einen sehr niedrigen Rückgabewert bei.

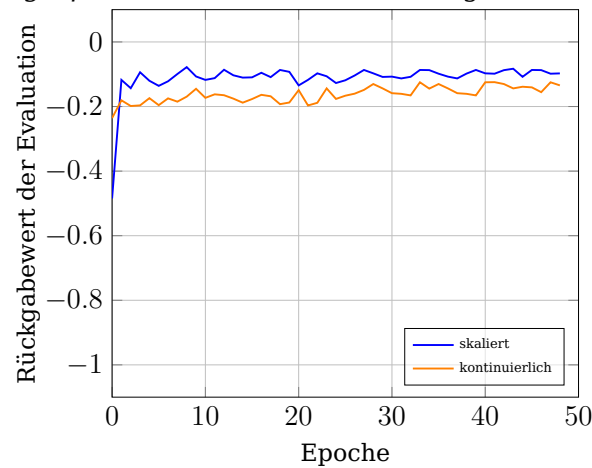
4.4.3. Resultate in *Blackjack*

In Abbildung 4.4 werden analog die Verläufe der Lernprozesse in *Blackjack* dargestellt. Mit allen Konfigurationen an Kodierungs- und Extraktionsverfahren konvergiert der durchschnittliche Rückgabewert bereits nach einer Epoche gegen einen Wert knapp unter 0. Dabei wurde festgestellt, dass die PQC's, welche kontinuierlich kodierte Eingaben erhalten, einen minimal kleineren Rückgabewert erreichen, als PQC's, welche skaliert kodierte Eingaben erhalten. Die Extraktionsverfahren scheinen darauf keinen Einfluss zu haben. Die schlechtere Leistung des kontinuierlichen Verfahrens lässt sich möglicherweise dadurch begründen, dass die Nichtlinearität des Arkustangens hohe Kartenwerte auf ein ähnliches Eingabedatum abbildet, wodurch der Agent diese schlechter unterscheiden kann. Dabei handelt es sich jedoch lediglich um eine Vermutung.

Wie in Abbildung 4.4a und Abbildung 4.4b dargestellt, erzielten PQC's, deren Ausgaben nicht mit einem lernbaren Faktor skaliert wurden, ähnliche Rück-



(a) Lernprozess mit Extraktion durch *Quanten-Bündelungsoperationen* (b) Lernprozess mit Extraktion durch Erwartungswerte



(c) Lernprozess des *hybriden* Modells

Abbildung 4.4.: Verlauf der Lernprozesse von PQC's nach der Strukturdefinition von Lockwood und Si in der *Blackjack*-Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien. Die dargestellten Graphen bilden die durchschnittlichen Rückgabewerte der Evaluierungsabschnitte nach jeder Epoche ab. Alle Lernvorgänge in a) wurden mit PQC's durchgeführt, welche eine Quanten-Bündelungsoperation nutzen, um Quantendaten zu extrahieren. In Graph b) werden die Verläufe der Lernprozesse gezeigt, in denen die *Q*-Werte nur durch Berechnung von Erwartungswerten ermittelt werden. Graph c) stellt die Lernvorgänge des *hybriden* Modells dar. Die cyan-farbigen Kurven bezeichnen die Ergebnisse der Lernprozesse, in denen die Ausgabewerte der PQC's nicht zusätzlich durch einen gelernten Faktor skaliert wurden.

gabewerte, wie die POCs, deren Ausgaben skaliert wurden. Der Grund dafür ist, dass der Erwartungswert eines Quantenschaltkreises den gleichen Wertebereich einnimmt, wie der Rückgabewert in *Blackjack*. In der *Blackjack*-Umgebung ist somit keine Skalierung des Ausgabewertes nötig, damit ein Agent eine optimale Strategie verfolgt.

Die Leistungen der verschiedenen NNs und POCs unterscheiden sich kaum, da, wie bereits in Abschnitt 4.3 beschrieben, eine optimale Strategie einfacher von einem Agenten erreicht werden kann [LS20]. Daher wurde in weiteren Experimenten auf ein Auswerten der Ergebnisse für die *Blackjack*-Umgebung verzichtet.

4.4.4. Diskussion

Aus den vorliegenden Ergebnissen wird gefolgert, dass der POCs mit der beschriebenen Struktur potenziell eine optimale Strategie in simplen RL-Umgebungen erreichen kann. So wurden nahezu optimale, stabile Rückgabewerte in der diskreten *Blackjack*-Umgebung erzielt. In der komplexeren *CartPole*-Umgebung unterliegen durch POCs gelernte Strategien allerdings starken Schwankungen.

Alle Kodierungsverfahren scheinen sich für den Einsatz in quantenbasierten RL zu eignen. Dabei erzielte die skalierte Kodierung in der *Blackjack*-Umgebung im Vergleich zu der kontinuierlichen minimal höhere Rückgabewerte, was in den Experimenten der klassischen NNs nicht festgestellt wurde. In der *CartPole*-Umgebung verfolgen Agenten, deren Eingaben mit einem festen Wertebereich skaliert und die restlichen Eingaben kontinuierlich kodiert wurden, tendenziell eine stabilere Strategie, als andere getestete Agenten. Dies wird in dem Experiment deutlich, in dem die Quantendaten mittels Quanten-Bündelung aus dem POC extrahiert wurden, wie in Abbildung 4.3 (linke Spalte, rote Kurve) dargestellt ist. In diesen Graphen ähneln die Verläufe der Rückgabewerte während des Lernvorgangs dem, welcher auch klassisch durch die skalierte und kontinuierliche Kodierungskombination erreicht wurde (Abbildung 4.1a, rote Kurve).

Ansonsten lassen sich auf Grundlage der bisher beschriebenen Experimente, wegen der starken Schwankungen des Rückgabewertes während des Lernprozesses bei den anderen Kodierungsverfahren, keine Stärken und Schwächen ausarbeiten. Die optimale Leistung, welche im klassischen Fall mit der rein kontinuierlichen Kodierung beobachtet wurde (Abbildung 4.1a, orange Kurve), konnte im quantenbasierten Kontext nicht festgestellt werden. Auch die klassisch beobachtete, schlechte Leistung der skaliert-gerichteten Kodierungskombination (Abbildung 4.1a, blaue Kurve) konnte in Quanten-RL nicht registriert werden.

In der *Blackjack*-Umgebung konnten kaum Unterschiede in der Leistung beobachtet werden, wenn verschiedene Extraktionsverfahren verwendet wurden. In

der *CartPole*-Umgebung hingegen hat sich das hybride Modell, unter Verwendung der skaliert-gerichteten und der rein kontinuierlichen Kodierung, als eher ungeeignet herausgestellt. Ob eine Quanten-Bündelungsoperation sinnvoll ist, oder lediglich die Berechnung der Erwartungswerte zur Extraktion ausreicht, lässt sich auf Grund der Schwankungen des Rückgabewertes nicht feststellen.

4.5. Resultate einer alternativen PQC-Struktur

Im Folgenden werden die Experimente betrachtet, welche auf der in [SJD21] beschriebenen PQC-Struktur basieren. Dabei werden, analog zu den durchgeführten Experimenten mit der PQC-Struktur von Lockwood und Si, die vorgestellten Kodierungs und Extraktionsverfahren evaluiert. Die Resultate beschreiben die Leistungen der getesteten Verfahren in der *CartPole*-Umgebung.

4.5.1. Architektur des Quantenschaltkreises

Die Struktur des im Folgenden verwendeten PQC ist in Abbildung 4.5 dargestellt. Die Eingabeschicht E besteht dabei aus einem Rotationsgatter R_x auf jedem Qubit. Der Winkel der Rotation ist dabei eine kodierte Komponente eines Eingabedatums. Es ist kein zusätzliches Z-Rotationsgatter in der Eingabeschicht, wie in [LS20], enthalten. Diese zusätzliche Rotation ist auch nicht nötig, um die Informationen der Eingaben in den PQC zu übertragen.

Eine parametrisierte Schicht L besteht aus zwei Rotationsgattern R_y, R_z auf jedem Qubit, sowie einer Sammlung an CZ-Gattern, wie in Abbildung 4.5 dargestellt.

In den folgenden Experimenten besteht ein PQC aus fünf Schichten nach der Strukturdefinition von Skolik et al. und besitzt damit für die *CartPole*-Umgebung 40 Parameter (ohne Bündelungsoperation). In [SJD21] werden Eingabedaten kontinuierlich kodiert und Quantendaten über Berechnung der Erwartungswerte extrahiert. In den Experimenten dieser Arbeit werden analog zu Abschnitt 4.4 die beschriebenen Kombinationen aus Kodierungs- und Extraktionsverfahren auch für diese Schaltkreisarchitektur evaluiert. Die Parameter wurden dabei mit 0 initialisiert.

4.5.2. Resultate in *CartPole*

Eine Übersicht über die Experimente mit der Schaltkreisarchitektur von Skolik et al. ist in Abbildung 4.6 dargestellt.

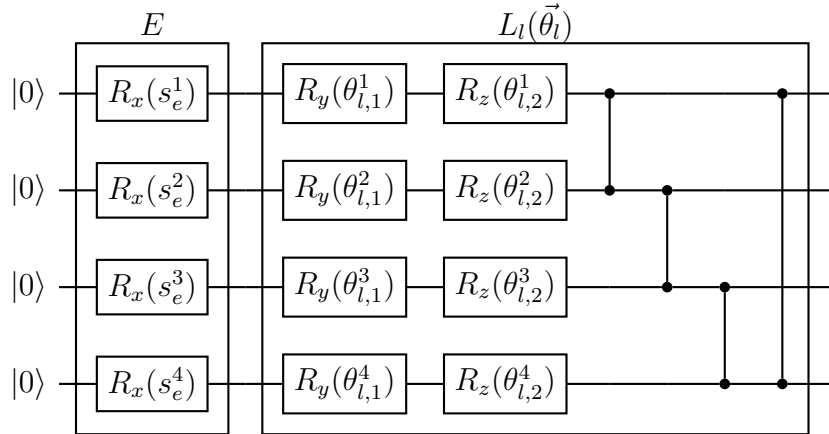


Abbildung 4.5.: PQC-Struktur nach Skolik et al. Die Eingabeschicht E besteht aus R_x -Gattern auf jedem Qubit, welche jeweils um den Winkel s_e^i rotiert werden. s_e^i ist eine kodierte Komponente des Eingabezustandes. Der in dieser Arbeit verwendete PQC umfasst fünf Schichten $L_l(\vec{\theta}_l)$. Die Quantenzustände werden nach den parametrisierten Schichten gemessen, oder es erfolgt eine Quanten-Bündelungsoperation.

Es wurde festgestellt, dass eine skaliert-kontinuierliche Kodierung der Eingabedaten des PQCs in Kombination mit einer Extraktion der Q -Werte durch eine Quanten-Bündelungsoperation (linke Spalte, rote Kurve) oder durch reine Erwartungswertberechnung (mittlere Spalte, rote Kurve) zu einer stabilen, optimalen Strategie führen kann. Die Konfiguration mit der Quanten-Bündelung erreichte und hielt den optimalen Rückgabewert ab etwa 17 Epochen, die Konfiguration mit dem Erwartungswert ab etwa 10 Epochen. Damit erreichten die auf PQCs basierenden Agenten circa zwei Epochen später eine optimale Strategie, als im klassischen Fall. Im Vergleich zu den Ergebnissen mit der Schaltkreisarchitektur von Lockwood und Si stellt dies eine klare Verbesserung dar, da dort keine stabile, optimale Strategie erreicht wurde, obwohl eine ähnliche Anzahl an Parametern verwendet wurde.

Auch im *hybriden* Modell wurden mit der skaliert-kontinuierlichen Kodierung (rechte Spalte, rote Kurve) ab circa 10 Epochen optimale Rückgabewerte erzielt. Allerdings blieben diese hohen Werte während des Lernprozesses nicht stabil erhalten, sondern sanken ab 30 Epochen auf einen Wert von etwa 70 ab und schwankten im weiteren Verlauf stark. Die Rückgabewerte mit der skaliert-gerichteten (rechte Spalte, blaue Kurve), sowie mit der kontinuierlichen Kodierung (rechte Spalte, orange Kurve) stiegen im *hybriden* Modell im Vergleich zu der äquivalenten Konfiguration mit dem PQC nach Lockwood und Si auf einen etwas höheren Wert an. Dabei blieben sie jedoch unter dem Rückgabewert, welcher durch die skaliert-kontinuierliche Kodierung erzielt wurde.

Die PQCs, deren Eingabedaten skaliert-gerichtet kodiert wurden, erzielten in der Konfiguration, in welcher die Q -Werte durch Erwartungswerte extrahiert

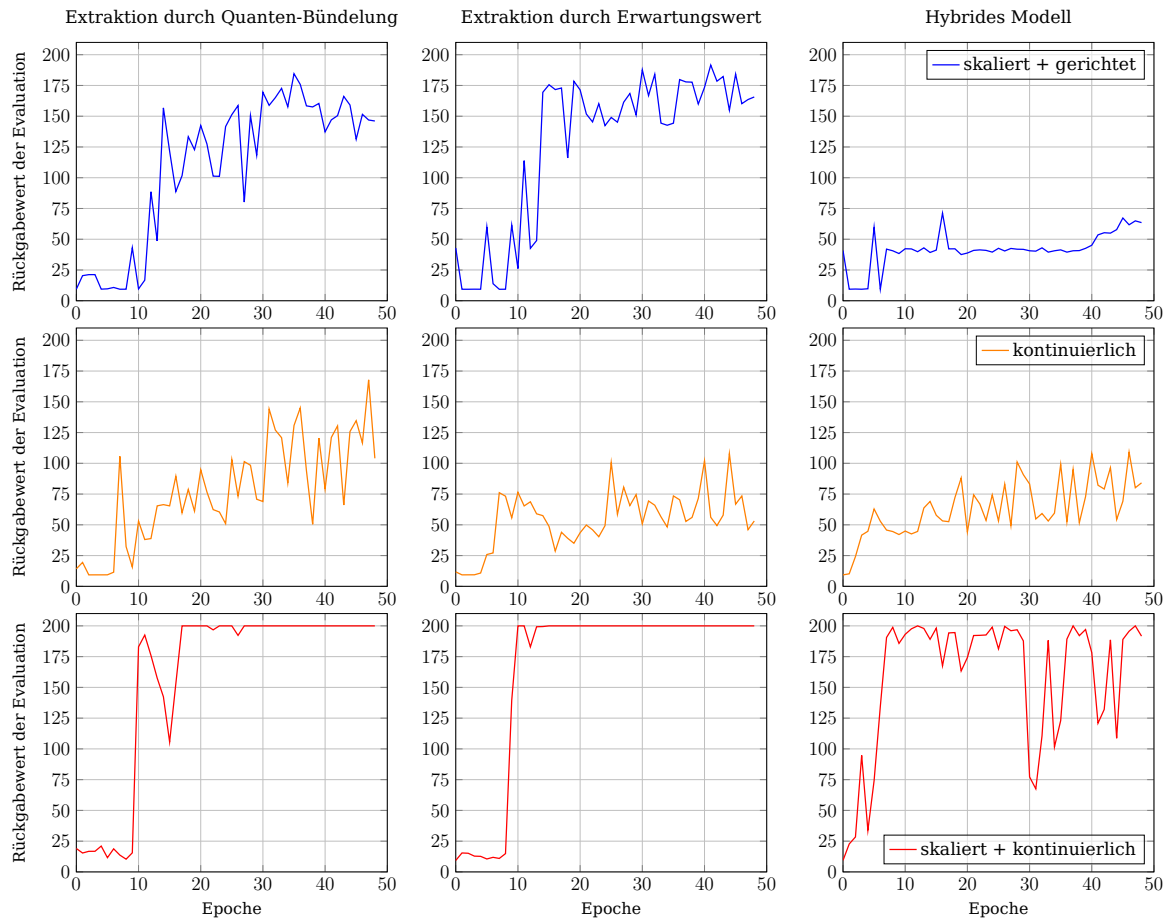


Abbildung 4.6.: Verlauf der Lernprozesse von PQC's nach der Strukturdefinition von Skolik et al. in der *CartPole*-Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien. Die Einteilung der Graphen ist, wie in Abbildung 4.3, beschrieben.

wurden (mittlere Spalte, blaue Kurve), nahezu konstant steigende Rückgabewerte bis etwa 175. Einen ähnlichen Verlauf nahm der Rückgabewert in der Konfiguration, in der die Quantendaten durch Bündelung extrahiert wurden (linke Spalte, blaue Kurve), wobei im Fall der Bündelungsoperation der Rückgabewert während des Lernprozesses etwas stärker schwankte.

Die erreichten Rückgabewerte unter Verwendung der kontinuierlichen Kodierung und der Quanten-Bündelung (linke Spalte, orange Kurve) stiegen etwas langsamer und instabiler an, als die Rückgabewerte durch skaliert-gerichtete Kodierung, erreichten aber einen ähnlichen Wert zum Ende des Lernprozesses. Der PQC, von welchem durch Erwartungswertberechnung *Q-Werte* extrahiert und dessen Eingabedaten kontinuierlich kodiert wurden (mittlere Spalte, orange Kurve), erzielte einen instabileren Rückgabewerteverlauf bis ungefähr 100. Dies ähnelt auch den in [SJD21] erreichten Ergebnissen.

4.5.3. Diskussion

Allgemein ist festzustellen, dass nahezu jede getestete Konfiguration an Kodierungs- und Extrahierungsverfahren mit der PQC-Struktur von Skolik et al. einen stabileren und höheren Verlauf an Rückgabewerten erzielt, als mit der Struktur von Lockwood und Si. In einer parametrisierten PQC-Schicht nach Skolik et al. werden auf einem Qubit lediglich zwei statt drei Rotationsgatter angewandt. Eine Rotation um zwei Achsen in der Bloch Kugel reicht aus, um jede mögliche Operation auf einem einzelnen Qubit durchzuführen [Shi+13]. Daher ist die Rotation um die zusätzliche Achse in einer Schicht der PQC-Struktur von Lockwood und Si nicht notwendig. In den Experimenten wurden PQCs mit einer ähnlichen Anzahl an Parametern verglichen. Da eine PQC-Schicht nach Skolik et al. weniger Parameter enthält, besteht der PQC aus mehr Schichten als der PQC nach Lockwood und Si. Der dadurch entstehende stärkere Grad an Verschränkung, sowie die größere Anzahl an Rotationsoperationen auf einzelnen Qubits, wirkt sich positiv darauf aus, wie ein PQC eine optimale Strategie erlernt.

Wurden die Eingabedaten eines PQCs, welche in einem festen Bereich liegen, skaliert kodiert und die Eingaben mit unendlichem Wertebereich kontinuierlich kodiert, konnten optimale Strategien von einem Agenten erzielt werden. Eine skalierte Kodierung bildet den Wertebereich einer Eingabekomponente auf den Bereich $[0, 2\pi]$ ab, wodurch keine Informationen verloren gehen. Zudem wird mit einem Winkel in diesem Bereich das volle Rotationsspektrum um eine Achse in der Bloch-Kugel ausgenutzt. Wird lediglich die kontinuierliche Kodierung verwendet, werden alle Eingabedaten durch einen Arkustangens in einen Wertebereich von $[-\frac{\pi}{2}, \frac{\pi}{2}]$ verschoben. Damit können auch Werte aus einem unendlichen Bereich ohne Informationsverlust kodiert werden. Allerdings sind damit lediglich Rotationen in einer Hälfte des Rotationsspektrums möglich,

wodurch dieses nicht voll ausgenutzt wird. Dies bietet möglicherweise eine Erklärung dafür, dass mit der reinen kontinuierlichen Kodierung vergleichsweise schlechtere und instabilere Leistungen erzielt wurden, als mit den anderen kombinierten Kodierungsverfahren. Ein Agent erzielte mit der gerichteten und der skalierten Kodierung durchschnittlich schlechtere Leistungen, als die skaliert-kontinuierliche Kombination. Der Grund dafür ist vermutlich, dass mit der gerichteten Kodierung bei der Abbildung von einem unendlichen Wertebereich auf einen binären Wert (0 oder π) Informationen verloren gehen, welche der Agent benötigt, um einen optimalen Rückgabewert zu erzielen.

Ein PQC, dessen Quantendaten durch die Quanten-Bündelungsoperation extrahiert wurden, erzielte ähnliche Rückgabewerte, wie ein PQC, von welchem lediglich die Erwartungswerte berechnet wurden. Daraus wird gefolgert, dass eine Quanten-Bündelungsoperation keinen zusätzlichen positiven Effekt auf den Lernprozess eines quantenbasierten Agenten hat. Mit dem *hybriden* Modell wurden überwiegend niedrigere Rückgabewerte ermittelt, als mit den anderen Extraktionsverfahren und ist somit weniger gut für quantenbasiertes RL geeignet.

Es wurde gezeigt, dass ein quantenbasierter RL-Agent eine optimale Strategie, ähnlich zu einem klassischen NN, erreichen kann. Allerdings sind in einigen Fällen die Rückgabewerte im Verlauf des Lernprozesses noch *instabil*. Deshalb werden im folgenden Kapitel Möglichkeiten untersucht, um dieser *Instabilität* entgegenzuwirken. Zudem wird getestet, ob sich PQCs-basierte RL-Verfahren auch für den Einsatz auf NISQ-Geräten eignen.

5. Weiterführende experimentelle Untersuchungen

Dieses Kapitel bietet einen Überblick über Erweiterungen der in Kapitel 4 vorgestellten Verfahrensumsetzungen. Im ersten Unterkapitel werden initiale Ergebnisse realer Quantensysteme vorgestellt. Die anderen Unterkapitel führen in mögliche Erweiterungen der bisherigen Verfahren und Konzepte ein.

5.1. Initiale Ergebnisse mit Quantencomputern

Auf aktuell verfügbaren NISQ-Geräten weichen berechnete Ergebnisse oft von simulierten ab, da unbeabsichtigte, elektromagnetische Interaktionen in den Quantengattern einen Quantenzustand manipulieren können, was auch *Rauschen* genannt wird [Pre18]. In Quantenschaltkreisen, welche lediglich auf einer kleinen Anzahl von Qubits arbeiten und eine geringe Tiefe aufweisen, hat das *Rauschen* einen kleineren Einfluss [Du+20].

Die PQCs der Experimente aus Kapitel 4 wurden nicht auf einem Quantencomputer ausgeführt, sondern klassisch simuliert. Da für diese Arbeit lediglich begrenzt Quantenressourcen zur Verfügung standen, konnte kein vollständiger RL-Lernvorgang auf einem Quantencomputer ausgeführt werden. Um trotzdem überprüfen zu können, wie stark der Einfluss des *Rauschens* auf einen PQC mit einer der beschriebenen Strukturen ist, wurden lediglich die ersten 250 Schritte des beschriebenen RL-Algorithmus für die *CartPole*-Umgebung auf einem Quantencomputer durchgeführt. Der verwendete PQC wurde nach der Strukturdefinition von Lockwood und Si aus drei Schichten und einer Quanten-Bündelungsoperation aufgebaut. Die Eingabedaten wurden skaliert-gerichtet kodiert. Dabei wurden die Hyperparametereinstellungen aus Abschnitt 4.2 verwendet und der PQC auf dem Quantensystem *ibmq_belem* von IBM Quantum [IBM21] ausgeführt.

In Abbildung 5.1 ist dargestellt, wie sich der durch die Fehlerfunktion berechnete Wert in diesen ersten Schritten verändert. Der Verlauf des Fehlers für den PQC, welcher auf dem Quantensystem durchgeführt wurde, wird dabei mit einem Quantensimulator von *Qiskit* [Ani+21] in einer gleichen Konfiguration verglichen.

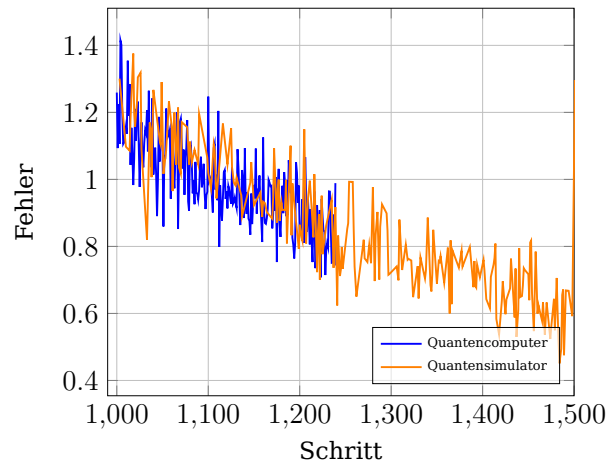


Abbildung 5.1.: Berechnete Fehler in den ersten Schritten des quantenbasierten RL-Algorithmus in der *CartPole*-Umgebung. Dabei wurde ein Quantensimulator von *Qiskit* (orange Kurve) und das Quantensystem *ibmq_belem* von IBM Quantum (blaue Kurve) mit der gleichen Konfiguration verwendet. Die Fehlerberechnung beginnt ab dem Schritt 1000, da zuvor lediglich der Zwischenspeicher mit Werten gefüllt wurde.

Wie beobachtet werden konnte, sank der durch den Quantensimulator berechnete Fehler im Verlauf des Lernprozesses. Die berechneten Werte durch den Quantencomputer weisen einen nahezu identischen Verlauf auf. Diese Ergebnisse sind rudimentär, geben aber Anlass zu der Annahme, dass auch der weitere Lernprozess auf Quantencomputern ähnliche Resultate liefert, wie auf Quantensimulatoren. Damit bieten PQCs eine vielversprechende Alternative zu klassischen NNs, da diese, wie in Abschnitt 4.5 gezeigt, ebenfalls eine optimale Strategie in einer nicht trivialen RL-Umgebung erreichen können.

5.2. Nichtlineare Parametrisierung von PQCs

In den durchgeführten Experimenten aus Kapitel 4 konnte beobachtet werden, dass Rückgabewerte im Verlauf des Lernprozesses zum Teil stark schwanken. Dadurch wird das Ziel des RLs, eine optimale und stabile Strategie eines Agenten zu ermitteln, verfehlt.

Wolf und Mauerer untersuchen eine Möglichkeit, diesen *Instabilitäten* entgegenzuwirken. So wurde in [WM21] festgestellt, dass die Gewichte eines PQCs teilweise Werte außerhalb des Bereichs $[0, 2\pi]$ annehmen. Da es sich dabei um Winkel der parametrisierten Rotationsgatter handelt, repräsentieren Gewichte, welche um ein Vielfaches von 2π auseinander liegen, den gleichen Wert. Es wird vermutet, dass durch diese periodische Eigenschaft der Gewichte auch

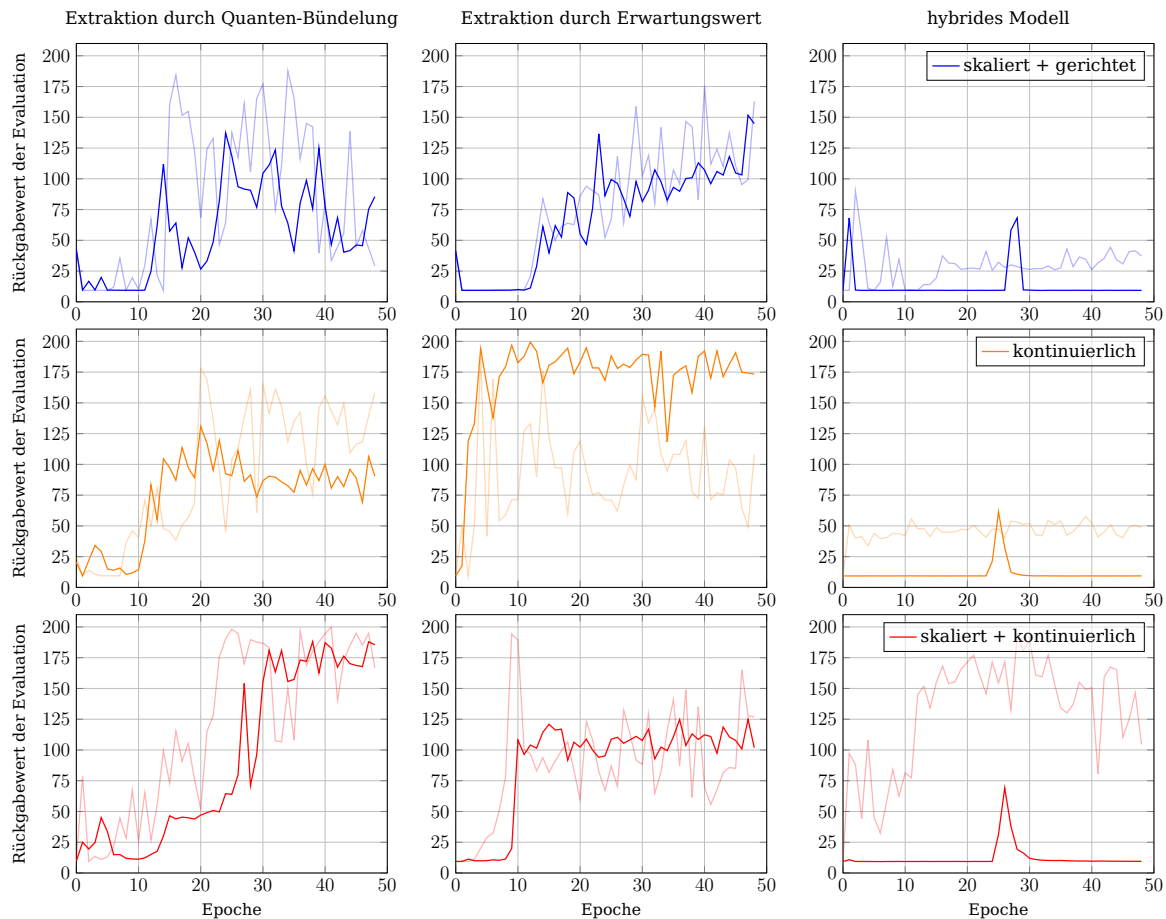


Abbildung 5.2.: Verlauf der Lernprozesse von PQCs nach der Strukturdefinition von Lockwood und Si mit nichtlinearer Parametrisierung der Gewichte und verschiedenen Kodierungs- und Extraktionsstrategien in der *CartPole*-Umgebung. Die Einteilung der Graphen ist, wie in Abbildung 4.3, beschrieben, wobei die transparenten Kurven den Verlauf des Lernprozesses in der jeweiligen Konfiguration ohne Reparametrisierung der Gewichte beschreiben.

die Fehlerfunktion beeinflusst wird und der Lernverlauf dadurch *instabil* wird [WM21].

Um die Periodizität der Gewichte zu verhindern, werden sie mit der *sigmoid*-Funktion $\sigma(x) = \frac{1}{1+e^{-x}}$ in den Bereich $[0, 1]$ verschoben und mit 2π skaliert. Dieser Schritt wird auch bei der Berechnung der partiellen Ableitungen im *gradient descent*-Verfahren beachtet.

Diese Strategie wurde anhand von Algorithmus 1 mit den PQC's aus Kapitel 4 getestet. Dabei wurden Experimente für alle beschriebenen Kodierungs- und Extraktionsverfahren mit den Hyperparametern aus Abschnitt 4.2 in der *CartPole*-Umgebung durchgeführt. Die Gewichte wurden, wie in den Lernprozessen ohne diese Reparametrisierung, mit 0 initialisiert.

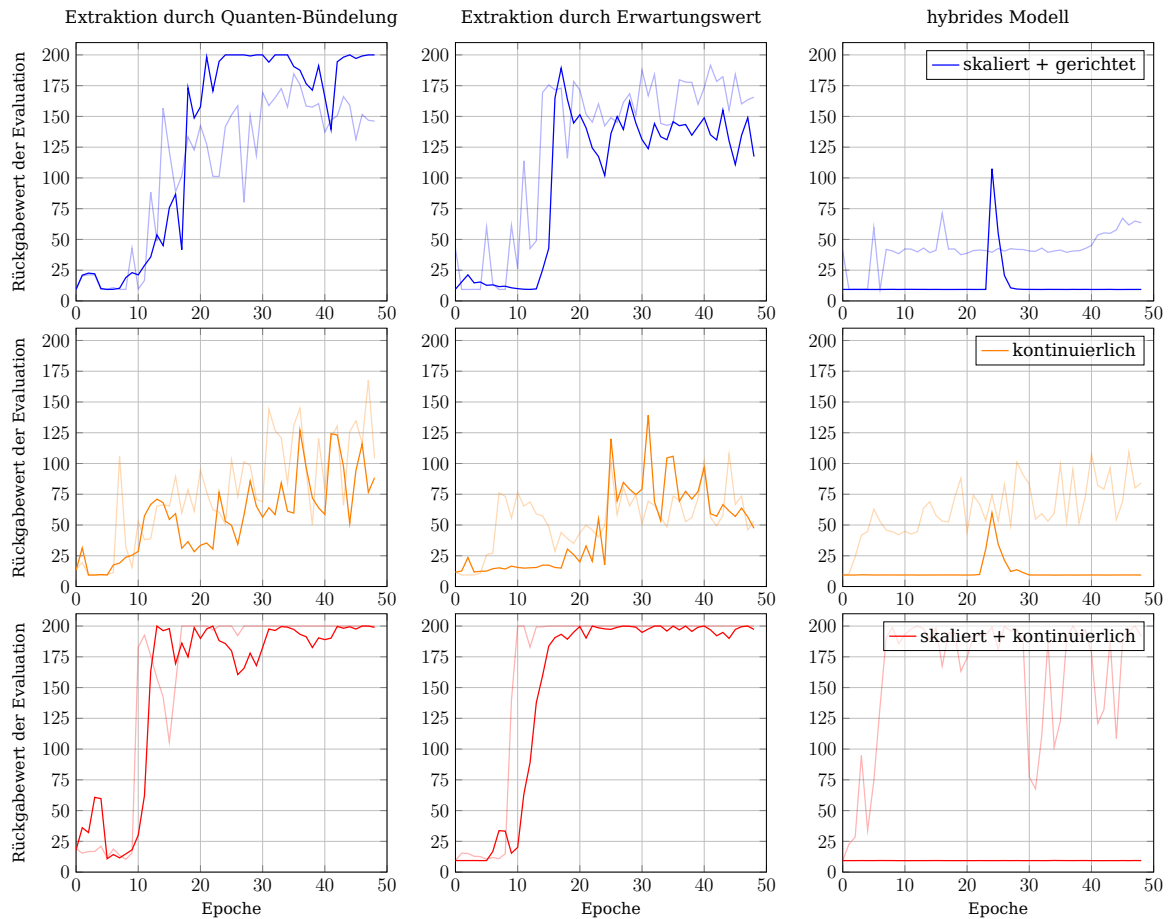


Abbildung 5.3.: Verlauf der Lernprozesse von PQC's nach der Strukturdefinition von Skolik et al. mit nichtlinearer Parametrisierung der Gewichte und verschiedenen Kodierungs- und Extraktionsstrategien in der *CartPole*-Umgebung. Die Einteilung der Graphen ist, wie in Abbildung 4.3, beschrieben, wobei die transparenten Kurven den Verlauf des Lernprozesses in der jeweiligen Konfiguration ohne Reparametrisierung der Gewichte beschreiben.

Die Verläufe der Rückgabewerte für die möglichen Konfigurationen sind für die PQC-Struktur nach Lockwood und Si in Abbildung 5.2 und für die Struktur nach Skolik et al. in Abbildung 5.3 dargestellt. Es wurde festgestellt, dass in drei Konfigurationen, in denen die Rückgabewerte mit linearer Parametrisierung eher *instabil* waren, diese mit nichtlinearer Reparametrisierung nahezu monoton anstiegen (Abbildung 5.2: mittlere Spalte, orange und rote Kurve; Abbildung 5.3: linke Spalte, blaue Kurve). Damit wurde der gewünschte Effekt erzielt und eine stabile Strategie erreicht. In den anderen Konfigurationen konnte ein Vorteil der Reparametrisierung nicht nachgewiesen werden. So wurden in den hybriden Modellen (rechte Spalte) mit Reparametrisierung über den gesamten Lernprozess sehr niedrige Rückgabewerte erzielt. In den Fällen, in denen auch mit linearer Parametrisierung gute, stabile Strategien erreicht wurden, wurden diese auch mit einer Nichtlinearität auf den Gewichten beobachtet (Abbildung 5.2: linke Spalte, rote Kurve; Abbildung 5.3: linke und mittlere Spalte, rote Kurve). In den restlichen getesteten Konfigurationen schwanken die Rückgabewerte sowohl mit linearer, als auch mit nichtlinearer Parametrisierung.

Diese Ergebnisse bestätigen die Resultate aus [WM21]. Darin wurde ebenfalls festgestellt, dass mit einer nichtlinearen Reparametrisierung weiterhin Schwankungen des Rückgabewertes auftreten können. Damit ist die Periodizität der Gewichte eines PQC nicht der hauptsächliche Grund für *Instabilitäten* während des Lernprozesses [WM21].

5.3. Schichtweises Lernen für Quanten-RL

In dem Bereich des quantenbasierten *supervised learnings* wird bei tiefen PQC mit vielen Qubits das Phänomen der sogenannten *barren plateaus* [McC+18] festgestellt. Dabei handelt es sich um ein Problem, in dem Gradienten im Verlauf eines PQC-Lernprozesses sehr klein werden. So kann auf NISQ-Geräten ein Gradient kaum von *Rauschen* unterschieden werden und die Gewichte werden nicht mehr aktualisiert, wodurch der Lernprozess stagniert [Sko+21].

In den vorgestellten Experimenten wurde kein *barren plateau* beobachtet. Dafür war vermutlich die Tiefe des PQC und die Anzahl der verwendeten Qubits zu gering. In komplexeren Anwendungsfällen für RL, die mehr Qubits benötigen, kann jedoch ein Verfahren zur Vermeidung dieses Phänomens notwendig werden.

Skolik et al. stellen dafür das *schichtweise* Lernverfahren vor [Sko+21]. Dabei wird während eines Lernprozesses schrittweise die Tiefe eines Quantenschaltkreises vergrößert. Erreicht ein PQC seine volle Größe, werden lediglich Teile seiner Parameter abwechselnd gelernt. In den letzten Schritten des Lernprozesses wird der PQC mit allen Parametern trainiert.

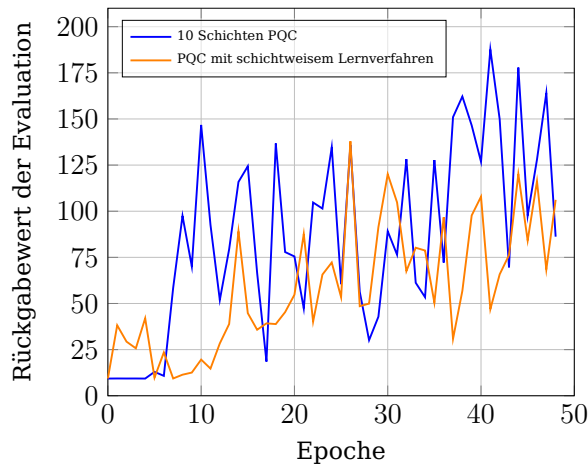


Abbildung 5.4.: Verlauf des Lernprozesses eines PQC mit 10 Schichten jeweils mit dem schichtweisen Lernverfahren und ohne in der *CartPole*-Umgebung. In beiden Lernvorgängen wurden die Eingabedaten kontinuierlich kodiert und die *Q-Werte* über Erwartungswertberechnung extrahiert.

Da in dieser Arbeit nahezu kein Anwendungsfall für das schichtweise Lernverfahren besteht, wurde dieses lediglich experimentell angewandt. Dafür wurde ein PQC nach der in Unterabschnitt 4.5.1 definierten Strukturdefinition von Skolik et al. aufgebaut. Dabei wurden die Eingabedaten kontinuierlich kodiert und die Quantendaten über Berechnung der Erwartungswerte extrahiert. Die Hyperparameter aus Abschnitt 4.2 wurden für den Lernprozess in der *CartPole*-Umgebung verwendet und die Parameter des PQC zufällig gleichverteilt initialisiert. Die Experimente wurden mit *Tensorflow Quantum* [Bro+20] und *Cirq* [Dev21] realisiert.

In dem durchgeführten Experiment des schichtweisen Lernverfahrens hatte der initiale PQC zwei Schichten. Alle 3000 Schritte wurden zwei weitere Schichten hinzugefügt, bis eine Anzahl von zehn erreicht wurde. Anschließend wurden abwechselnd größer werdende Teile der Schichten gelernt und aktualisiert, bis der komplette PQC trainiert wurde. Der Verlauf dieses Lernprozesses ist in Abbildung 5.4 dargestellt. Zum Vergleich wurde ebenfalls ein PQC aus zehn Schichten mit der gleichen Konfiguration ohne das schichtweise Verfahren durchgeführt.

Der Rückgabewert, der in dem schichtweisen Lernverfahren erreicht wurde, stieg vergleichsweise langsam an. Dies kann dadurch begründet werden, dass lediglich ein Teil der Parameter während des Lernvorgangs aktualisiert wird und damit das Erreichen einer guten Strategie eine größere Anzahl an Schritten beansprucht. Es wurde festgestellt, dass keine wesentlich bessere Strategie mit dem schichtweisen Lernverfahren erzielt wurde. Das Phänomen der *barren plateaus* konnte jedoch auch in dem Fall des PQC mit zehn Schichten nicht beobachtet werden. Ein Anwendungsfall für das schichtweise Lernverfahren war

somit nicht gegeben. Deswegen sollte dieses Verfahren in einer zukünftigen Arbeit anhand einer passenderen Umgebung entsprechend evaluiert werden.

5.4. Diskussion

In diesem Kapitel wurde dargestellt, dass aktuelle NISQ-Geräte das Potenzial haben, optimale RL-Strategien zu erreichen. Dabei wird vermutet, dass dies derzeit nur mit einer begrenzten Anzahl an Qubits und einer begrenzten Tiefe mit geringem Fehlereinfluss möglich ist [Che+20].

Um stabiles RL auch für komplexere Probleme und größere Quantenschaltkreise zu ermöglichen, wurde das schichtweise Lernverfahren vorgestellt. Das Potenzial dieses Algorithmus wurde in [Sko+21] beschrieben. Da in dieser Arbeit das Problem der *barren plateaus* allerdings nicht aufgetreten ist, gab es keinen Anwendungsfall für das Verfahren, sodass dieses nicht entsprechend evaluiert werden konnte.

Es wurde außerdem untersucht, ob beobachtete *Instabilitäten* in bisherigen RL-Verfahren durch eine nichtlineare Parametrisierung eines PQC vermieden werden können. Dabei wurde festgestellt, dass eine Periodizität der PQC-Gewichte nicht der hauptsächliche Grund der *Instabilitäten* ist. In einigen Konfigurationen wirkte sich die Reparametrisierung jedoch günstig auf den Lernverlauf aus.

6. Fazit und Erkenntnisse

Dieses abschließende Kapitel fasst die gewonnenen Erkenntnisse zusammen und bietet einen Ausblick auf zukünftige Arbeiten.

6.1. Zusammenfassung

In dieser Arbeit wurde gezeigt, dass ein PQC, ähnlich zu einem klassischen NN, eine optimale Strategie in simplen RL-Umgebungen erreichen kann. Dabei wurde dargestellt, dass neben der Quantenschaltkreisarchitektur, auch die Kodierung der Eingabedaten und die Extraktion der Quantendaten für den Erfolg des RL-Algorithmus entscheidend sind.

So wurde die vergleichsweise beste Strategie in der simplen *CartPole*-Umgebung mit dem PQC nach der Strukturdefinition von Skolik et al. zusammen mit der Kombination aus der skalierten und der kontinuierlichen Kodierung und der Extraktion der Quantendaten durch Erwartungswertberechnung erzielt. Auch durch andere Konfigurationen konnten gute Strategien erzielt werden. Allerdings wurden teilweise *instabile* Verläufe von Lernprozessen beobachtet.

Um diesen *Instabilitäten* entgegenzuwirken, wurde ein Verfahren durchgeführt, in dem die Parameter eines PQCs nichtlinear reparametrisiert werden. Dies erwies sich jedoch nur teilweise als erfolgreich und konnte das Problem der *Instabilitäten* nicht lösen.

Initiale Ergebnisse auf einem Quantencomputer geben Anlass zu der Annahme, dass quantenbasierte RL-Verfahren, die auf wenig Qubits arbeiten, auch auf aktuellen NISQ-Geräten mit geringem Fehlereinfluss durchgeführt werden können.

6.2. Ausblick

Die vorgestellten Verfahren bieten großen Raum für Erweiterungen. In dieser Arbeit konnte auf Grund der verfügbaren Quantenressourcen kein vollständiger RL-Lernprozess auf einem Quantencomputer durchgeführt werden. In einer zukünftigen Arbeit kann somit evaluiert werden, ob, wie vermutet, mit den

beschriebenen Konzepten simple RL-Umgebungen auf realer Hardware gelöst werden können.

Da die betrachteten RL-Anwendungen eher simpel waren, können diese in einem nächsten Schritt auf komplexere Einsatzgebiete ausgeweitet werden. Da dadurch vermutlich auch die Anzahl der Qubits und die Tiefe des verwendeten PQC's ansteigt, sollten Verfahren definiert werden, um den Lernprozess stabiler zu gestalten. Ein erster Ansatz dafür ist die Evaluation des vorgestellten schichtweisen Lernverfahrens aus Abschnitt 5.3. Die vorgestellten Strukturen der PQC's, sowie die Kodierungs und Extraktionsverfahren haben durch die optimal erreichte Strategie in der simplen *Cartole*-Umgebung großes Potenzial aufgezeigt, sollten jedoch auch in einem größeren Kontext evaluiert werden.

A. Anhang: Liste der Quantengatter

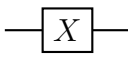
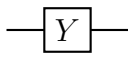
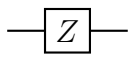
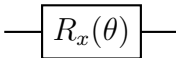
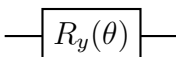
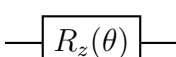
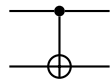
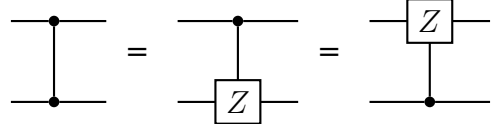
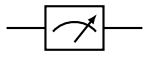
Gatter	Symbol	Matrix
Pauli-X / NOT		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
X-Rotation		$\begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$
Y-Rotation		$\begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$
Z-Rotation		$\begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$
CX / CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
CZ		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
Messoperation		

Tabelle A.1.: Liste der für diese Arbeit relevanten Quantengatter. Schaltsymbole und Schaltkreise wurden mit Quantikz angefertigt [Kay18].

Abbildungsverzeichnis

2.1. Transition in einem MDP	4
2.2. Darstellung eines DQNs	8
2.3. Darstellung eines Quantenzustands in der Bloch-Kugel	12
2.4. Wirkung des <i>CNOT</i> -Gatters	15
2.5. Wirkung der Pauli-Gatter auf einen Quantenzustand	15
2.6. Wirkung des Pauli-X-Gatters auf einen Quantenzustand in der Bloch-Kugel	16
2.7. Wirkung eines Rotationsgatters auf einen Quantenzustand in der Bloch-Kugel	17
3.1. Übersicht über die beteiligten Komponenten bei Quanten-RL	20
3.2. Struktureller Aufbau eines PQC	22
3.3. Darstellung der Quanten-Bündelungsoperation nach Lockwood und Si	26
4.1. Verlauf des klassischen Lernprozesses in den Umgebungen <i>CartPole</i> und <i>Blackjack</i> mit verschiedenen Kodierungsstrategien	34
4.2. PQC-Struktur nach Lockwood und Si	35
4.3. Verlauf der Lernprozesse von PQC nach der Strukturdefinition von Lockwood und Si in der <i>CartPole</i> -Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien	37
4.4. Verlauf der Lernprozesse von PQC nach der Strukturdefinition von Lockwood und Si in der <i>Blackjack</i> -Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien	39
4.5. PQC-Struktur nach Skolik et al.	42
4.6. Verlauf der Lernprozesse von PQC nach der Strukturdefinition von Skolik et al. in der <i>CartPole</i> -Umgebung mit verschiedenen Kodierungs- und Extraktionsstrategien	43
5.1. Berechnete Fehler in den ersten Schritten des quantenbasierten RL-Algorithmus	48
5.2. Verlauf der Lernprozesse von PQC nach der Strukturdefinition von Lockwood und Si mit nichtlinearer Parametrisierung der Gewichte und verschiedenen Kodierungs- und Extraktionsstrategien in der <i>CartPole</i> -Umgebung	49
5.3. Verlauf der Lernprozesse von PQC nach der Strukturdefinition von Skolik et al. mit nichtlinearer Parametrisierung und verschiedenen Kodierungs- und Extraktionsstrategien der Gewichte in der <i>CartPole</i> -Umgebung	50

5.4. Verlauf des Lernprozesses eines PQC's mit 10 Schichten jeweils
mit dem schichtweisen Lernverfahren und ohne in der *CartPole*-
Umgebung. 52

Literatur

- [Ama93] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), S. 185–196.
- [Ani+21] Sajid Anis et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021.
- [Aru+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon und Joseph C. Bardin. "Quantum supremacy using a programmable superconducting processor". In: *Nature* (2019). ISSN: 1476-4687.
- [Asf+20] Abraham Asfaw et al. *Learn Quantum Computation Using Qiskit*. 2020. URL: <http://community.qiskit.org/textbook>.
- [Bab+18] Zunaira Babar, Daryus Chandra, Hung Nguyen, Panagiotis Botsinis, Dimitrios Alanis, Soon Ng und L. Hanzo. "Duality of Quantum and Classical Error Correction Codes: Design Principles & Examples". In: *IEEE Communications Surveys & Tutorials* 21 (2018).
- [Ben+19] Marcello Benedetti, Erika Lloyd, Stefan Sack und Mattia Fiorentini. "Parameterized quantum circuits as machine learning models". In: *Quantum Science and Technology* 4.4 (2019), S. 043001. ISSN: 2058-9565.
- [Bro+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang und Wojciech Zaremba. *OpenAI Gym*. 2016.
- [Bro+20] Michael Broughton et al. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning*. 2020.
- [Che+20] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma und Hsi-Sheng Goan. "Variational quantum circuits for deep reinforcement learning". In: *IEEE Access* 8 (2020), S. 141007–141024.
- [Cro19] Gavin E. Crooks. *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition*. 2019.
- [Dev21] Cirq Developers. *Cirq*. Version v0.11.0. 2021.
- [Du+20] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu und Dacheng Tao. "Expressive power of parametrized quantum circuits". In: *Physical Review Research* 2.3 (2020). ISSN: 2643-1564.
- [GC11] Juan Carlos Garcia-Escartin und Pedro Chamorro-Posada. *Equivalent Quantum Circuits*. 2011.

- [HGS15] Hado van Hasselt, Arthur Guez und David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015.
- [Hom18] Matthias Homeister. *Quantum Computing verstehen - Grundlagen - Anwendungen - Perspektiven*. Berlin Heidelberg New York: Springer-Verlag, 2018. ISBN: 978-3-658-22884-2.
- [IBM21] IBM. *IBM Quantum*. 2021. URL: <https://quantum-computing.ibm.com/>.
- [Kan+17] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow und Jay M. Gambetta. "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets". In: *Nature* 549.7671 (2017), S. 242-246. ISSN: 1476-4687.
- [Kay18] Alastair Kay. "Tutorial on the quantikz package". In: *arXiv preprint arXiv:1809.03842* (2018).
- [Lin91] Long Ji Lin. "Programming Robots Using Reinforcement Learning and Teaching." In: *AAAI*. 1991, S. 781-786.
- [LM20] Gabriel Paludo Licks und Felipe Meneguzzi. *Automated Database Indexing using Model-free Reinforcement Learning*. 2020.
- [LS20] Owen Lockwood und Mei Si. "Reinforcement learning with quantum variational circuit". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Bd. 16. 1. 2020, S. 245-251.
- [LS21] Owen Lockwood und Mei Si. "Playing Atari with Hybrid Quantum-Classical Reinforcement Learning". In: *NeurIPS 2020 Workshop on Pre-registration in Machine Learning*. PMLR. 2021, S. 285-301.
- [LW18] Jin-Guo Liu und Lei Wang. "Differentiable learning of quantum circuit Born machines". In: *Physical Review A* 98.6 (2018). ISSN: 2469-9934.
- [McC+18] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babush und Hartmut Neven. "Barren plateaus in quantum neural network training landscapes". In: *Nature Communications* 9.1 (2018). ISSN: 2041-1723.
- [Mel01] Francisco S Melo. "Convergence of Q-learning: A simple proof". In: *Institute Of Systems and Robotics, Tech. Rep* (2001), S. 1-4.
- [Mir+20] Azalia Mirhoseini et al. *Chip Placement with Deep Reinforcement Learning*. 2020.
- [Mit+18] K. Mitarai, M. Negoro, M. Kitagawa und K. Fujii. "Quantum circuit learning". In: *Phys. Rev. A* 98 (2018), S. 032309.
- [Mni+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra und Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013.

- [Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), S. 529-533.
- [Moh+17] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy und John Martinis. "Commercialize quantum technologies in five years". In: *Nature News* 543.7644 (2017), S. 171.
- [NC10] Michael A. Nielsen und Isaac L. Chuang. *Quantum Computation and Quantum Information - 10th Anniversary Edition*. Cambridge: Cambridge University Press, 2010. ISBN: 978-1-107-00217-3.
- [Opea] OpenAI. *OpenAI Gym Wiki, Cartpole v0*. URL: <https://github.com/openai/gym/wiki/CartPole-v0> (besucht am 22.08.2021).
- [Opeb] OpenAI. *OpenAI Gym, Blackjack*. URL: https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py (besucht am 22.08.2021).
- [Ope+19] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019.
- [Pre18] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (2018), S. 79. ISSN: 2521-327X.
- [SB18] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning - An Introduction*. Cambridge: MIT Press, 2018.
- [Sch+19] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac und Nathan Killoran. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (2019). ISSN: 2469-9934.
- [Sch+20] Maria Schuld, Alex Bocharov, Krysta M. Svore und Nathan Wiebe. "Circuit-centric quantum classifiers". In: *Physical Review A* 101.3 (2020). ISSN: 2469-9934.
- [Shi+13] Yun-Pil Shim, Jianjia Fei, Sangchul Oh, Xuedong Hu und Mark Friesen. *Single-qubit gates in two steps with rotation axes in a single plane*. 2013.
- [Sil+18] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), S. 1140-1144. ISSN: 0036-8075.
- [SJD21] Andrea Skolik, Sofiene Jerbi und Vedran Dunjko. *Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning*. 2021.
- [Sko+21] Andrea Skolik, Jarrod R McClean, Masoud Mohseni, Patrick van der Smagt und Martin Leib. "Layerwise learning for quantum neural networks". In: *Quantum Machine Intelligence* 3.1 (2021), S. 1-11.

- [VGS16] Hado Van Hasselt, Arthur Guez und David Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Bd. 30. 1. 2016.
- [Vin+19] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* (2019), S. 1–5.
- [Wat89] Christopher Watkins. “Learning from Delayed Rewards”. Diss. Cambridge, UK: King’s College, 1989.
- [WM21] Lucas Wolf und Wolfgang Mauerer. *Uncovering Instabilities in Variational Quantum Deep Q-Networks*. Unveröffentlichte Arbeit. 2021.
- [Zhu+19] Daiwei Zhu, Norbert M Linke, Marcello Benedetti, Kevin A Landsman, Nhung H Nguyen, C Huerta Alderete, Alejandro Perdomo-Ortiz, Nathan Korda, A Garfoot, Charles Brecque et al. “Training of quantum circuits on a hybrid quantum computer”. In: *Science advances* 5.10 (2019).

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Bachelorarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Bachelorarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinn-gemäße Zitate als solche gekennzeichnet habe.

Ort, Datum und Unterschrift

Vorgelegt durch:	Maja Franz
Matrikelnummer:	3130348
Studiengang:	Bachelor Informatik
Bearbeitungszeitraum:	1. April 2021 - 31. August 2021
Betreuung:	Prof. Dr. Wolfgang Mauerer
Zweitbegutachtung:	Prof. Dr. Kai Selgrad