Hybride Lernstrategien für dateneffiziente Robotik

Ausarbeitung zum Proseminar „Informatik trifft Maschinenbau“

Zugeteiltes Paper:

Vision intelligence-conditioned reinforcement learning for precision assembly

Jesse Marekwica
Matrikelnummer: 238530
2026-01-11

Inhaltsverzeichnis

1. Einleitung & Motivation

Roboter sind für Unternehmen in der Industrie, insbesondere in der Fertigungstechnik, essenziell geworden. Sie reduzieren die Kosten für Fachkräfte, erhöhen die Zeiteffizienz und sind in Zeiten der Industrie 4.0 weit etabliert. Dies lässt sich am Beispiel der Automobilindustrie veranschaulichen, in der Firmen wie KUKA mit ihren Automatisierungslösungen bereits weit verbreitet sind. Dort werden hochautomatisierte Produktionsketten als Lösung angeboten, die hauptsächlich auf *„Flexible Robotik“* setzen. In diesen übernehmen Roboterarme Aufgaben wie Montage, Schweißen oder Lackierung. [1]. Laut einer Analyse der International Federation of Robotics (IFR) erreichte der weltweite operative Bestand an Industrierobotern zuletzt mit rund einer Million Einheiten einen neuen Höchststand (2023) [2].

Trotz der guten Etablierung weisen diese Lösungen dennoch Nachteile in der Feinmotorik und bei kontaktreichen dynamischen Aufgaben auf. Die oben genannten Lösungen basieren meist auf statischen Regelwerken und gehen von deterministischen Abläufen aus. Tauchen in der Produktionskette kleinere Fehler auf, müssen diese meist durch Eingriffe von Menschen behoben werden, da klassische Robotiksysteme nicht „intelligent“ auf derartige Probleme reagieren. Eine Lösung für die statische Programmierung und die festen Regelwerke, könnte die Informatik liefern. Die vorliegende Arbeit untersucht das Paper von Sichao Liu und Lihui Wang und liefert eine Lösung, die Konzepte aus der Informatik wie Markov-Decision-Processes (MDP), Reinforcement Learning (RL) und neuronale Netze verwendet [3].

In dem Paper wird eine mögliche Lösung für den Lernprozess von Robotern postuliert, bei der durch intelligente visuelle Verarbeitung der räumlichen Umgebung und „Human-in-the-Loop“-Reinforcement Learning Präzisionsarbeit erzielt wird. Dabei kommen neuronale Netze, lernende Algorithmen und intelligente Impedanzcontroller zum Einsatz. Die Methode wird anhand der Montage von Computer-Hardware-Komponenten demonstriert, konkret an RAM-Modulen und einem Kühlsystem auf einem Mainboard. Die Montage solcher kontaktreicher Komponenten mit Schrauben, korrektes Einsetzen und Widerstandserkennung würde einen großen Implementierungsaufwand beim klassisch statischen Programmieren aufweisen. Zusätzlich dazu wäre die Fehlerquote hoch, da bereits Abweichungen im Millimeterbereich zu Schäden an Bauteilen führen könnten (ein RAM-Riegel, der 1 mm daneben liegt, wird nicht passen). Trotz dieser Herausforderungen erreichten die Autoren eine nahezu perfekte Erfolgsquote von über 98 %.

Im Folgenden wird tiefer auf die Informatik des Papers eingegangen. Zunächst wird das eigentliche Problem als Markov-Decision-Process (MDP) modelliert. Dies ist nötig, da Reinforcement-Learning-Algorithmen auf solche MDPs operieren. Anschließend wird die Architektur, das Actor-Critic-Modell, kurz erklärt und zur Umsetzung des lernenden Algorithmus übergeleitet. Der im Paper vorgestellte RLPD (Reinforcement Learning with Prior Data) bietet eine dateneffiziente Lösung, die von Liu und Wang aufgegriffen und umgesetzt wird. Dabei wird die Umsetzung im Folgenden genauer beleuchtet und unter Bezugnahme des Papers von Ball et al. analysiert. [4]. Abgeschlossen wird die Ausarbeitung mit einer Diskussion und Evaluation der gesamten Umsetzung der informatikbezogenen Ansätze sowie einer Kritikwürdigung.

2. Systemarchitektur & Problemmodellierung

2.1. Markov Decision Process (MDP)

In dem Paper wird das Montageproblem als Markov-Decision-Process (MDP) modelliert. Dabei handelt es sich um eine formale, mathematische Definition eines Entscheidungsproblems, das hier zur Modellierung einer Montage verwendet wird. Ein MDP lässt sich als gerichteter Graph modellieren, wobei die Knoten als Zustände und die Kanten als durch Handlungen ausgelösten Zustandsübergänge (Tran-

sitionen) darstellen werden. Zum besseren Verständnis betrachten wir ein bekanntes Beispiel aus der Vorlesung von David Silver (DeepMind/UCL): Der beschriebene Graph modelliert den Studienalltag [5].

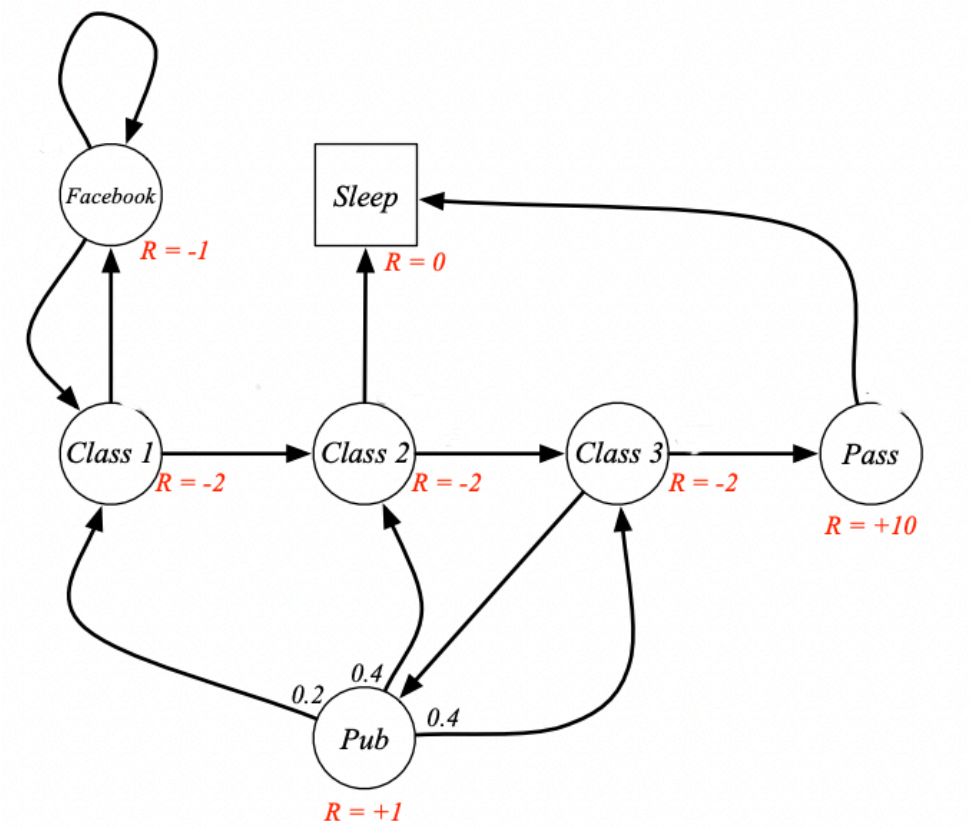

Abbildung 1: Optimal Action-Value Function for Student MDP - David Silver [5].

Um den Kurs zu bestehen, müssen Studierende alle drei „Class"-Zustände erfolgreich durchlaufen. Die Kreise repräsentieren hierbei die Zustände, wobei „Class 1" als Startzustand dient. In diesem Zustand kann der Student eine Handlung wählen: „Facebook" oder „Study". Wählt er „Study", folgt eine Transition, die mit einer bestimmten Wahrscheinlichkeit behaftet ist (in diesem Fall implizit 100%). Der Reward (R) ist der Wert, den der Student für das Ausführen einer Aktion erhält. In „Class 1" kostet beispielsweise jeder Zeitschritt $R = -2$ (negativer Reward/Bestrafung). Sobald sich der Agent in „Class 3" befindet, kann er sich für „Pub" entscheiden. Von dort aus gelangt er mit unterschiedlichen Wahrscheinlichkeiten in die Klassen 1–3 [Abbildung 1]. Der Zustand „Sleep" ist ein terminierender Zustand, der das Ende markiert.

Das Ziel (die Optimierung) in einem MDP besteht darin, eine Strategie (Policy) zu finden, mit der sich die Summe der erwarteten Rewards maximieren lässt.

Um dies auf das Paper zu übertragen, nutzen wir dessen formale Definition eines MDPs [3]:

$$\boldsymbol{M = \{S, A, P, p, R, \gamma\}}$$

- $\boldsymbol{S}$ **(State Space)**: Beschreibt die Menge aller Zustände. In unserem Beispiel sind alle dargestellten Knoten S und konkrete Knoten $s \in S$. Dabei wäre „Class 1" ein Zustand s.
- $\boldsymbol{A}$ **(Action Space)**: Beschreibt die Menge aller verfügbaren Aktionen. Im Zustand „Class 1" wären das „Facebook" oder „Study".
- $\boldsymbol{P}$ **und** $\boldsymbol{p}$ **(Wahrscheinlichkeiten)**: Das große P beschreibt die **Startverteilung** (Initial State Distribution). Da es praktisch unendlich viele Startkonfigurationen geben kann, gibt P an, wie wahrscheinlich es ist, in einem bestimmten Zustand s_0 zu starten. Im Beispiel wäre „Class 1" unser Startzustand mit

$P(Class1) = 1.0$. Das kleine p beschreibt die **Systemdynamik**. Es gibt die Erfolgswahrscheinlichkeit einer gewählten Aktion an. Im Beispiel: Wenn man „Pub“ in „Class 3“ wählt, ist $p = 0.4$ für den Übergang zu „Class 2“. Diese Definition unterscheidet sich von der in David Silvers Vorlesung. Um auf das weitere Vorgehen aufzubauen, wird die Definition des Papers verwendet [3].

- $\boldsymbol{R}$ **(Reward Function)**: Bewertet die Qualität der Entscheidung. Im Beispiel erhält man $R = -2$ für „Study“. Dieser Wert ist der entscheidende Parameter, an dem die Optimierung gemessen wird.
- $\boldsymbol{\gamma}$ **(Discount Factor)**: Dies ist der Gewichtungsfaktor $(0 \leq \gamma < 1)$, der bestimmt, wie wichtig zukünftige Belohnungen im Vergleich zu sofortigen sind. Ein γ nahe 0 beschreibt eine „kurzsichtige“ Strategie (nur der nächste Reward zählt), während ein γ nahe 1 die Strategie „weitsichtig“ macht (langfristige Ziele wie „Pass“ werden wichtiger als kurzes Facebook-Vergnügen). In der Vorlesung von David Silver wurden beide Beispiele einmal gezeigt: Das $(\gamma = 0)$-Beispiel verfiel in eine Facebook-Schleife, während das $(\gamma = 1)$-Beispiel in „Pass“ überging.

Nachdem ein MDP formuliert wurde, muss es gelöst werden. Bei einem Optimierungs-/Maximierungsproblem beschreibt die Lösung den höchstmöglichen erreichbaren Erfolgswert, in unserem Fall den Reward R. Die gängigste Vorgehensweise beim Lösen von MDPs ist das Aufstellen einer Strategie, einer sogenannten **Policy** $(\boldsymbol{\pi})$. Nach einem bekannten Theorem gilt, dass es für jeden MDP eine optimale Policy (π) gibt, die besser oder gleich allen anderen Policies ist, also $\pi_* \geq \pi, \forall \pi$ [5]. Auf unser Beispiel bezogen, wird eine Policy (π_*) gesucht, die den höchstmöglichen Wert für

$$E\left[\sum_{t=0}^{h} \gamma^t R(s_t, a_t)\right]$$

findet. Da die Übergänge zwischen den Zuständen stochastisch sind und es dementsprechend Unsicherheiten gibt, bilden wir den Erwartungswert E aller möglichen Verläufe. Das γ dient hier wie bereits angedeutet als Diskontierungsfaktor, der je nach Wahl über die Zeitschritte t die Gewichtung immer kleiner bis nahe 0 einfließen lässt. Die Rewardfunktion mit $R(s_t, a_t)$ definiert für den derzeitigen Zustand s_t unter der Handlung a_t den Reward R. Dieser wird dann bis zum angegebenen Horizont h akkumuliert.

Für das Beispiel des Studentenlebens ist eine optimale Policy (π_*) einfach zu bestimmen. Mithilfe der **optimalen Action-Value-Function** $Q_*(s, a)$ können wir den maximalen akkumulierten Reward bestimmen, den man erwarten kann, wenn man sich im Zustand s für die Aktion a entscheidet und anschließend optimal weitermacht. Ein Blick auf [Abbildung 1] verdeutlicht dies am Zustand „Class 3“:

- Der Wert für Lernen ist $Q_*(\text{Class 3}, \text{Study}) = 10$.
- Der Wert für die Kneipe ist $Q_*(\text{Class 3}, \text{Pub}) = 8.4$.

Da $10 > 8.4$ ist, ist die Handlung „Study“ hier die optimale Wahl. Die Policy (π_*) wählt also stets gierig („greedy“) das Maximum über Q_*. Ein bekanntes Vorgehen, um Q_* in allen Zuständen zu bestimmen, ist mit dem Ende zu beginnen und in Vorgängerzuständen zurückzuschauen [5]. Dem vereinfacht ausgedrückten Bellmann'schen Optimalitätsprinzip zufolge ist nämlich jede Teilpolitik (Subpolicy) π_{sub} einer optimalen Politik (Policy) π_*, ebenfalls optimal [5]. Zur Veranschaulichung: Wenn die kürzeste ICE-Strecke von Berlin nach München über Leipzig führt, dann ist die kürzeste Strecke von Leipzig nach München dieselbe. Aufgrund dieser Eigenschaft können wir das Problem vom Ende ausgehend lösen, da die kürzeste Strecke von München nach München trivial und bekannt ist. Man beginnt in den terminierenden Zuständen (z.B. „Sleep“ mit Wert 0) und berechnet die Werte der davorliegenden Zustände rekursiv rückwärts. Dabei gilt für jeden Schritt: Der Wert eines Zustands ist der sofortige Reward plus der (bereits berechnete) maximale Wert des Nachfolgezustands. So propagieren sich die korrekten Q_*-Werte von hinten nach vorne durch den gesamten Graphen, bis für alle Zustände $s \in S$

die optimale Entscheidung feststeht. Daraus leiten wir dann unsere optimale Policy $\pi_*(a|s)$ ab, also die Wahrscheinlichkeit, mit der wir Handlung a in Zustand s wählen.

2.2. MDP zur Modellierung eines Montageproblems

Da nun eine gewisse Grundlage für Verständnis von MDPs geschaffen wurde, wird nun das einfache Beispiel von David Silver mit dem des Montageproblems im Paper verglichen [3]. Es existieren nämlich starke Unterschiede in der Bestimmung einer optimalen Policy (π_*). Die Autoren modellieren ihr Montageproblem als ein Markov-Decision-Process (MDP) mit $M = \{S, A, P, p, R, \gamma\}$. Während im Prinzip die Elemente des MDPs konzeptionell gleich bleiben, unterscheiden sie sich in der Umsetzung.

- $\boldsymbol{S}$ **(State Space)**: Die Zustandsmenge wird in dem Paper definiert als **State Observation Space**, also der gesamte beobachtbare Bereich der Montage über die Kameras und den Zustand des Armes. Die Kameras nehmen Bilder auf, die über ein ResNet-10 (ein Convolutional Neuronal Network [6]) in Vektoren übersetzt werden. Dadurch wird die Dateneffizienz gesteigert, da ResNet-10 die strukturellen Merkmale abstrahiert und deutlich komprimierter, ohne relevanten Informationsverlust aufbereitet.
- $\boldsymbol{A}$ **(Action Space)**: Die Handlungsmöglichkeiten werden über die kartesische Koordinatenposition des Roboterarms und den Griffzustand definiert.
- $\boldsymbol{P}$ **und** $\boldsymbol{p}$ **(Wahrscheinlichkeiten)**: Da der Roboterarm kaum immer von der gleichen Stelle aus mit der Montage beginnt und das Mainboard nicht immer im exakten Millimeterbereich gleich liegt, muss eine Wahrscheinlichkeitsverteilung $P(s)$ definiert werden, die unterschiedliche Start-Zustände modelliert. Zudem stellt p nicht mehr einfache Wahrscheinlichkeiten an einer Transition dar, sondern repräsentiert die gesamte Dynamik des Systems. Unabhängig davon, wie präzise der Roboterarm ist, wird er mit einer gewissen physikalischen Schwankung von der angegebenen Trajektorie abweichen. Aufgrund der physikalischen Komplexität ist p uns nicht bekannt, sondern wird durch Reinforcement Learning approximiert.
- $\boldsymbol{R}$ **(Reward Function)**: Die Autoren haben für R ein binäres Klassifizierungssystem gewählt, das anhand zuvor tranierten Demos beurteilt, ob eine Montage erfolgreich war, oder fehlgeschlagen ist. Diese wird als Reward/Binary-Classifier bezeichnet.
- γ **(Discount Factor)**: Erfüllt den exakt selben Zweck wie im vorher aufgeführten Beispiel [Abbildung 1].

Bei näherer Betrachtung der Komponenten S und p zeigen sich die zentralen Herausforderungen dieses Ansatzes: die **stochastische Systemdynamik** (Nicht-Determinismus) und die **enorme Dimensionalität** des Zustandsraums. Anhand dieser Grundlage ist es nicht möglich, eine *„wahre“* optimale Policy (π_*) zu finden. Stattdessen wird diese durch Reinforcement approximiert. Um sich einer optimalen Policy überhaupt annähern zu können, muss eine **parametrisierte Funktionsapproximation** erfolgen. Das Paper nutzt hierfür einen sogenannten Soft-Actor-Critic-Ansatz (SAC) mit signifikanten Designerweiterungen, der zum gewählten **Reinforcement Learning with Prior Data (RLPD)** führt. Dabei wird das komplexe Optimierungsproblem auf zwei neuronale Netze aufgeteilt, die gegenseitig voneinander lernen. Im Folgenden wird das Zusammenspiel von Actor und Critic (SAC) veranschaulicht [7].

2.3. Actor-Critic-Modell

Die analytische Lösung eines MDP basiert [5] auf der **Bellman-Gleichung**. Diese besagt, dass der Wert eines Zustands genau dem Erwartungswert aus dem Reward und dem Wert des Folgezustands entspricht: $Q_*(s,a) = E[R(s,a) + \gamma \max_{a'} Q^*(s',a')]$. Im klassischen Fall wird diese Gleichung iterativ gelöst, bis die Werte konvergieren. Für das in dem Paper beschriebene Montageproblem ist dies aufgrund der hochdimensionalen Bilddaten und Systemdynamik jedoch nicht möglich. Daher müssen wir die analytische Funktion Q^* durch ein neuronales Netz Q_φ approximieren. Ein künstliches neuronales Netz (KNN) ist ein Modell des maschinellen Lernens, das in seiner Funktionsweise grob dem menschlichen Gehirn nachempfunden ist. Es dient dazu, Muster in Daten zu erkennen und Entscheidungen zu treffen.

Der Einfachheit halber nehmen wir Folgendes an: Ein neuronales Netz Q_φ lernt eine Funktion $f(X)$ indem es einen Eingabevektor $X = (x_1, x_2, x_3, ...)$ einer Reaktion Y zuordnet. Ein neuronales Netz besteht dabei aus unterschiedlichen Schichten: Einem Input Layer für X, mehrschichtigen versteckten Layern, die hauptsächlich für das Lernen von $f(X)$ zuständig sind, und einem Output Layer, in dem Y ausgegeben wird [8].

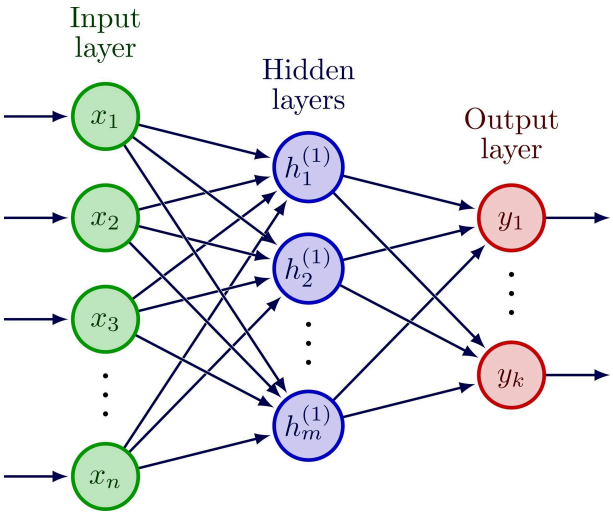

Abbildung 2: Neuronales Netz mit drei Ebenen - [9]

Die Autoren des Papers nutzen zwei neuronale Netzwerke beim SAC: Eines für den Actor und eines für den Critic. Im Folgenden werden die einzelnen Netzwerke näher erläutert und ihre Wechselwirkung analysiert.

2.3.1. Critic

Der Critic bewertet die aktuelle Einschätzung des Actors und vergleicht sie mit der tatsächlichen Situation und Zukunft. Dazu wurde folgende Loss-Funktion $\mathcal{L}_Q$ aufgestellt:

$$\mathcal{L}_Q(\varphi) = E_{s,a,s'}\left[\left(Q_\varphi(s,a) - \left(R(s,a) + \gamma E_{a'\sim\pi_\theta}\left[Q_{\overline{\varphi}}(s',a')\right]\right)\right)^2\right]$$

Die Funktion $\mathcal{L}_Q$ bestimmt die Fehlerquote der Parameter φ im neuronalen Netzwerk Q_φ. Dazu wird anhand einer Stichprobe (einem Batch aus dem Replay Buffer) über den **mittleren quadratischen Fehler** (Mean Squared Error) zwischen zwei Werten die Differenz ermittelt. Dafür wird einmal der Wert der eigenen Vorhersage $Q_\varphi(s,a)$, also die Einschätzung des Netzwerks über die Handlung a in Zustand s und zum anderen dem **Bellman-Zielwert** (Target) berechnet. Der Zielwert setzt sich aus dem tatsächlich erhaltenen Reward $R(s,a)$ und der diskontierten Prognose, also $\gamma \times Q_{\overline{\varphi}}(s',a')$, des **Target-Networks** $Q_{\overline{\varphi}}$ für den Folgezustand zusammen. Wichtig zu beachten ist, dass es sich beim Target-Network um ein anderes Netzwerk als Q_φ handelt. Das liegt daran, dass eine sofortige Aktualisierung der Werte zu einer gleichzeitigen Veränderung des Netzwerks und des Ziels führen würde. Deshalb wird eine Kopie $Q_{\overline{\varphi}}$ erstellt, wodurch das Training stabilisiert wird. Dadurch wird verhindert, dass das Ziel („Moving Target“) zu stark schwankt, indem die Parameter $\overline{\varphi}$ nicht direkt optimiert werden, sondern den Hauptparametern φ folgen und über einen gleitenden Durchschnitt (Soft Update) aktualisiert werden [7].

Der Critic dient als Leiter für den Actor, der die Policy (φ_θ) „*kritisiert*“.

2.3.2. Actor

Der Actor steuert den Roboter, da er die tatsächliche Policy (φ_θ) definiert, die das Montageproblem löst. Dazu wurde folgende Loss-Funktion $\mathcal{L}_\pi$ aufgestellt:

$$\mathcal{L}_\pi(\theta) = -E_s\left[E_{a\sim\pi_\theta(\theta)}\left[Q_\varphi(s,a)\right] + \tau\Phi(\pi_\theta(.\mid s))\right]$$

Die Funktion $\mathcal{L}_\pi$ setzt sich aus zwei Zielen zusammen: Einerseits gierig (Exploitation) zu sein und andererseits neugierig (Exploration). Die Subtraktion zu Beginn der Funktion wandelt das Maximierungsproblem in ein Minimierungsproblem um, da Computer besser darin sind, Fehler zu minimieren. Die Maximierung des Rewards, also die Suche nach dem globalen Optimum einer Funktion $f(x)$ ist für uns gleichbedeutend mit der Suche nach dem globalen Minimum $-f(x)$, jedoch einfacher für Computer umzusetzen. Der Teil, der die Gier des Actors steuert, ist in diesem Teil enthalten: $E_{a\sim\pi_\theta(\theta)}\left[Q_\varphi(s,a)\right]$. Dabei hat das $a \sim \pi_\theta(\theta)$ eine relativ wichtige Bedeutung. Es ist mathematisch nicht möglich, Rückpropagierung (Backpropagation) in einem neuronalen Netz zu durchzuführen, wenn Stochastik zugrunde liegt. Denn aus einem Sample a können keine Rückschlüsse auf die Zufallsverteilung gezogen und keine plausiblen Anpassungen am neuronalen Netz vorgenommen werden. Die Autoren bedienen sich hier des Tricks der **Reparametrisierung (Reparameterization)**, indem der Zufall grob gesagt in ein Standard-Rauschen ϵ ausgelagert wird, wodurch der stochastische Sample differenzierbar wird. [10]. Mithilfe dieses Tricks kann der Actor Q_φ lernen, sich dem Critic Q_π anzupassen. Der zweite Teil der Funktion $\tau\Phi(\pi_\theta(.\mid s))$ ist für die Exploration zuständig. Damit wird vorgebeugt, dass sich der Actor nicht zu früh in einer approximierten Lösung festsetzt, sondern nach anderen, eventuell besseren sucht. Der Hyperparameter τ (Temperatur) steuert dabei das Gleichgewicht: Ein hohes τ fördert die Exploration, während ein niedriges τ die Policy stärker auf die Nutzung des besten bekannten Weges (Exploitation) fokussiert. Die Entropie Φ gibt die Standardabweichung σ vor, also wie „experimentierfreudig" der Actor ist. Diese Exploration wenden wir auf unseren Zustand s unter der Berücksichtigung aller möglichen Handlungen a (hier gekennzeichnet durch $(.|s)$, innerhalb der Policy (π_θ)) an.

Der Actor leitet den Roboter unter der Berücksichtigung des Critics und eigenen *„Neugierfaktor"*.

3. Effizientes Online Reinforcement Learning mit offline Daten - RLPD

Deep Reinforcement Learning (RL) konnte in vielen Feldern bereits Erfolge verzeichnen wie in Atari oder Go [11], [12]. In diesen Beispielen werden hohe Erfolge durch Reinforcement Learning und viele Online Interaktionen erzielt, was durch Simulationen gut umsetzbar ist. Leider sind Probleme, wie das Montageproblem von Liu & Wang, in der Realität oft deutlich komplexer, als in einer Simulation [3]. Rewards sind meist abstrahiert, während sie in der Realität schwer greifbar und hochdimensional sind. Die Autoren des Papers Ball et al. postulieren den Ansatz von **RLPD** [4]. Dieser unterscheidet sich von Deep RL und SAC + Offline Daten. Liu & Wang stützen sich stark mit ihrer Architektur auf den Ansatz aus dem Paper von Ball et al., wobei in RLPD drei erweiterte Designentscheidungen den Ansatz prägen. Im Folgenden werden wir die Motivation hinter diesen Erweiterungen anschauen und deren Umsetzung von Liu & Wang.

3.1. Hybrides Buffer-System und Replay Ratio

Beim klassischen Deep Reinforcement Learning dauert es in der Anfangsphase besonders lange, bis der Algorithmus eine Richtung ermittelt hat, in der sich die optimale Policy befindet. Dieser Prozess der Richtungfindung kann jedoch minimiert werden, indem beim Deep RL im Voraus bereits suboptimale Policies oder menschliche Demonstrationen die Richtung vorgeben. Mit diesen Demonstrationen gehen jedoch Probleme einher. Während der RL-Algorithmus lernt, werden in kurzer Zeit bereits eine Vielzahl von Durchläufen in dem Replay-Buffer geladen. Das ist der Speicher, in dem alle Epochen gespeichert werden, und die geladenen Daten überwiegen schnell die selbst generierten. Ball et al. erkannten dies ebenfalls und entwarfen basierend auf Ross & Bagnell (2012) eine symmetrische Replay-Buffer-Architektur [13]. Dabei werden statt nur einem Replay-Buffer, zwei Buffer angelegt: Einer speichert und nimmt die Online-Daten des RL-Algorithmus auf, der andere dient als Offline-Buffer und speichert

beispielsweise die eben genannten menschlichen Demonstrationen. Das hat vor allem den Vorteil, dass die vorgefertigten Richtungsgeber nicht unter der schnell wachsenden Menge an RL-Abläufen untergehen. Damit beim Samplen aber auch die Gewichtung erhalten bleibt, wird aus beiden Buffern jeweils genau 50% der Daten entnommen. Diese Designentscheidung erweist sich in der Evaluation als besonders effektiv, auch wenn sie alleine nicht ausreicht [4].

Dieses Vorgehen des symmetrischen Samplen wird auch von Liu & Wang verwendet, wie in [Abbildung 3] zu erkennen ist.

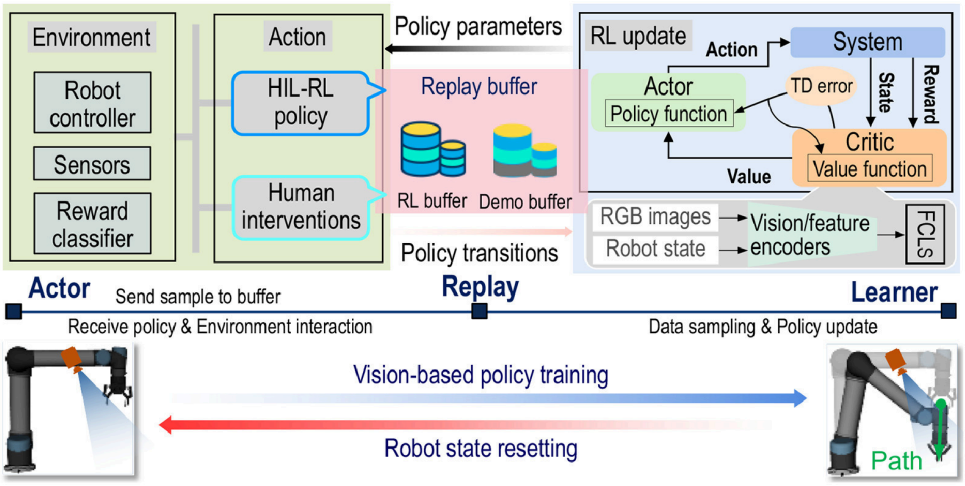

Abbildung 3: Actor-Learner Architektur mit zwei Buffern - [3]

Näher wird auch erklärt, dass es sich um menschliche Demonstrationen handelt, die im Voraus erstellt wurden. Für jede Montageaufgabe (CPU-Kühlkörper, RAM, Lüfter) wurden jeweils 30 erfolgreiche Trajektorien, also vollständige Bewegungsabläufe einer Montage, verwendet. „Erfolgreich“ wird hierbei durch zwei Kriterien definiert. Zum einen darf sich das zu montierende Objekt nicht mehr als 0.1mm von der Zielposition entfernt befinden. Andererseits muss, sofern Ersteres erfüllt ist, der zuvor tranierten **Binary Classifier**, die Montage ebenfalls mit einer Wahrscheinlichkeit von min. 97% als „1“, also erfolgreich bewerten. Hier wurde entschieden, hochqualitative menschliche Abläufe als Offline-Daten zu verwenden. Ebenfalls werden beide Buffer gleich gewichtet, indem ein Batch gleiche Menge an Daten aus beiden Buffern enthält. Die genauen Vor- und Nachteile variieren stark je nach zu optimierendem Problem. Unter der Voraussetzung, dass Liu & Wang das Modell in der Realität trainieren, ist dieses Vorgehen, wenn auch aufwendig, durchaus sinnvoll und erfolgreich, wie der Evaluation der Versuchsabläufe zu entnehmen ist.

3.2. Layer Normalization

Eine fundamentale Schwäche von Deep Reinforcement Learning ist der Umgang mit unbekannten Daten, die sogenannten **Out-of-Distribution (OOD) Daten**. Bei Actor-Critic-Architekturen sind diese Daten während des Lernprozesses von RL meist nicht definiert. Dabei passiert es, dass der Critic die neuen eingehenden Daten stark „überschätzt“, da er diese mit den Offline-Daten aus dem Demo-Buffer in Beziehung setzt. Die Folge sind Instabilitäten im Training und eine **Divergierung des Critics (Overestimation)**, während er versucht, den immer größer werdenden Werten zu folgen [14]. Die Lösung von Ball et al. nutzt Normalisierung der Werte innerhalb des neuronalen Netzes, damit diese Werte innerhalb eines Zahlenbereichs (in der Regel einem Mittelwert $\mu = 0$ und Standardabweichung $\sigma = 1$) bleiben. Dabei ist es möglich, dass der RL-Ansatz mit Layer Normalization trotzdem Neues lernen kann, ohne zu stark von den Demo-Daten limitiert zu sein [15]. Mathematisch wurde das wie folgt bewiesen und umgesetzt: $\|Q(s,a)\| \leq \|w\|$ [4]. Dies bedeutet, dass der vorhergesagte Q-Wert niemals größer werden kann als die Norm der Netzwerkgewichte w.

Die Umsetzung von Liu & Wang der Layer Normalization ist anzunehmen, da eine Umsetzung von RLPD ansonsten nicht möglich wäre. Wie die Gleichung der Q-Loss-Funktion zeigt, nutzen die Autoren die standardmäßige Soft-Bellman-Optimierung des SAC-Algorithmus:

$$\mathcal{L}_Q(\varphi) = E_{s,a,s'}\left[\left(Q_\varphi(s,a) - \left(R(s,a) + \gamma E_{a'\sim\pi_\theta}\left[Q_{\overline{\varphi}}(s',a')\right]\right)\right)^2\right]$$

Mit dieser Funktion wird lediglich die Differenz zwischen der Vorhersage Q_φ und dem Zielwert Q_π minimiert. Sie beinhaltet jedoch keinen Mechanismus, der das Netzwerk vor der erwähnten **Overestimation** bei unbekannten Daten schützt. Die Stabilisierung muss daher strukturell innerhalb der Funktion $Q_\varphi(s,a)$ selbst erfolgen. Während Ball et al. hierfür explizit **Layer Normalization** vor der letzten Ausgabeschicht vorschreiben ($\|Q\| \leq \|w\|$), lassen Liu und Wang die genaue Innenarchitektur ihres Critics im Paper unerwähnt. Es ist möglich, dass die Autoren sich hier implizit auf die Beschaffenheit des **Reward Classifiers** verlassen, der den Reward hart auf das Intervall $[0,1]$ begrenzt. Unter der Annahme, dass die Autoren die RLPD-Methodik vollständig adaptiert haben, ist der Einsatz dieser Normalisierungsschichten zwingend erforderlich, da der Critic bei der Verarbeitung der Daten aus den zwei Buffern sonst zur Divergenz neigen würde.

3.3. Sample Efficient RL

Die Designentscheidung für einen zweiten Offline-Buffer führt zu einem aufwendigeren Sampling. Um den Aufwand zu minimieren und die Effizienz beizubehalten, wird die Geschwindigkeit des Lernprozesses modifiziert. Eine Möglichkeit ist es, den UTP-Wert zu erhöhen. Dieser gibt an, wie viele Lernschritte (der Critic passt die Gewichte an) pro Arbeitsschritt (der Actor führt eine Handlung aus) durchgeführt werden. Dabei besteht jedoch die Gefahr, dass der RL-Algorithmus in **Überanpassung (Overfitting)** verfällt. Dieser beschreibt einen Zustand, in dem sich ein Modell zu stark an einem lokalen Optimum angepasst hat und irrelevante Faktoren berücksichtigt [16]. Zum besseren Verständnis wird oft der Unterschied zwischen „Verstehen“ und „Auswendig lernen“ anhand eines veranschaulichten Beispiels aufgezeigt. Wenn ein RL-Algorithmus zu exakt gelernt hat, eine Aufgabe zu lösen, sorgt der Zustand der Überanpassung dafür, dass die spezifisch gelernte Aufgabe mit einer hohen Genauigkeit gelöst wird, bei leichten Änderungen jedoch bereits scheitert. Als Lösung dafür nennen Ball et. at. einige Möglichkeiten, wobei sie sich für die Methoden mit **Random Ensemble Distillation** und **Random Shift Augmentations** entscheiden. Ersteres beschreibt die Verwendung mehrerer Critics, die sich gegenseitig vor Divergenz schützen. Letzteres wird genutzt, da Ball et al. ebenfalls Bilder zum Training verwendet. Dabei werden die Bilder um wenige Pixel zufällig verschoben, wodurch eine Art *„Wackeln“* imitiert wird. Das ist besonders nützlich, da der RL-Algorithmus so lernt den RAM-Sockel wirklich als solchen zu erkennen und sich nicht auf statische Pixelpositionen versteift (Overfitting).

Indizien für eine Overfitting-Prävention sind im Paper von Liu & Wang vor allem bei der Extraktion von Daten aus Bildern zu finden. Dazu nutzen sie ein vortrainiertes ResNet-10-Netzwerk zur Bilderkennung, das bereits auf Millionen von Daten trainiert wurde. Dadurch ist davon auszugehen, dass dieses Netzwerk gegen Overfitting resistent ist. ResNet-10 ist eine vereinfachte Version des bekannten ResNets, das Bilder auf relevante Daten extrahiert, mit denen viele RL-Algorithmen arbeiten können [6]. In dem Paper werden die Kamerabilder des Roboters ins ResNet gesetzt, wodurch dem Problem der geringen Datenmenge von 1400 Bildern, vorgebeugt wird. Diese Daten werden dann in den Binary Classifier eingespeist, der die Abläufe des Actors bewertet und die Ergebnisse an die Learner-Architektur liefert. Die Actor-Learner-Architektur ist in Abbildung 4 gut zu erkennen.

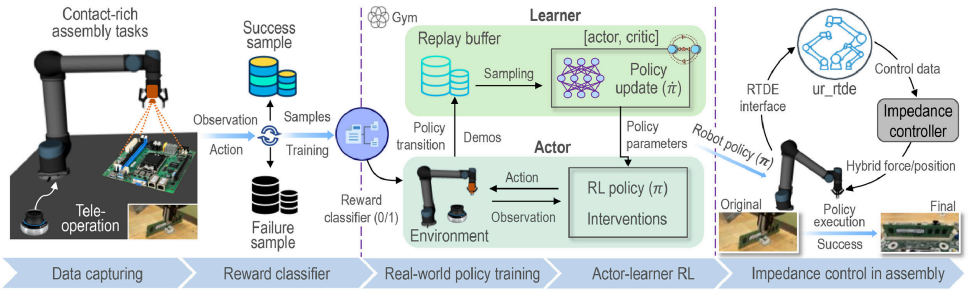

Abbildung 4: Actor-Learner Architektur und Binary/Reward Classifier - [3]

Zusätzlich wird die Methode **„Image Cropping“** verwendet, um Bilder auf eine Größe von 128×128 Pixeln zu verkleinern und sie auf relevante Bereiche zu beschränken. Eine direkte Nutzung von **Random Shift Augmentations**, **Random Ensemble Distillation** oder anderer im Paper von Ball et al. genannter Methoden, die als essenzielle Designentscheidung postuliert wurden, geht jedoch nicht hervor. Besonders die **Random Shift Augmentation** wäre eine robuste Verbesserung und würde der Statik, die in den Bildern von Liu & Wang gegeben ist, vorbeugen.

4. Diskussion und Evaluation

Die im Paper präsentierten Ergebnisse und die Methodik zur Lösung eines Montageproblems, bei dem Feinmotorik, Schrauben und präzise Kraftverhältnisse erforderlich sind, wurden mit beeindruckenden Technologien und modernen Ansätzen der Informatik angegangen. Die wissenschaftliche Grundlage der Problematik wurde zuerst formal mit einem MDP formuliert und anschließend mit dem RLPD-Ansatz approximiert. Dieser Ansatz ist technisch anspruchsvoll und berührt Transfergebiete wie Maschinenbau, Elektrotechnik, Physik und Informatik. Der gewählte RLPD-Ansatz wurde aus Gründen der Datenneffizienz und Lerngeschwindigkeit ausgewählt. Die Umsetzung des RLPD wurde in der Ausarbeitung anhand des genutzten Papers von Ball et al. grundlegend erklärt und anschließend mit der Umsetzung von Liu & Wang verglichen. Dabei spielt die Reproduzierbarkeit von Arbeiten eine wichtige Rolle in der Wissenschaft. Es fällt auf, dass Liu und Wang auf eine detaillierte Spezifikation ihrer Critic-Architektur verzichten und diese Designentscheidung unerwähnt lassen. Da der Standard-SAC-Algorithmus ohne diese Modifikation in einem hybriden Setting zu Instabilitäten neigt, bleibt unklar, durch welchen Mechanismus die Autoren die hohe Erfolgsquote des Modells sicherstellen. Diese Intransparenz erschwert nicht nur die Reproduktion der Ergebnisse, sondern lässt auch offen, ob der Erfolg auf einer robusten Architektur oder der Verwendung von Human-in-the-Loop (HIL) beruht.

Die Verwendung von HIL dem Paper ist an vielen Stellen sehr diskret und geht nicht in die Tiefe, was die Art und Weise dieser menschlichen Eingriffe betrifft. In vielen Passagen werden die Eingriffe angesprochen und erwähnt, aber wie viele es tatsächlich sind und wie diese aussehen, bleibt offen, obwohl gerade die Menge der menschlichen Einwirkungen die Qualität der konstruierten RLPDs unterstreichen würde. Dass die Lerngeschwindigkeit deutlich erhöht wird, ist beim mehrfachen Eingreifen und Korrigieren durch den Menschen selbstverständlich. Dementsprechend ist der Vergleich zwischen anderen RL-Ansätzen wie BC, SAC und DP unzureichend, wenn der RLPD + HIL berücksichtigt wird. Eine Ausnahme scheint hier der HG-Dagger-Vergleich zu sein. Beim HG-Dagger wurden „so viele Eingriffe wie in RL“ vorgenommen, wobei auch hier wieder die vage Formulierung von „Interventions“ problematisch bei der Validierung des Lernprozesses ist.

Zudem fehlen Variationen im Lernprozess und in den Testabläufen. Während des Lernprozesses werden identische Bauteile mehrfach unter gleicher Ausgangs- und Montageposition montiert. Bilder werden statisch ins System geladen, wodurch Overfitting eine Begründung für die hohen Erfolgsergebnisse sein könnte. In der Literatur zum visuellen Reinforcement Learning with Prior Data gilt die Datenaugmen-

tation als Standard, um zu verhindern, dass das neuronale Netz lediglich statische Pixelkonstellationen auswendig lernt [4].

Der Ansatz der Autoren stützt sich zudem auf einen Demo-Buffer, der mit 30 erfolgreichen, menschlichen Trajektorien gefüllt ist. Dieses Vorgehen steigert zwar zweifellos die Dateneffizienz des Lernprozesses, verschiebt den Aufwand jedoch lediglich vom Training auf die Datenerhebung. Ein System, das auf hochpräzise menschliche Abläufe angewiesen ist, widerspricht dem Ziel der autonomen Robotik teilweise, da diese auch mit imperfekten Daten umgehen können sollte. Dieses Argument wird durch den „Human-in-the-Loop“ (HIL) verstärkt, bei dem während der Trainingsphase ebenfalls hohe menschliche Aufmerksamkeit und Expertenwissen erforderlich sind. Die Häufigkeit und Form der Eingriffe würde hier deutlichen Aufschluss liefern.

4.1. Fazit

In der vorliegenden Arbeit wurde die Anwendung hybrider Lernstrategien im Kontext der Präzisionsmontage untersucht. Die Modellierung des Montageproblems als Markov-Decision-Process (MDP) und die Nutzung von RLPD zeigen das Potenzial auf, starre Regelwerke durch lernfähige Algorithmen zu ersetzen. Für zukünftige Arbeiten an der Schnittstelle von Informatik und Maschinenbau bleibt die Herausforderung bestehen, solche Systeme transparent für beide Parteien zu gestalten, um die Rekonstruierbarkeit zu fördern. Grundsätzlich ist die hohe Erfolgsquote der Arbeit von Liu und Wang in Bezug auf den HG-Dagger-Lernprozess eindeutig. Die darauffolgenden Testläufe sprechen ebenfalls für sich und liefern einen eindeutigen Nachweis für die erfolgreiche Umsetzung der Montage nach dem Lernen. Es ist jedoch nicht eindeutig, ob diese Erfolgsquote wirklich durch den RLPD erreicht wurde oder ob sie auf die hohe Anzahl an HIL-Eingriffen zurückzuführen ist. Grund dafür ist die mangelnde Dokumentation der Eingriffe unter menschlichem Einfluss. Zudem geht hervor, dass der Impedanz-Controller ein fundamentaler Faktor für die Erfolgsquote ist. Da der Lernprozess zwischen anderen Algorithmen nicht aufbereitet wurde, ist unklar, ob diese auch den Impedanz-Controller nutzen. Ohne einen Impedanz-Controller sind die Vergleiche nicht tragfähig, da ein deutlicher Vorteil gegenüber den *„rohen“* Algorithmen vorliegt. Eine bessere Aufbereitung und mehr Transparenz beim Vergleich wären hier ausschlaggebend für die Validierung der Ergebnisse.

Bibliografie

[1] „KR QUANTEC“. Zugegriffen: 7. Januar 2026. [Online]. Verfügbar unter: https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/kr-quantec

[2] I. I. F. o. Robotics, „International Federation of Robotics“. Zugegriffen: 7. Januar 2026. [Online]. Verfügbar unter: https://ifr.org/

[3] S. Liu und L. Wang, „Vision intelligence-conditioned reinforcement learning for precision assembly“, *CIRP Annals*, Bd. 74, Nr. 1, S. 13–17, Jan. 2025, doi: 10.1016/j.cirp.2025.04.016.

[4] P. J. Ball, L. Smith, I. Kostrikov, und S. Levine, „Efficient Online Reinforcement Learning with Offline Data“.

[5] Google DeepMind, „RL Course by David Silver - Lecture 2: Markov Decision Process“. Zugegriffen: 7. Januar 2026. [Online]. Verfügbar unter: https://www.youtube.com/watch?v=lfHX2hHRMVQ

[6] J. Gong, W. Liu, M. Pei, C. Wu, und L. Guo, „ResNet10: A lightweight residual network for remote sensing image classification“, in *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, Jan. 2022, S. 975–978. doi: 10.1109/ICMTMA54903.2022.00197.

[7] T. Haarnoja, A. Zhou, P. Abbeel, und S. Levine, „Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor“. Zugegriffen: 8. Januar 2026. [Online]. Verfügbar unter: http://arxiv.org/abs/1801.01290

[8] „Was ist ein neuronales Netz? | IBM“. Zugegriffen: 8. Januar 2026. [Online]. Verfügbar unter: https://www.ibm.com/de-de/think/topics/neural-networks

[9] S. S. T. Yau, X. Chen, X. Jiao, J. Kang, Z. Sun, und Y. Tao, „Estimation Algorithms Based on Deep Learning“, *Principles of Nonlinear Filtering Theory*. Springer Nature Switzerland, Cham, S. 385–427, 2024. doi: 10.1007/978-3-031-77684-7_10.

[10] D. P. Kingma und M. Welling, „Auto-Encoding Variational Bayes“. Zugegriffen: 11. Januar 2026. [Online]. Verfügbar unter: http://arxiv.org/abs/1312.6114

[11] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, und S. J. Gershman, „Human Learning in Atari“.

[12] D. Silver *u. a.*, „Mastering the game of Go with deep neural networks and tree search“, *Nature*, Bd. 529, Nr. 7587, S. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[13] S. Ross und J. A. Bagnell, „Agnostic System Identification for Model-Based Reinforcement Learning“. Zugegriffen: 11. Januar 2026. [Online]. Verfügbar unter: http://arxiv.org/abs/1203.1007

[14] S. Thrun und A. Schwartz, „Issues in Using Function Approximation for Reinforcement Learning“, *Proceedings of the 1993 Connectionist Models Summer School*. Psychology Press, 1994.

[15] J. L. Ba, J. R. Kiros, und G. E. Hinton, „Layer Normalization“. Zugegriffen: 11. Januar 2026. [Online]. Verfügbar unter: http://arxiv.org/abs/1607.06450

[16] „What is Overfitting? | IBM“. Zugegriffen: 11. Januar 2026. [Online]. Verfügbar unter: https://www.ibm.com/think/topics/overfitting