

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	2
2. Systemarchitektur & Problemmodellierung	2
2.1. Markov Decision Process (MDP)	2
2.2. MDP zur Modellierung eines Montageproblems	5
2.3. Actor-Critic-Modell	5
2.3.1. Critic	6
2.3.2. Actor	6
3. Effizientes Online Reinforcement Learning mit offline Daten - RLPD	7
3.1. Hybrides Buffer-System und Replay Ratio	7
3.2. Layer Normalization	8
3.3. Sample Efficient RL	9
4. Diskussion und Evaluation	10
4.1. Fazit	11
Bibliografie	12

1. Einleitung & Motivation

Roboter in der Industrie, insbesondere der Fertigungstechnik, sind für Unternehmen essenziell geworden und reduzieren hohe Kosten durch 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, bereits weit verbreitet mit ihren Automatisierungslösungen sind. Dort werden hochautomatisierte Produktionsketten als Lösung angeboten, die hauptsächlich auf „*Flexible Robotik*“ setzen, in denen Roboterarme Aufgaben wie Montage, Schweißen oder Lackierung übernehmen [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 Feinmotorik und kontaktreicher dynamischer 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 keine „intelligente“ Reaktion auf derartige Probleme geben. Eine Lösung zur statischen Programmierung und festen Regelwerken, 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]. Alle genannten Konzepte sind in der Informatik moderne Technologien und erfordern tiefes Wissen aus der Informatik.

Das Paper postuliert eine mögliche Lösung eines Lernprozesses für Roboter, in der Präzisionsarbeit durch intelligente visuelle Verarbeitung von räumlicher Umgebung und „Human-in-the-Loop“ Reinforcement Learning erzielt wird. Dabei werden neuronale Netze, lernende Algorithmen und intelligente Impedanzcontroller verwendet. Demonstriert wird die Methode anhand der Montage von Computer-Hardware-Komponenten (konkret: RAM-Module und Kühlsystem auf einem Mainboard). Die Montage solcher kontaktreicher Komponenten mit der Kombination von Schrauben, korrektes Einsetzen und Widerstandserkennung beim RAM, würde einen großen Implementierungsaufwand beim klassisch statischen Programmieren aufweisen. Zusätzlich dazu wäre die Fehlerquote trotzdem hoch, da Abweichungen in Millimeterbereich bereits in Schäden an Bauteilen resultieren könnten (ein RAM-Riegel, der 1mm 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. Zuerst wird das eigentliche Problem als ein Markov Decision Process (MDP) modelliert. Dies ist nötig, da Reinforcement Learning Algorithmen auf solche MDP operieren. Des Weiteren wird dann noch die verteilte Architektur, das Actor-Learner-Modell kurz erklärt und übergeleitet zur Umsetzung des lernenden Algorithmus. Der im Paper vorgestellte RLPD (Reinforcement Learning with Prior Data) bietet eine dateneffiziente Lösung an, die von Liu & Wang aufgegriffen und zum Teilen umgesetzt wird. Dabei wird die Umsetzung im Paper genauer beleuchtet und analysiert unter Bezugnahme des Papers von Ball et. al. [4]. Abgeschlossen wird die Ausarbeitung mit einer Diskussion und Evaluation der gesamten Umsetzung der Informatik-Ansätze und Kritikwürdigung.

2. Systemarchitektur & Problemmodellierung

2.1. Markov Decision Process (MDP)

Das Paper modelliert das Montageproblem als einen Markov-Decision-Process (MDP). Dabei handelt es sich um eine formale, mathematische Definition eines Entscheidungsproblems, welches hier zur Optimierung der Montage verwendet wird. Ein MDP lässt sich als gerichteter Graph modellieren, wobei die Knoten als Zustände und die Kanten als Zustandsübergänge (Transitionen) interpretiert werden,

die durch Handlungen ausgelöst werden. Zum besseren Verständnis wird ein bekanntes Beispiel aus der Vorlesung von David Silver (DeepMind / UCL) betrachtet: Der beschriebene Graph [Abbildung 1] modelliert den Studienalltag [5].

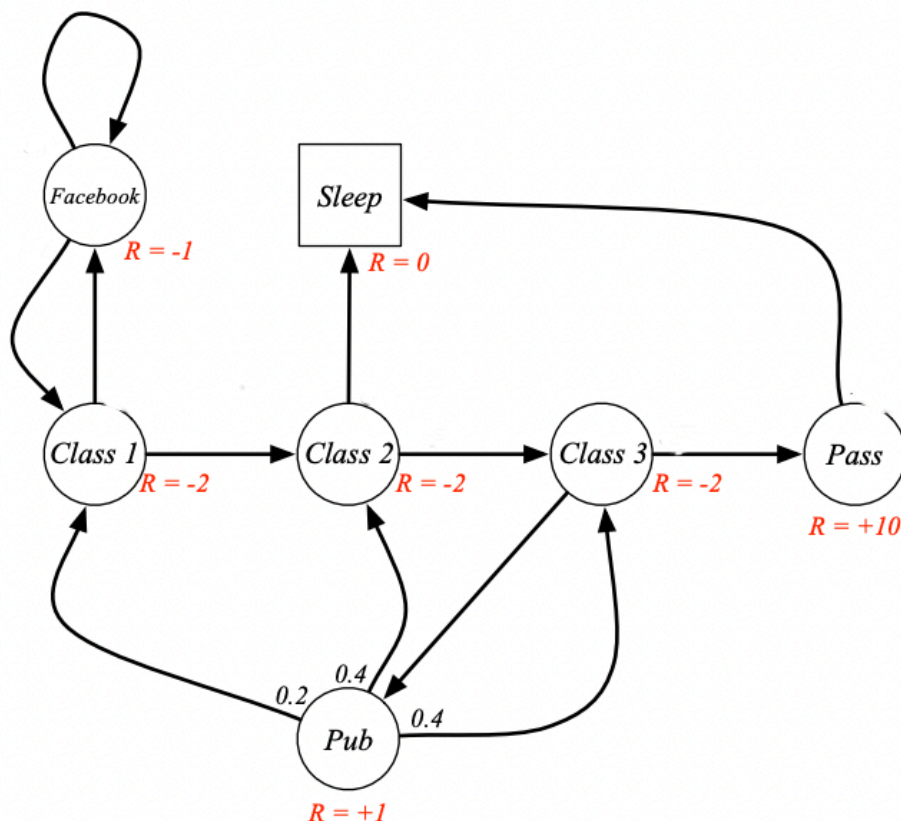


Abbildung 1: Optimal Action-Value Function for Student MDP - David Silver [5].

Um seinen Kurs bestehen zu können, müssen Studenten alle drei „Class“-Zustände erfolgreich durchlaufen. Die Kreise repräsentieren hierbei die Zustände, wobei der Zustand „Class 1“ hier als Start-Zustand dient. In diesem Zustand kann der Agent (Student) nun eine Handlung wählen, entweder „Facebook“ oder „Study“. Wählt er „Study“, folgt eine Transition, die mit einer Wahrscheinlichkeit behaftet ist (in dem Fall implizit 1.0/100%). Der Reward (R) ist der Wert, den der Agent für das Ausführen einer Aktion erhält. In „Class 1“ kostet jeder Zeitschritt beispielsweise $R = -2$ (negativer Reward / Bestrafung). Sobald sich der Agent in „Class 3“ befindet, kann er sich für „Pub“ entscheiden. Von da aus landet er mit unterschiedlichen Wahrscheinlichkeiten in „Class“ 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, die die Summe der erwarteten Rewards maximiert.

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

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

- **S (State Space):** Beschreibt die Menge aller Zustände. In unserem Beispiel ist S alle dargestellten Knoten, wobei $s \in S$ ein konkreten Knoten beschreibt
- **A (Action Space):** Beschreibt die Menge aller verfügbarer Aktionen. Im Zustand „Class 1“ wäre das „Facebook“ oder „Study“.
- **P und 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 Start-Zustand 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 mit der von David Silvers Vorlesung. Um auf weiteres Vorgehen aufzubauen, wird sich an die Definition des Papers gehalten [3].

- **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.
- **γ (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, wobei das ($\gamma = 0$)-Beispiel in einer Facebook-Schleife verfiel und ($\gamma = 1$) in „Pass“ überging.

Hat man nun ein MDP formuliert, gilt es dieses zu lösen, wobei die Lösung bei einem Optimierungs-/Maximierungsproblem der höchstmögliche erreichbare Wert beschreibt, in unserem Fall der Reward R . Die gängigste Vorgehensweise bei dem Lösen von MDPs ist das Aufstellen einer Strategie, einer sogenannten **Policy** (π). Nach einem bekannten Theorem gilt, dass es für jeden MDP eine optimale Policy (π) gibt, die besser oder gleich aller 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 und dementsprechend Unsicherheiten vorhanden sind, bilden wir den Erwartungswert aller möglichen Verläufe. Das γ dient hier wie bereits angedeutet als Diskontierungsfaktor, der je nach Wahl über 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 eines Studentenleben ist eine optimale Policy (π_*) einfach zu bestimmen. Mithilfe der **optimale Action-Value Function** $Q_*(s, a)$ können wir bestimmen, welchen maximalen akkumulierten Reward man erwarten kann, wenn man im Zustand s die Aktion a wählt und danach optimal weitermacht. Ein Blick auf [Abbildung 1] verdeutlicht dies am Zustand „Class 3“:

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

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 für das Bestimmen von Q_* in allen Zuständen ist der Beginn am Ende und zurückschauen in Vorgängerzuständen [5]. Nachdem vereinfacht ausgedrückten Bellmann'schen Optimalitätsprinzip gilt nämlich, dass jede Teilpolicy π_{sub} einer optimalen Policy π_* , auch optimal ist [5]. Zur Veranschaulichung: Wenn die kürzeste ICE Strecke von Berlin nach München, über Leipzig ist, dann ist die kürzeste Strecke von Leipzig nach München, die gleiche. Aufgrund dieser Eigenschaft können wir das Problem beginnend vom Ende 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 mit welcher Wahrscheinlichkeit 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.

- **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 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.
- **A (Action Space):** Die Handlungsmöglichkeiten werden über die kartesische Koordinatenposition des Roboterarms und Griff-Zustand definiert.
- **P und p (Wahrscheinlichkeiten):** Der Roboterarm wird kaum immer von der gleichen Stelle aus anfangen, zu montieren, noch wird das Mainboard immer im exakten Millimeterbereich gleich liegen. Deshalb muss eine Wahrscheinlichkeitsverteilung $P(s)$ definiert werden, die unterschiedliche Start-Zustände modelliert. Zudem stellt p nicht mehr einfache Wahrscheinlichkeiten an einer Transition dar, sondern die gesamte Dynamik des Systems wird durch p repräsentiert. Der Roboterarm wird egal wie präzise er ist, sich mit einer gewissen physikalischen Schwankung von der angegebenen Trajektorie abweichend. Aufgrund der physikalischen Komplexität, ist p uns nicht bekannt, sondern wird durch Reinforcement Learning approximiert.
- **R (Reward Function):** Die Autoren haben für R ein binäres Klassifizierungssystem gewählt, dass anhand von vorher trainierten Demos beurteilt, ob eine Montage erfolgreich, oder fehlgeschlagen ist.
- **γ (Discount Factor):** Erfüllt den exakt selben Zweck wie in 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 aber 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 der Kern, das Zusammenspiel und die formale Definition vom Actor und Critic (SAC) veranschaulicht [7].

2.3. Actor-Critic-Modell

Die analytische Lösung eines MDPs 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 Montageproblem im Paper ist dies aufgrund der hochdimensionalen Bilddaten und Systemdynamik 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 einfachhaltshalber

nehmen wir folgendes an: Ein neuronales Netz Q_φ lernt eine Funktion $f(X)$ durch Zuordnung eines Eingabevektors $X = (x_1, x_2, x_3, \dots)$ um eine Reaktion Y vorherzusagen. Dabei besteht ein Neuronales Netz aus unterschiedlichen Schichten, einem Input Layer für X , mehrschichtigen versteckten Layer, das hauptsächlich für das Lernen von $f(X)$ zuständig ist und einem Output Layer, indem Y ausgegeben wird [8].

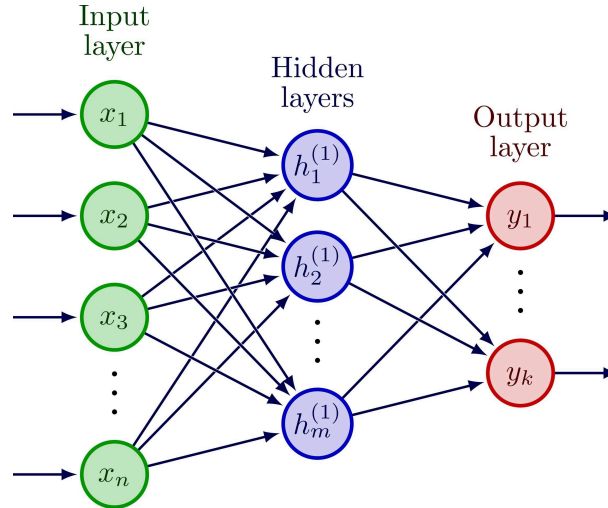


Abbildung 2: Neuronales Netz mit drei Layer Ebene - [9]

Die Autoren des Papers nutzen zwei Neuronale Netzwerke, eines für den Actor und eines für den Critic. Im folgenden wird tiefer auf die einzelnen Netzwerke eingegangen und ihre Wechselwirkung aufeinander analysiert.

2.3.1. Critic

Der Critic bewertet eine derzeitige Einschätzung des Netzwerks und vergleicht diese mit der tatsächlichen Situation und Zukunft. Dafür wurde folgende Loss-Funktion \mathcal{L}_Q aufgestellt:

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

Die Funktion \mathcal{L}_Q bestimmt die Fehlerquote der Parameter φ im neuronalen Netzwerk Q_φ , indem es anhand einer Stichprobe (einem Batch aus dem Replay Buffer) den **mittleren quadratischen Fehler** (Mean Squared Error) zwischen zwei Werten berechnet. Dafür nimmt es einmal den Wert der eigenen Vorhersage $Q_\varphi(s, a)$, also eine Einschätzung des Netzwerks über die Handlung a in Zustand s und zum anderen dem **Bellman-Zielwert** (Target), der sich zusammen aus dem tatsächlich erhaltenen Reward $R(s, a)$ und der diskontierten Prognose, also $\gamma \times Q_{\bar{\varphi}}(s', a')$, des **Target-Netzwerks** $Q_{\bar{\varphi}}$ für den Folgezustand ergibt. Wichtig zu beachten ist, dass es sich beim Target-Network um ein anderes Network handelt als Q_φ . Das liegt daran, dass eine sofortige Aktualisierung der Werte zum gleichzeitigen Veränderung des Netzwerkes und Zieles führen würde. Deshalb wird eine Kopie $Q_{\bar{\varphi}}$ erstellt, wodurch das Training stabilisiert wird. Dadurch wird verhindert, dass das Ziel („Moving Target“) während des Updates zu stark schwankt, indem die Parameter $\bar{\varphi}$ 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 tatsächlichen Bewegungen ausführt bzw. die Policy (φ_θ) aktualisiert, auf dem die Bewegungen basieren.

2.3.2. Actor

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

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

Die Funktion \mathcal{L}_π setzt sich aus zwei Zielen zusammen: Einmal Gierig (Exploitation) zu sein und andererseits Neugierig (Exploration). Das $(-)$ zu Beginn der Funktion wandelt das Maximierungsproblem in ein Minimierungsproblem um, da Computer besser darin sind, Fehler zu minimieren. Denn die Maximierung vom Reward, also dem Suchen eines globalen Optimums einer Funktion $f(x)$ ist für uns gleichbedeutend wie das Suchen des globalen Minimums $-f(x)$, nur einfacher für Computer umzusetzen. Der Teil, der die Gier des Actors steuert, ist in diesem Teil enthalten: $E_{a \sim \pi_\theta(\theta)} [Q_\varphi(s, a)]$. Das $a \sim \pi_\theta(\theta)$ hat dabei eine relativ wichtige Bedeutung. Es ist mathematisch nicht möglich, Rückpropagierung (Backpropagation) in einem neuronalen Netz zu betreiben, wenn Stochastik zugrundeliegt. Denn aus einem Sample a können keine Rückschlüsse zur Zufallsverteilung gezogen und plausible Anpassungen am Neuronalen Netz vorgenommen werden. Die Autoren bedienen sich hier dem Trick der **Reparametrisierung (Reparameterization)**, indem grob gesagt der Zufall in ein Standard-Rauschen ϵ ausgelagert wird, wodurch der stochastische Sample, differenzierbar wird [10]. Durch diesen Trick kann über dem Critic Q_φ , der Actor Q_π lernen, sich anzupassen. Der zweite Teil der Funktion $\tau \Phi(\pi_\theta(\cdot | s))$ ist zuständig für die Exploration. Damit wird vorgebeut, dass sich der Actor nicht zu früh in einer approximierten Lösung festsetzt, sondern nach anderen, eventuell besseren sucht. Der Hyperparameter τ (Temperatur) steuert dabei die Balance: Ein hohes τ fördert 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 an unter der Berücksichtigung aller möglichen Handlungen a (hier gekennzeichnet durch $(\cdot | s)$, innerhalb der Policy (π_θ)).

Der Actor leitet den Roboter unter der Berücksichtigung des Critics und eigenem „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, dass 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]. Reward 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, 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 kommt die Dauer des Lernprozesses vor allem aus der Anfangsphase, in denen der Algorithmus erst eine gewisse Richtung ermitteln muss, in der sich die optimale Policy befindet. Die Richtungsfindung, kann aber minimiert werden, indem beim Deep RL im voraus bereits suboptimale Policies oder menschliche Demonstrationen die Richtung vorgeben. Mit einher gehen jedoch Probleme mit diesen Demonstrationen. Während der RL-Algorithmus lernt, werden in kurzer Zeit bereits eine Vielzahl von Durchläufen in den Replay-Buffer, den Speicher, in dem alle Epochen gespeichert werden, geladen und überwiegen schnell die Anzahl an selbstgelandeten Daten. Die Autoren Ball et. al. erkannten das ebenfalls und entwarfen basierend auf Ross & Bagnell (2012) eine symmetrische Replay-Buffer Architektur [13]. Dabei werden statt nur einem Replay-Buffer, zwei Buffer angelegt, wobei einer die Online Daten des RL-Algorithmus speichert und aufnimmt und einem

Offline Buffer, der als Beispiel die eben genannten menschlichen Demonstrationen, speichert. Das hat vor allem den Vorteil, dass unter der schnell groß werdenden Menge an RL-Abläufen, die vorgefertigten Richtungsgeber nicht untergehen. Damit aber auch beim Samplen die Gewichtung erhalten bleibt, wird aus beiden Buffern die gleiche Menge an Daten entnommen, genau genommen jeweils 50%. Diese Designentscheidung erweist sich als besonders effektiv, wie in der Evaluation zu erkennen ist, auch wenn alleine nicht ausreichend [4].

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

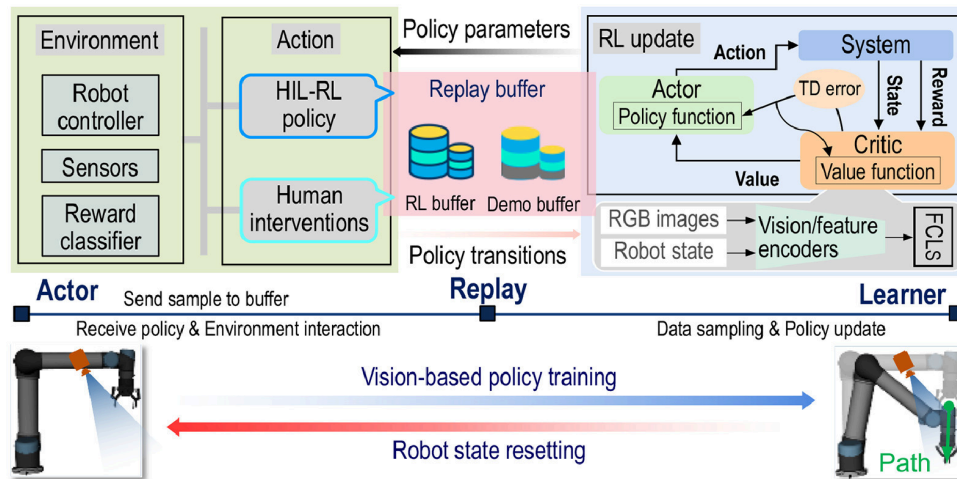


Abbildung 3: Arctor-Learner Architektur mit zwei Buffern - [3]

Näher wird auch erklärt, dass es sich um menschliche Demonstrationen handelt, die im voraus erstellt wurden. Dabei wurden für jede Montageaufgabe (CPU-Kühlkörper, RAM, Lüfter) jeweils 30 erfolgreiche Trajektorien, also volle Bewegungsabläufe einer Montage, verwendet. Hierbei wird „Erfolgreich“ durch zwei Kriterien definiert. Zum einen darf sich das zu montierende Objekt nicht mehr als 0.1mm der Zielposition entfernt befinden. Zum anderen, sollte Ersteres erfüllt sein, muss der vorher trainierte **Binary Classifier**, ebenfalls mit einer Wahrscheinlichkeit von min. 97% die Montage als „1“, also erfolgreich bewerten. Hier wurde sich dazu entschieden, hoch qualitative menschliche Abläufe als Offline Daten zu verwenden. Die genaue Vor- und Nachteile variieren stark nach zu optimierendem Problem. Unter der Grundlage jedoch, 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, das sogenannte **Out-of-Distribution (OOD)** Problem. Bei Actor-Critic-Architekturen mit OOD Daten sind diese meist nicht definiert während dem Lernprozess von RL. Dabei passiert es, dass der Critic die neuen eingehenden Daten stark „überschätzt“, da dieser die Daten mit den Offline Daten aus dem Demo-Buffer in Beziehung bringt. Folge daraus in der Praxis sind Instabilitäten im Training und **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 Werte innerhalb eines Zahlenbereiches (meist mit 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 ansonsten eine Umsetzung von RLPD nicht möglich wäre. Wie die Gleichung der Q -Loss Funktion zeigt, nutzen die Autoren die standardmäßige Soft-Bellman-Optimierung des SAC-Algorithmus:

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

Diese Funktion minimiert lediglich die Differenz zwischen der Vorhersage Q_φ und dem Zielwert Q_π . 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 sie die RLPD-Methodik vollständig adaptiert haben, ist der Einsatz dieser Normalisierungsschichten zwingend erforderlich, da der Critic sonst bei der Verarbeitung der Daten aus den zwei Buffern zur Divergenz neigen würde.

3.3. Sample Efficient RL

Die Designentscheidung eines zweiten Offline Buffers führt dazu, dass Sampling aufwendiger wird. Zur Minimierung des Aufwands wird die Geschwindigkeit der Lernprozesses modifiziert, um Effizienz beizubehalten. Eine Möglichkeit dabei ist es den UTP-Wert zu erhöhen, den Wert der vorgibt, wie viele Lernschritte (Critic passt Gewichte an) pro Arbeitsschritt (Actor führt Handlung aus) durchgeführt werden. Jedoch läuft man damit Gefahr, dass der RL-Algorithmus in **Überanpassung (Overfitting)** verfällt. Die Überanpassung beschreibt einen Zustand, indem ein Modell sich zu stark an einem lokalen Optimum angepasst hat und irrelevante Faktoren berücksichtigt [16]. Zum besseren Verständnis wird als veranschaulichtes Beispiel oft der Unterschied zwischen „Verstehen“ und „Auswendig lernen“ aufgewiesen. Wenn ein RL-Algorithmus zu exakt gerlernt hat, eine Aufgabe zu lösen, sorgt der Zustand der Überanpassung dafür, dass die spezifisch gelernte Aufgabe mit einer hohen Genauigkeit gelöst wird, jedoch bei leichten Änderungen bereits scheitert. Als Lösung dafür nennen Ball et. al. einige Möglichkeiten, wobei sie sich für die Methoden mit **Random Ensemble Distillation** und **Random Shift Augmentations** entscheiden. Ersters beschreibt die Verwendung mehrerer Critics, die in Wechselwirkung zueinander sich vor Divergenz schützen. Zweiteres wird genutzt, da Ball et. al. ebenfalls Bilder zum Training verwendet. Dabei werden die Bilder um wenige Pixel zufällig verschoben, wodurch ein Art „*wackeln*“ imitiert wird. Das ist besonders nützlich, da so der RL-Algorithmus lernt den RAM-Sockel wirklich als RAM-Sockel zu erkennen und sich nicht statisch auf Pixelpositionen versteift (Overfitting).

Indizien für Overfittingprävention im Paper von Liu & Wang geht vor allem hervor bei der Extraktion von Daten aus Bildern. Sie nutzen dafür ein vortrainiertes ResNet-10 Netzwerk zur Bilderkennung, dass bereits schon auf Millionen von Daten trainiert wurde, wodurch davon auszugehen ist, dass dieses bereits gegen Overfitting besteht. ResNet-10 ist eine vereinfachte Version des bekannten ResNet, dass Bilder auf relevante Daten extrahiert, mit denen viele RL-Algorithmen arbeiten können [6]. Im Paper werden die Kamerabilder des Robters ins ResNet gesetzt, wodurch das Problem der geringen Datenmenge aus 1400 Bildern, vorgebeugt wird. Diese Daten werden dann in den Binary Classifier eingespeist, der dann Abläufe des Actors bewertet und mit in die Learner-Architektur liefert. Die Actor-Learner-Architektur ist gut aus Abbildung 4 zu entnehmen.

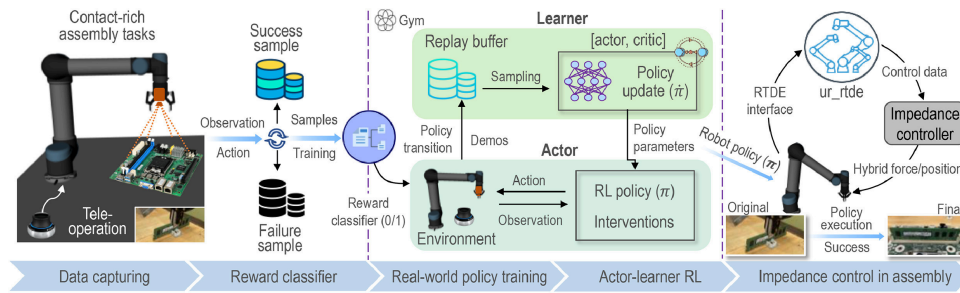


Abbildung 4: Arctor-Learner Architektur und Binary/Reward Classifier - [3]

Zusätzlich dazu wird „**Image Cropping**“ verwendet, wodurch Bilder auf 128×128 in relevante Bereiche verkleinert werden. Jedoch geht keine direkte Nutzung von **Random Shift Augmentations**, **Random Ensemble Distillation** oder anderer genannter Methoden aus dem Paper von Ball et. al. hervor, die als essenzielle Designentscheidung postuliert wurden, um RLPD umzusetzen und Overfitting vorzubeugen. Besonders die **Random Shift Augmentation** wäre eine robuste Verbesserung und würde die Statik, die in den Bildern von Liu & Wang gegeben ist, vorbeugen.

4. Diskussion und Evaluation

Die im Paper präsentierte Ergebnisse und Methodik zur Lösung eines Montageproblems, indem Feinmotorik, Schrauben und präzise Kraftverhältnisse benötigt werden, wurden mit Technologien und modernen Ansätzen der Informatik beeindruckend angegangen. Die wissenschaftliche Grundlage der Problematik, die zuerst Formal mit einem MDP formuliert und anschließend per RLPD approximiert wurde, ist technisch anspruchsvoll und schneidet stark mit Transfergebieten wie Maschinenbau, Elektrotechnik, Physik und der Informatik. Der als Kern gewählte RLPD Ansatz, wurde aus Gründen der Dateneffizienz und Lerngeschwindigkeit gewählt. Die Umsetzung des RLPDs wurde in der Ausarbeitung einmal anhand des genutzten Papers von Bell et. al. für RLPD grundlegend erklärt und anschließen mit der Umsetzung von Liu & Wang verglichen. Dabei spielt die Reproduzierbarkeit von Arbeiten eine wichtige Rolle in der Wissenschaft. Es fällt auf, dass Liu & 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 auf der Verwendung von Human-in-the-Loop (HIL) zurückzuführen ist.

Die Verwendung von HIL in den Paper ist an vielen Stellen sehr diskret und geht nicht in Tiefe darauf ein, wie diese menschlichen Eingriffe aussahen. In vielen Passagen werden die Eingriffe angesprochen und erwähnt, aber wie viele tatsächlich es sind und diese aussehen, wird offen gelassen, obwohl gerade die Menge menschlicher Einwirkungen die Qualität der konstruierten RLPDs unterstreichen würden. Das die Lerngeschwindigkeit deutlich erhöht wird, ist beim mehrfachen Eingreifen und korrigieren des Menschen selbstverständlich. Dementsprechend ist der Vergleich zwischen anderen RL-Ansätzen wie BC, SAC und DP unzureichend, wenn der RLPD + HIL verglichen wird. Eine Ausnahme scheint hier der HG-Dagger Vergleich zu sein. Beim HG-Dagger wurde die „gleiche Anzahl an Eingriffen wie RL“ vorgenommen, wobei auch hier wieder die vage Äußerung von „Interventions“ problematisch bei der Validierung des Lernprozesses ist.

Zudem fehlt es an Variation im Lernprozess und Test-Abläufen. Während des Lernprozesses werden identische Bauteile mehrmals 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 Datenaugmentation als Standard, um zu verhindern, dass das neuronale Netz lediglich statische Pixel-Konstellationen

auswendig lernt [4]. Da der Versuchsaufbau in einer kontrollierten Laborumgebung stattfand, besteht der begründete Verdacht, dass die Policy π_θ weniger eine intelligente Reaktionsfähigkeit auf Systemdynamiken erlernt hat, sondern vielmehr eine Überanpassung an die fixe Kameraperspektive und Beleuchtung angenommen hat.

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

4.1. Fazit

Die vorliegende Arbeit untersuchte die Anwendung von hybriden Lernstrategien im Kontext der Präzisionsmontage. 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 zu postulieren um Rekonstruierbarkeit zu fördern. Grundsätzlich ist die hohe Erfolgsquote der Arbeit von Liu & Wang in Bezug auf den HG-Dagger Lernprozess Vergleich eindeutig. Die darauffolgenden Testläufe sprechen ebenfalls für sich und liefern eindeutig die Erfolgreiche Umsetzung der Montage nach dem Lernen. Jedoch ist es nicht eindeutig, ob diese Erfolgsquote wirklich durch den RLPD erreicht wurde, oder die Hohe Anzahl an HIL-Eingriffen zurückzuführen ist. Grund dafür ist die mangelnde Dokumentation der Eingriffe unter menschliches Wirken. Zudem geht hervor, dass der Impedanz-Controller ein fundamentaler Faktor bei der Erfolgsquote darstellt. Aufgrund des fehlenden Aufbereitung des Vergleiches beim Lernprozess zwischen anderen Algorithmen, wird nicht klar, 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 Transparenz des Vergleiches wäre 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>