

# Praktische Anwendungsfälle von Deep Reinforcement Learning

Sebastian Graf  
Fakultät für Informatik  
Seminar Theoretische Informatik  
Rosenheim, Deutschland  
sebastian.graf@stud.th-rosenheim.de

**Zusammenfassung**—Diese Arbeit thematisiert aktuelle, praktische Anwendungsfälle für Deep Reinforcement Learning. Nach der Vorstellung von OpenAI Five [1] - einer von OpenAI entwickelten künstlichen Intelligenz, die auf Deep Reinforcement Learning basiert - folgen zwei konkrete, praktische Anwendungsfälle. Zum einen wurde eine Roboterhand trainiert, einen Würfel durch geschickte Manipulation in eine bestimmte Position und Orientierung zu bringen [2]. Und zum anderen sollen zwei jeweils kooperierende KIs das Spiel Hide and Seek lernen und dabei Gegenstände ihrer Umgebung sinnvoll nutzen [3]. Ziel der Arbeit ist es, die Kernpunkte in aktuellen Anwendungsfällen von Deep Reinforcement Learning und aktuelle Problemstellen zu verdeutlichen.

## I. EINLEITUNG

Maschinelles Lernen hat in den letzten Jahren zunehmend an Bedeutung in vielen Bereichen unseres Lebens gewonnen. Von der Sprach- und Bilderkennung bis hin zum autonomen Fahren reichen Anwendungen weit über die traditionellen Felder der Informatik hinaus. Eine besondere Form des maschinellen Lernens ist Deep Reinforcement Learning (DRL). DRL ermöglicht es, das Verhalten von Systemen auf Grundlage des Konzepts von Belohnung und Bestrafung zu optimieren und sie dazu zu bringen, komplexe Aufgaben zu lösen, die mit den Ansätzen der traditionellen Informatik nicht lösbar sind. Diese Seminararbeit beschäftigt sich intensiv mit dem Konzept DRL und untersucht, wie ein Algorithmus, der ursprünglich aus der Idee heraus entstanden ist, ein professionelles Team in einem Computerspiel zu besiegen, in zwei weiteren konkreten Fällen angewandt wurde. Dabei werden sowohl theoretische Grundlagen, aktuelle Entwicklungen und Herausforderungen als auch interdisziplinäre Forschungsfelder, wie zum Beispiel Robotik und Evolutionsforschung, behandelt.

## II. THEORETISCHE GRUNDLAGEN

In dieser Arbeit treffen interdisziplinäre Forschungsfelder aus den Gebieten der Informatik und Robotik aufeinander. Das Kapitel *Theoretische Grundlagen* behandelt grundlegende theoretische Hintergründe und bildet so die Basis der Arbeit. Thematisiert werden die Unterpunkte Deep Reinforcement Learning mit Fokus auf den Begriffen Policy, Proximal Policy Optimization, Reward Function, Self-Play Experience, Long

Short-Term Memory (LSTM), intrinsische Motivation sowie Grundlagen der Robotik mit Fokus auf Shadow Dexterous Hand.

### A. Deep Reinforcement Learning

Von **Deep Reinforcement Learning** spricht man, wenn Probleme des Reinforcement Learnings mit Methoden des Deep Learnings gelöst werden [4].

Entsprechend ihrem Buch *Reinforcement Learning: An Introduction* kann **Reinforcement Learning** laut Sutton und Barto [5] als Lernen der Verknüpfung von Situationen und Aktionen verstanden werden, bei dem es darum geht, eine Belohnung zu maximieren (siehe Abbildung 1). Der Lernende (Agent) muss durch Ausprobieren selbst herausfinden, welche Aktionen in welchen Situationen günstig sind und welche nicht. In komplexen Anwendungsfällen beeinflussen die Aktionen nicht nur die unmittelbare Belohnung, sondern auch die der nächsten Situationen. Versuch und Irrtum (trial and error) und verzögerte Belohnung (delayed reward) gelten laut Sutton und Barto als die bedeutendsten Merkmale von Reinforcement Learning.

Ein weiteres Merkmal des RL ist, dass es normalerweise keine vorgegebenen Labels für die Aktionen gibt. Dadurch fällt Reinforcement Learning in die Kategorie des unüberwachten Lernens (unsupervised learning). Reinforcement Learning unterteilt sich nach Deisenroth et al. in die Kategorien modellfrei und modellbasiert. In dieser Arbeit werden ausschließlich modellfreie Algorithmen betrachtet, welche sich wiederum in wertbasierte und strategiebasierte Ansätze unterteilen lassen. Die Mischform wird in der Regel als Actor-Critic bezeichnet. Wertbasierte Formen des RL sind zum Beispiel Monte-Carlo-Methoden. Sie basieren darauf, dass ein Agent eine Nutzenfunktion besitzt, die einen Zustand oder eine Aktion bewertet. Strategiebasierte Methoden, wie zum Beispiel Proximal Policy Optimization (PPO), versuchen, die zu erwartende Belohnung direkt durch die Parametrisierung einer Policy zu maximieren [6].

Goodfellow et al. beschreiben **Deep Learning** im Lehrbuch *Deep Learning* [7] als Teilbereich des maschinellen Lernens,

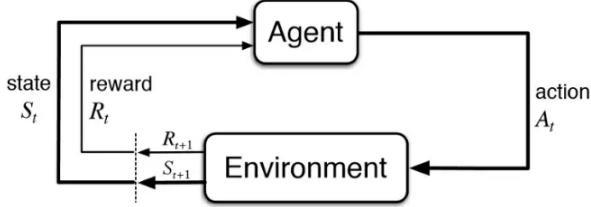


Abbildung 1. Schematischer Ablauf beim Reinforcement Learning. (Quelle: <https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292>, aufgerufen am 02.07.2023)

bei dem mehrschichtige künstliche neuronale Netze (Deep Neural Networks) verwendet werden. Durch diese tiefen neuronalen Netze können beispielsweise komplexe Muster aus Daten abstrahiert werden. Entsprechend sind Methoden des Deep Learnings besonders gut für Probleme geeignet, bei denen große Datensätze verwendet werden.

Nachfolgend werden weitere wichtige Begriffe und Konzepte des Deep Reinforcement Learning erläutert:

1) *Policy*: Unter der Policy versteht man beim Deep Reinforcement Learning eine Funktion, die einen bestimmten Zustand mit einer Aktion verknüpft [4]. Beim DRL wird das Modell, das nach einer gewissen Policy handelt, als *Agent* bezeichnet. Die Optimierung der Policy ist beim DRL von zentraler Bedeutung. Hierfür gibt es verschiedene Methoden, die als Optimierungsverfahren bezeichnet werden. Sie lassen sich grob in zwei Arten unterteilen. Auf der einen Seite ist es das value-based Reinforcement Learning und auf der anderen Seite gibt es auch Ansätze, die optimale Policy direkt zu finden. Bekannte Optimierungsverfahren sind PILCO [8], LSPI [9] und Q-Learning [10]. Die vorliegende Arbeit beschäftigt sich mit Papieren, in denen Proximal Policy Optimization (PPO) [11] verwendet wird.

2) *Proximal Policy Optimization*: Proximal Policy Optimierung (PPO) wurde von OpenAI entwickelt und 2017 vorgestellt [11]. Mittlerweile zählt PPO zu den populärsten on-policy RL Algorithmen. Durch den Algorithmus werden gleichzeitig eine stochastische Policy und eine Schätzfunktion für die Value Function optimiert. PPO verschachtelt das Sammeln von neuen Episoden mit der Optimierung der Policy. Nachdem ein Stapel (Batch) neuer Übergänge gesammelt wurde, wird die Policy wie folgt optimiert: Minibatch-SGD wird verwendet, um das Maximum zwischen der Wahrscheinlichkeit einer Handlung basierend auf der aktuellen Policy und der Wahrscheinlichkeit der gleichen Handlung unter Verwendung der alten Policy zu bestimmen. Dadurch kann die Policy dazu gebracht werden, dass sie bevorzugt gute Entscheidungen trifft und sich eher selten für schlechte Aktionen entscheidet. Neben der Policy wird eine Schätzfunktion für die Value-Function durch überwachtes Lernen (supervised learning) trainiert. Deren Aufgabe ist es, aus dem aktuellen Zustand die Summe der Belohnungen (Rewards) zu schätzen. Dadurch kann dem Agenten noch mehr Feedback zu seinen getroffenen Entscheidungen gegeben werden. Dies führt dazu, dass der Agent in Zukunft mit höherer Wahrscheinlichkeit die richtigen

Entscheidungen trifft. Gegenüber vergleichbaren Algorithmen besteht PPO aus einfacherem Code und ermöglicht effizientes Sampling und einfaches Parametertuning.

3) *Reward Function*: Yuxi Li beschreibt in ihrem Werk *Deep Reinforcement Learning: An Overview* [4] die Reward Function als eine Funktion, die eine mathematische Formulierung der Belohnungen beschreibt. Oft sind solche Belohnungen sehr spärlich. Beim Lernen des Computerspiels Pong würde man zum Beispiel den Agenten positiv belohnen, wenn der Ball den Schläger des Gegners passiert und negativ belohnen, wenn der Ball den eigenen Schläger passiert. Anhand dieser Rewards kann die Wahrscheinlichkeit für Entscheidungen, die zu einer möglichst hohen Punktzahl führen, erhöht werden, wohingegen die Wahrscheinlichkeit für Entscheidungen, die zu geringer Punktzahl führen, verringert werden. Üblich ist es, die Reward Function durch Reward Shaping anzupassen. Das bedeutet im Beispiel von Pong, dass die letzten Aktionen, bevor es zum Kontakt zwischen Spielball und Schläger kommt, stärker gewichtet werden. Dies sind, letztendlich die Aktionen, die maßgeblich dafür sind ob man den Ball blockt oder nicht. Die Aktionen direkt nach dem Ballkontakt sind im Grunde bedeutungslos. Ein weiteres Beispiel für Reward Shaping ist es, den Agenten dafür zu belohnen, die Spielwelt zu erkunden. Da Reward Shaping jedoch ein sehr aufwändiger Prozess ist, wird dies in der Praxis nicht immer angewandt. So wurde zum Beispiel bei der Arbeit *Playing atari with deep reinforcement learning* [12] kaum Reward Shaping angewandt, da es das Ziel war, dass der Agent möglichst viele verschiedene Spiele lernt und dabei möglichst gut generalisiert.

4) *Self-play experience*: Unter **self-play experience** versteht man, dass der Agent aus Spielen lernt, die er gegen eine Kopie seiner selbst durchführt oder durchgeführt hat. Horgan et al. stellen in ihrem Paper *Distributed prioritized experience replay* [13] eine Methode vor, die es dem Agenten ermöglicht, effektiv aus großen Datensätzen zu lernen. Dies geschieht durch einen Algorithmus, der Handeln von Lernen entkoppelt. Die Agenten interagieren mit eigenen Instanzen der Umgebung und wählen Aktionen, die aus der Policy hervorgehen. Anschließend werden die Erfahrungen in einem gemeinsamen Erfahrungsspeicher gesammelt und das neuronale Netz aktualisiert. Die Netzwerkarchitektur erhält dabei Zugriff auf die gespeicherten Erfahrungen und kann so die Policy verbessern. Das beschriebene Vorgehen erreicht eine bessere Performance als bisherige vergleichbare Verfahren und konnte die gesamte Trainingszeit der Policy deutlich verringern. Es wurde auch von OpenAI in dem Paper verwendet, das in dieser Arbeit vorgestellt wird.

5) *Long Short-Term Memory*: 1997 stellten Hochreiter und Schmidhuber in ihrer Veröffentlichung *Long short-term memory* [14] eine Netzwerkarchitektur vor, durch die es möglich ist, das Problem der verschwindenden Gradienten [15] bei Recurrent Neural Networks (RNN) [16] zu lösen. LSTM sind ein spezieller Typ von RNNs, die langfristige Abhängigkeiten in zeitabhängigen Inputs erkennen und so relevante Informationen speichern und nicht relevante Informationen vergessen können. In Abbildung 2 ist eine LSTM-Zelle dargestellt.

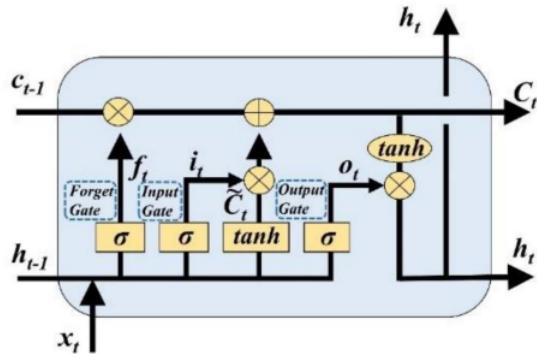


Abbildung 2. Aufbau einer LSTM-Zelle. (Quelle: [17])

6) *Intrinsische Motivation*: In dem Paper *Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation* [18] beschreiben Kulkarni et al., wie mit der Tatsache umgegangen werden kann, dass zielgerichtetes Lernen unter Verwendung von spärlichen Feedbacks problematisch ist. Laut ihnen ist es die Hauptschwierigkeit, dass die Umgebung nicht ausreichend erforscht wird und infolgedessen ein Agent nicht in der Lage ist, eine robuste Wertefunktion zu lernen. Intrinsisch motivierte Agenten können neue Verhaltensweisen um ihrer selbst willen erforschen und nicht, um direkt Probleme zu lösen. Die Autoren stellen ein Framework vor, in dem eine hierarchische Wertefunktionen in den normalen Ablauf des Reinforcement Learnings (siehe Abbildung 1) integriert wird. Dadurch entsteht Raum für die Erforschung komplexer Umgebungen. Somit können umfangreiche Aufgaben, beispielsweise das ATARI Spiel *Montezuma's Revenge*, erfolgreich gelernt werden. In Abbildung 3 ist das Lernschema unter Einbezug von intrinsischer Motivation zu sehen.

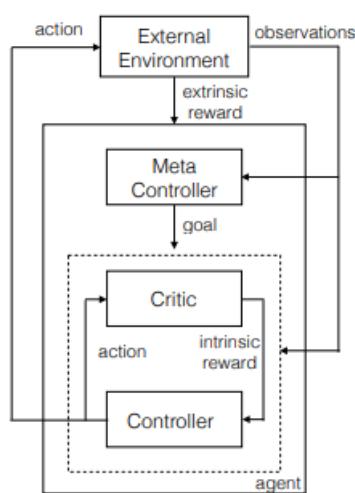


Abbildung 3. Hierarchischer Aufbau des Lernvorgangs bei Verwendung von intrinsischer Motivation. (Quelle: [18])

## B. Robotik

Pieter Abdeel zeigte 2017 während eines Vortrags eine Demonstration, in der Roboter Tätigkeiten, wie zum Beispiel Staubaugen, Getränke holen oder Flaschen öffnen, ausführten. Er zeigte damit, dass Roboter durchaus in der Lage sind, alltägliche Aufgaben zu bewältigen. Nach dieser Demonstration eröffnete er dem Publikum, dass alle Roboter, die man gerade gesehen hat remote gesteuert wurden. Abdeel zeigte so, dass der aktuelle Stand der Technik auf Seite der Hardware ausreichend ist, um Roboter eine Vielzahl von komplexen Tätigkeiten mit hoher Genauigkeit ausführen zu lassen. Das Problem bei der Realisierung stellt aktuell die Software dar. Der Begriff *Roboter* wurde erstmals 1920 in dem Drama *R.U.R - Rossum's Universal Robots* von Karel Čapek verwendet. Er geht auf das westslawische Wort *robota* zurück, was so viel bedeutet wie Fronarbeit. Im allgemeinen Sprachgebrauch versteht man unter Roboter eine Maschine, die bestimmte Funktionen übernehmen kann. Hat der Roboter ein menschenähnliches Aussehen, spricht man auch oft von Androïden. Eine einheitliche Definition des Begriffes gibt es jedoch nicht. Folgend sind zwei mögliche Definitionen des Begriffes *Roboter* aufgeführt:

- Robotics Institute of America (RIA): Ein Roboter ist ein programmierbares Mehrzweckhandhabungsgerät für das Bewegen von Material, Werkstücken, Werkzeugen oder Spezialgeräten. Der frei programmierbare Bewegungsablauf macht ihn für verschiedenste Aufgaben einsetzbar [19].
- VDI (Richtlinie 2860): Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d.h. ohne mechanischen Eingriff) programmierbar und gegebenenfalls sensorgesteuert sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und/oder Fertigungsaufgaben ausführen [20].

Das Themengebiet der **Robotik** befasst sich mit dem Versuch, das Konzept der Interaktion mit der physischen Welt auf Prinzipien der Informationstechnik sowie auf eine technisch machbare Kinetik zu reduzieren. Der Begriff wurde erstmals 1942 von Isaac Asimov in dessen Kurzgeschichte *Runaround* verwendet. Nach dessen Definition beschreibt die Robotik das Studium der Roboter oder auch der Maschinen.

In der Robotik fließen zum Beispiel Ingenieur- und Naturwissenschaften, Maschinenbau, Elektrotechnik und die Informatik ein. Ein Roboter benutzt Sensoren, Aktoren und Informationsverarbeitung, um mit seiner Umgebung zu interagieren. Zu den Sensoren gehören beispielsweise Lasersensoren, Lichtsensoren, Bumper oder Ultraschallsensoren. Aktoren sind jene Bestandteile des Roboters, mit denen physikalische Aktionen durchgeführt werden können. Und durch Informationsverarbeitung werden aus den gesammelten Informationen der Sensoren mittels eines Programms bestimmte Aktionen der Aktoren gewählt, die dann ausgeführt werden.

**Shadow Dexterous Hand** Im Paper *Learning dexterous*

*in-hand manipulation* (siehe IV) verwenden die Entwickler von OpenAI als Hardware eine Roboterhand, die als Shadow Dexterous Hand bezeichnet und von The Shadow Robot Company vertrieben wird, um filigrane Tätigkeiten auszuführen. Die genaue Bezeichnung der verwendeten Roboterhand ist EDC-Hand mit Elektromotoraktoren. Die Hand hat 24 Gelenke zwischen ihren Gliedern und wird von 40 Spectrassehnen gesteuert, die wiederum von 20 Gleichstrommotoren angetrieben werden. 16 Freiheitsgrade können unabhängig voneinander gesteuert werden, während acht Gelenke jeweils aus einem Gelenkpaar bestehen. Außerdem besitzt die Hand fünf Drucksensoren, die allerdings in der Arbeit von OpenAI nicht verwendet werden, da die Position der Gegenstände, die in der Hand gehalten werden, und die Position der Hand durch Kameras bestimmt wird.

### III. OPENAI FIVE

Am 13.04.2019 wurde das amtierende Weltmeisterteam<sup>1</sup> erstmals in einem e-Sports-Wettkampf von einer KI besiegt<sup>2</sup>. Infolgedessen veröffentlichten am 13.12.2019 Berner et al. das Paper *Dota 2 with Large Scale Deep Reinforcement Learning* [1]. In diesem Paper beschreiben sie detailliert die KI - die sie OpenAI Five nennen -, der der angesprochene Erfolg gelang. OpenAI Five wurde in einem einzelnen Trainingsdurchlauf trainiert, der vom 30.06.2018 bis zum 22.04.2019 stattfand. In diesem Training wurde Spielerfahrung im Wert von 180 Jahren durch self-play experience (siehe II-A4) gesammelt. Trainiert wurde auf 256 GPUs und 128.000 CPU-Kernen. Es folgt eine Zusammenfassung der Arbeit von Berner et al., wobei die Struktur im Wesentlichen der des originalen Papers entspricht. Sofern nicht anders gekennzeichnet stammen alle verwendeten Informationen aus der Veröffentlichung von OpenAI.

#### A. Dota 2

Dota 2 ist ein populäres Echtzeitstrategiespiel für mehrere Spieler (Multiplayer Real-Time Strategy Game, kurz: Multiplayer RTS oder MRTS) und erschien im Juli 2013. In dem Spiel geht es darum, dass auf einer quadratischen Karte zwei Teams jeweils bestehend aus fünf Spielern Basen in gegenüberliegenden Ecken verteidigen. In jeder Basis befindet sich ein Bauwerk, genannt Antike (Ancient). Um ein Spiel zu gewinnen, muss ein Team die Antike des gegnerischen Teams zerstören. Jeder Spieler kontrolliert eine Heldeneinheit (Hero). Außerdem verfügen beide Teams über einen Strom von Einheiten (Creeps). Sie werden von keinem Spieler direkt kontrolliert. Die Creeps bewegen sich auf die gegnerische Basis zu und greifen alle gegnerischen Einheiten und Gebäude an. Es gibt auch Creeps, die zu keinem Team gehören. Wenn Spieler Creeps besiegen, können sie Ressourcen, wie zum Beispiel Gold, sammeln und dadurch die Macht ihrer Helden erhöhen, indem sie bestimmte Gegenstände kaufen oder Fähigkeiten

<sup>1</sup>Teamname: OG, <https://liquipedia.net/dota2/OG>, aufgerufen am 02.07.2023

<sup>2</sup>Artikel zum Verlauf des Wettkampfes: <https://openai.com/blog/openai-five-defeats-dota-2-world-champions/>, aufgerufen am 02.07.2023

verbessern.

Mittlerweile ist Dota 2 eines der am meisten gespielten Videospiele unserer Zeit. Außerdem wurden mit über 310 Millionen US-Dollar mehr Preisgelder ausgeschüttet als in jedem anderen E-Sport-Titel<sup>3</sup>. Verglichen mit bisherigen Meilensteinen auf dem Gebiet der Brett- und Videospiele, wie zum Beispiel Backgammon (1992), Schach (1997), Atari (2013) oder auch Go (2016), weist Dota 2 eine wesentlich höhere Komplexität auf. Neben diesen Argumenten sind vor allem die folgenden drei Gründe relevant für das große Interesse an Multiplayer RTS im Forschungsgebiet DRL.

- **Long time horizons.** Ein Spiel dauert durchschnittlich 45 Minuten bei 30 Frames pro Sekunde. Die KI trifft jeweils nach 4 Frames eine Entscheidung. Daraus ergeben sich durchschnittlich etwa 20.000 Entscheidungen pro Spiel. Im Vergleich dazu sind es bei Schach 80 und bei Go 150 Entscheidungen pro Spiel.
- **Partially-observed state.** Jedes Team kann nur einen eingeschränkten Bereich des Spielfeldes (Map) sehen. Cleveres Spiel erfordert Entscheidungen auf der Grundlage unvollständiger Informationen und der Einschätzung des gegnerischen Verhaltens. Im Vergleich dazu hat man bei Go und Schach jederzeit alle Informationen über alle Spielsteine, die sich auf dem Spielfeld befinden.
- **High-dimensional action and observation spaces.** Auf dem Spielfeld von Dota 2 befinden sich zehn Helden, einige Gebäude, weitere Einheiten und zusätzliche Spielemente wie Runen, Bäume oder Seen. Insgesamt beobachtet OpenAI Five ungefähr 16.000 veränderliche Werte<sup>4</sup> pro Zeitschritt (Timestep). In einem Zeitschritt entscheidet sich das Modell - abhängig vom gewählten Helden - zwischen 8.000 und 80.000 möglichen Aktionen. Zum Vergleich benötigt Schach etwa 1000 Werte pro Beobachtung (meist kategorische Werte mit 6 Möglichkeiten) und Go etwa 6000 Werte (alle binär). Bei Schach gibt es etwa 35 gültige Aktionen und bei Go etwa 250 gültige Aktionen pro Zug.

Das System von OpenAI wurde mit zwei Einschränkungen gegenüber dem eigentlich Spiel gespielt:

- **Subset of 17 heroes.** Normalerweise kann ein Spieler vor dem Spiel aus einem Pool von 117 Helden wählen. Für die KI wurde diese Zahl auf 17 reduziert.
- **No support for certain items.** Items, die dazu führen, mehrere Einheiten zur gleichen Zeit steuern zu können, werden nicht unterstützt. Das vermeidet einen erhöhten technischen Aufwand bei der Umsetzung.

#### B. Training des Systems

In den folgenden Abschnitten wird beschrieben, wie aus der schwammigen Formulierung “Lerne, dieses komplexe Spiel auf übermenschlichem Niveau zu spielen“ ein konkretes, objektiv evaluierbares Optimierungsproblem wird. Im

<sup>3</sup>Quelle: <https://www.esportearnings.com/games>, aufgerufen am 02.07.2023

<sup>4</sup>Überwiegend Float und kategorische Werte

Anschluss daran folgt der Lösungsansatz von OpenAI für dieses Problem. Das Kapitel unterteilt sich in die Abschnitte *Observation*, *Policy*, *Optimierung der Policy* und kontinuierlicher Transfer durch *Surgery*.

**1) Observation:** Normalerweise erfolgt die Interaktion eines Menschen mit Dota 2 durch Tastatur, Maus und Bildschirm. Die in Echtzeit getroffenen Entscheidungen der Spieler können den Spielverlauf langfristig beeinflussen.

OpenAI Five erhält die Beobachtungen (Observations) des Spiels nicht aus der graphischen Darstellung, also der Pixel auf dem Bildschirm, sondern in Form einer Vektordarstellung (Data Array/ Observation Array). In diesem Vektor sind sämtliche Informationen des aktuellen Spiels enthalten. Dazu zählen beispielsweise Informationen über die Helden (Gesundheitszustand, Position), Creeps usw. Eine Darstellung des vollständigen Observation Arrays bietet Abbildung 4. Bei der Vektordarstellung der Informationen ist es der KI möglich, maximal so viele Informationen über das Spiel zu erlangen, wie ein Mensch beim Spielen durch die graphische Oberfläche erhalten würde. Umgekehrt wurden jedoch nicht alle Informationen, die ein Mensch dem Spiel entnehmen könnte, auch in der Vektordarstellung kodiert. Besonders wichtig ist den Entwicklern, dass die KI durch die Vektordarstellung keinen Vorteil gegenüber den menschlichen Konkurrenten erhält.

Global data	22
time game started	1
is it day or night?	1
time to next day/night change	2
time to next spawn: creep, neutral, bounty, runes	4
time since seen enemy courier is that > 40 seconds? <sup>a</sup>	2
min/max time to Rosh spawn	2
Roshan's current max hp	1
is Roshan definitely alive?	1
is Roshan definitely dead?	1
Next Roshan drops cheese?	1
Next Roshan drops refresher?	1
Roshan health randomization <sup>b</sup>	1
Glyph cooldown (both teams)	2
Stock counts	4
<b>Per-unit (189 units)</b>	<b>43</b>
position (x, y)	2
facing angle (cos, sin)	2
currently attacking?	2
time since last attack <sup>d</sup>	2
max health	1
last 16 timesteps' hit points	17
attack damage, attack speed	2
physical resistance	1
invulnerable due to glyph?	1
glyph timer	2
movement speed	1
on my team? / neutral?	1
animation cycle time	1
eta of attacking tower & tower creep hitting this unit <sup>e</sup>	2
# miles creeps atting this unit <sup>d</sup>	3
[Shrine only] shrine cooldown	1
vector to me (dx, dy, length) <sup>c</sup>	3
am I attacking this unit? <sup>d,e</sup>	2
is this unit attacking me? <sup>d,e</sup>	1
eta projectile from unit to me <sup>e</sup>	3
unit type	1
current animation	1
<b>Per-hero add'l (10 heroes)</b>	<b>25</b>
is currently alive?	1
number of deaths	1
hero currently in sight?	1
time since this hero last seen	2
hero currently teleporting?	1
if so, target coordinates (x, y) time they've been channeling	4
respawn time	2
current gold (allies only)	1
level	1
mana: max, current, & regen	3
health regen rate	1
magic resistance	1
strength, agility, intelligence	3
current invincibility?	1
is invincibility ability?	1
# allied/enemy creeps/heroes in line btwn me and this hero <sup>e</sup>	4
<b>Per-allied-hero additional</b>	<b>211</b>
(5 allied heroes)	211
Scripted purchasing settings <sup>b</sup>	7
Buyback: has?, cost, cooldown	3
Empty inventory & backpack slots	2
Lane Assignments <sup>b</sup>	3
Flattened nearby terrain: 14x14 grid of passable/impassable?	196
scripted build id	2
next item to purchase <sup>b</sup>	2
Nearby map (8x8) <sup>e</sup>	6
terrain: elevation, passable?	2
# allied & enemy creep density	2
area of effect spells in effect. <sup>f</sup>	2
area of effect spells in effect. <sup>f</sup>	2
<b>Previous Sampled Action<sup>g</sup></b>	<b>310</b>
Offset? (Regular, Caster, Ward)	3x2x9
Unit Target's Embedding	128
Primary Action's Embedding	128

Abbildung 4. Vollständiger Observation Array von OpenAI Five. (Quelle: [1])

Außerdem schreiben sie in ihrem Paper, dass es durchaus möglich gewesen wäre, die KI so zu trainieren, dass sie dazu im Stande wäre, Informationen aus den Pixeln auslesen zu können, dies aber nicht das primäre Ziel ihrer Arbeit sei. Der Fokus ihrer Arbeit liegt nicht darin, visuelle Aufbereitung (visual processing) zu optimieren, sondern die KI zu trainieren, strategisch richtige Entscheidungen zu treffen. Außerdem wäre das Rendern der einzelnen Frames mit einem extrem hohen technischen Aufwand verbunden und daher sehr kostspielig. Obwohl Dota 2 mit 30 Frames pro Sekunde läuft, agiert

OpenAI Five bloß jeden vierten Frame. Diese Spanne wird als *Timestep* bezeichnet (siehe Abbildung 5)

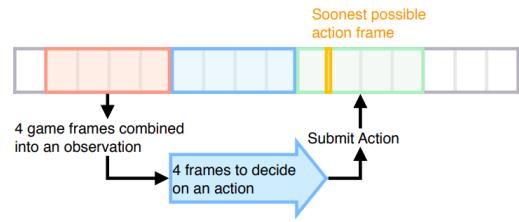


Abbildung 5. Zeitschritt in OpenAI Five. (Quelle: [1])

**2) Policy:** Aus dem Paper geht hervor, dass manche Abläufe per Hand geskriptet wurden. Dazu zählen die Reihenfolge, in der Helden Items und Fähigkeiten kaufen, die Kontrolle über die Unique Courier Unit und die Entscheidung, welche Items der Held als Reserve behält.

Alle anderen Entscheidungen im Spiel werden wie folgt getroffen: Es gibt eine Funktion, die als **Policy** bezeichnet wird (siehe II-A1). Diese erstellt basierend auf den Beobachtungen eine Wahrscheinlichkeitsverteilung der möglichen nächsten Aktionen der KI. Die Policy ist durch ein neuronales Netz (LSTM (siehe II-A5)) realisiert und hat etwa 159 Millionen Parameter. In diesem Netz werden die aktuellen Beobachtungen (also der Data Array der Observations) als Input verwendet und dann eine Aktion aus der Verteilung der möglichen Aktionen, die von der Policy erstellt wurde, gesampelt. Diese Aktion wird dann von der KI ausgeführt. Da OpenAI Five insgesamt fünf Helden steuern muss, werden auch fünf Policies trainiert und verwendet. In Abbildung 6 ist der Datenfluss vereinfacht dargestellt. Da die fünf Spieler über eine gemeinsame Sicht verfügen, ist auch der Observation Array identisch. Unterschiede gibt es nur in den gesteuerten Helden. Dafür werden spezielle Hero Embeddings verwendet. Anschließend werden die Beobachtungen und die Kenntnisse über den gesteuerten Helden (Hero Embeddings) als Eingabe für das LSTM verwendet, um die Value Function und die Gewichte der Policy zu aktualisieren.

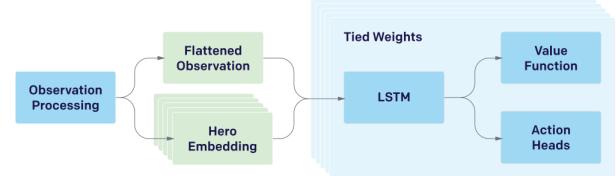


Abbildung 6. Vereinfachte Darstellung der Netzwerkarchitektur. (Quelle: [1])

**3) Optimierung der Policy:** Das Ziel der Policy ist es, die Gewinnwahrscheinlichkeit der KI zu maximieren. Dafür ist es notwendig, eine möglichst passende Reward Function (Belohnungsfunktion) (siehe II-A3) zu finden. Die Belohnungsfunktion ist aus den Erfahrungen der Mitarbeiter von OpenAI erstellt worden, die bereits Dota 2 gespielt haben

(siehe Abbildung 7). Ein erster Entwurf stellte sich schon als sehr brauchbar heraus, sodass nur noch kleine Änderungen vorgenommen werden mussten.

Name	Reward	Heroes	Description
Win	5	Team	
Hero Death	-1	Solo	
Courier Death	-2	Team	
XP Gained	0.002	Solo	
Gold Gained	0.006	Solo	For each unit of gold gained. Reward is not lost when the gold is spent or lost.
Gold Spent	0.0006	Solo	Per unit of gold spent on items without using courier.
Health Changed	2	Solo	Measured as a fraction of hero's max health. <sup>‡</sup>
Mana Changed	0.75	Solo	Measured as a fraction of hero's max mana.
Killed Hero	-0.6	Solo	For killing an enemy hero. The gold and experience reward is very high, so this reduces the total reward for killing enemies.
Last Hit	-0.16	Solo	The gold and experience reward is very high, so this reduces the total reward for last hit to ~ 0.4.
Deny	0.15	Solo	
Gained Aegis	5	Team	
Ancient HP Change	5	Team	Measured as a fraction of ancient's max health.
Megas Unlocked	4	Team	
T1 Tower*	2.25	Team	
T2 Tower*	3	Team	
T3 Tower*	4.5	Team	
T4 Tower*	2.25	Team	
Shrine*	2.25	Team	
Barracks*	6	Team	
Lane Assign†	-0.15	Solo	Per second in wrong lane.

Abbildung 7. Bei OpenAI Five verwendetes Belohnungssystem. (Quelle: [1])

Die Policy wird mittels Proximal Policy Optimization (siehe II-A2) trainiert. Der verwendete Optimierungsalgorithmus ist Generalized Advantage Estimation (GAE) [21] mit Adam [22] als Optimierer.

Das Modell wurde durch gesammelte Self-Play Erfahrung (siehe II-A4), ähnlich wie im Paper *Distributed prioritized experience replay* [13] beschrieben, trainiert. Ein zentraler Pool von Optimierungs-GPUs empfängt Spieldaten und speichert sie asynchron in lokalen Puffern, den so genannten Erfahrungspuffern (experience buffer). Jede Optimierungs-GPU berechnet dann Gradienten anhand von Minibatches, die nach dem Zufallsprinzip aus dem Erfahrungspuffer entnommen werden. Die Gradienten werden mit NCCL<sup>25</sup> über den Pool gemittelt, bevor sie synchron auf die Parameter angewendet werden. Auf diese Weise errechnet sich die effektive Batchgröße aus der Batchgröße auf jeder GPU (120 Samples, jedes mit 16 Zeitschritten) multipliziert mit der Anzahl der GPUs (maximal 1536), was einer gesamten Batchgröße von 2.949.120 Zeitschritten entspricht. Jeweils nach 32 Gradientenschritten übergibt der Optimierer eine neue Version der Parameter an einen zentralen Speicher - den Controller. Der Controller speichert außerdem Metadaten der Zustände des gesamten Systems, falls Trainingsläufe gestoppt oder neu gestartet werden.

**Rollout Worker Machines** spielen Spiele gegen sich selbst. Diese Spiele werden bei der Hälfte des originalen Tempos gespielt. Es hat sich gezeigt, dass so Informationen mit der höchsten Effizienz für das Training gewonnen werden können,

<sup>5</sup>NVIDIA NCCL: <https://developer.nvidia.com/nccl>, aufgerufen am 02.07.2023

da es auf diese Weise möglich ist, mehr als doppelt so viele Spiele gleichzeitig durchzuführen, wie es im Originaltempo machbar wäre. In den Spielen wird immer die aktuellste Policy verwendet. Diese spielt in 80% der Fälle gegen sich selbst und sonst gegen ältere Versionen seiner selbst. Rollout Machines fragen häufig den Controller ab um die neuesten Parameter für das LSTM zu erhalten und senden Daten von Spielen die gerade laufen. Das gesamte System läuft auf Rapid, einer speziell für OpenAI Five entwickelten Trainingsplattform, welche wiederum in der Google Cloud betrieben wird. Abbildung 8 gibt einen Überblick über das gesamte Trainingssystem.

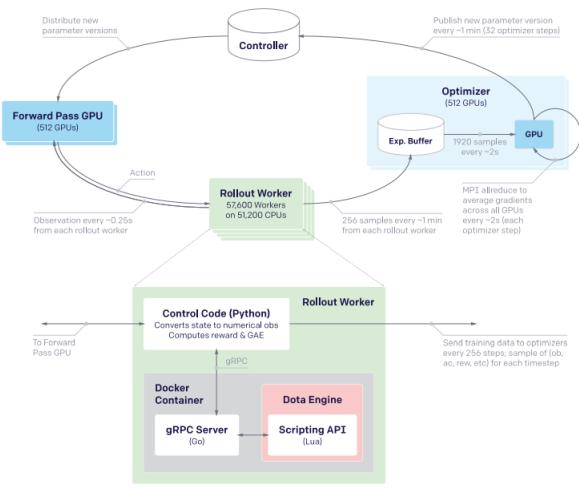


Abbildung 8. Überblick über das gesamte Trainingssystem von OpenAI Five. (Quelle: [1])

4) *Surgery*: Das Training des Modells erfolgte über einen Zeitraum von zehn Monaten. Während dieser Zeit mussten immer wieder Anpassungen an eine sich veränderte Umgebung gemacht werden. Dies geschah aus den folgenden drei Gründen:

- Optimierung des Trainingsprozesses. Anpassung wichtiger Elemente, wie zum Beispiel der Reward Function oder der Architektur der Policy.
- Erweiterung der möglichen Aktionen des Agents mit zunehmender Trainingsdauer.
- Anpassung an neue Versionen von Dota 2, die von Zeit zu Zeit von Valve veröffentlicht wurden.

Laut OpenAI ist es der beste Weg, auf diese Veränderungen mit einem Neustart des Trainings zu reagieren. Da dies aber ein sehr kostspieliges Unterfangen ist, wurden Tools entwickelt, um während des Trainings wichtige Elemente des gesamten Prozesses mit minimalen Verlusten in der Performance zu verändern. Diese Eingriffe werden als *Surgery* bezeichnet. Insgesamt gab es mehr als 20 Surgeries, die in einem etwa zweiwöchigen Rhythmus durchgeführt wurden. Bei einer Surgery werden offline Veränderungen an einem alten Modell vorgenommen, um ein neues Modell zu erhalten, das mit seiner aktuellen Umgebung wieder kompatibel ist. Dabei ist entschei-

dend, dass sich die Dimensionen der Vektoren, in denen die Parameter gespeichert sind, sich verändern können, das Modell aber trotzdem auf dem gleichen Niveau wie zuvor agieren kann. Im einfachsten Fall, in dem sich weder Umgebungs-, Beobachtungs- und Aktionsraum verändern, wird die Funktion der Policy direkt übernommen und es ändert sich nichts.

### C. Ergebnisse

Wie bereits in III erwähnt, gelang es OpenAI Five, das Team OG in einem Wettkampf zu besiegen, der live übertragen wurde. Damit ist OpenAI Five die erste KI, der ein vergleichbarer Erfolg gelang. Der Verlauf des erreichten spielerischen Niveaus - über der Trainingszeit gemessen in Gleitkommazahlen-Operationen - ist in Abbildung 9 dargestellt. Das spielerische Niveau wird in einem eigens entwickelten TrueSkill-Rating gemessen. Einem zufälligen Agent wird ein Wert von 0 als Referenz zugewiesen. Erhöht man den TrueSkill-Wert um 8,3, so entspricht das einer Gewinnrate von 80% gegenüber dem Referenz-Agent.

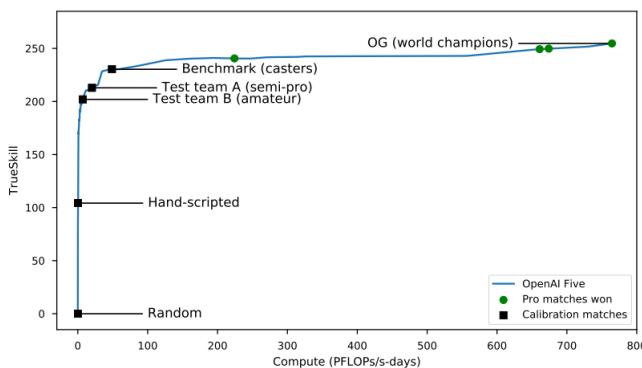


Abbildung 9. Entwicklung des Spielniveaus von OpenAI Five - über der Trainingszeit gemessen in Gleitkommazahlen-Operationen. (Quelle: [1])

Vom 18.04.2019 bis zum 21.04.2019 wurde OpenAI Five der Öffentlichkeit als Gegner zur Verfügung gestellt. Ziel war es, festzustellen, wie sich die KI bewährt, wenn sie auf Strategien trifft, die sich Spieler extra zurechtgelegt haben, um OpenAI Five zu besiegen. In dem Zeitraum wurden 7257 Spiele gespielt, von denen die KI 42 verlor. Das entspricht einer Gewinnrate von 99,4 Prozent.

In Dota 2 ist es üblich, die Geschicklichkeit eines Spielers durch seine Reaktionszeit auszudrücken. Durchschnittlich liegt die Reaktionszeit von Menschen bei 250 ms. OpenAI Five besitzt eine mittlere Reaktionszeit von 217 ms.

Der Spielstil der KI ist schwierig einzuordnen und stark durch das Reward Shaping bei der Belohnungsfunktion beeinflusst. Am Anfang des Trainings bevorzugte die KI frühe, große Kämpfe und hatte eine sehr hohe Gewinnwahrscheinlichkeit, wenn der Gegner solche Kämpfe annahm. Vermied der Gegner diese Kämpfe, so konnte sich die KI kaum von einem daraus entstandenen Rückstand erholen. Spätere Policies näherten sich dem Spielstil eines Menschen an. Die KI lernte, Ressourcen geschickt zu verwenden und die Möglichkeiten ihres stärksten Helden so bestmöglich einzusetzen. Auch das Ausnutzen

taktischer Vorteile und den strategisch cleveren Umgang mit einem Rückstand lernte die KI. Die Strategie der finalen Version ähnelte sehr stark der eines professionellen Spielers, wobei einige interessante Abweichungen erkennbar sind. Das Markanteste ist die Tendenz der KI dazu, wesentlich öfter die Position der Helden zu verändern, wohingegen Menschen dazu tendieren, ihre Helden in bestimmten Bereichen auf dem Spielfeld zu positionieren. Außerdem wurde beobachtet, dass Menschen des Öfteren geschwächte Helden aus dem Kampf zurückziehen, um sie zu heilen, wohingegen OpenAI Five ein sehr gutes Verständnis dafür entwickelt hat, wann es sich lohnt, auch mit einem geschwächten Helden einen Kampf zu riskieren, um daraus einen langfristigen Vorteil zu kreieren. Letztlich zeigt die KI eine hohe Bereitschaft, Ressourcen und Fähigkeiten mit langen Abklingzeiten früh zu verbrauchen. Menschen hingegen tendieren dazu, diese länger zu behalten, um auf eine gute Gelegenheit zu warten, um sie einzusetzen.

### D. Fazit zu OpenAI Five

Insgesamt konnten die Mitarbeiter von OpenAI in ihrem Paper zeigen, dass künstliche Systeme durch moderne Techniken des Reinforcement Learnings übermenschliche Fähigkeiten im Bereich der kompetitiven E-Sport-Games lernen können. Voraussetzung ist, dass diese richtig skaliert werden. Die wichtigsten Aspekte hierbei sind die Erweiterung der Rechenleistung durch Vergrößerung der Batch Size und der gesamten Trainingszeit. Eine Erhöhung der Trainingszeit wurde durch die Surgery realisiert. Dadurch konnte das Training trotz Veränderungen an Umgebung und Modell fortgesetzt werden. Die Autoren vermuten, dass neue Aufgaben weiter an Komplexität zunehmen werden und die Skalierung der bestehenden Methoden noch wichtiger wird.

## IV. LEARNING DEXTEROUS IN-HAND MANIPULATION

Am 18.01.2019 veröffentlichten Andrychowicz et al. das Paper *Learning Dexterous In-Hand Manipulation* [2]. Darin zeigten die Mitarbeiter von OpenAI, dass mit Hilfe von Reinforcement Learning die geschickte Manipulation eines Gegenstandes in einer Roboterhand gelernt werden kann. Das gesamte System nennen sie *Dactyl*. Die Policy von Dactyl wurde komplett in einer simulierten Umgebung trainiert und dann auf die Shadow Dexterous Hand<sup>6</sup> (siehe II-B) in der Realität angewandt. Obwohl beim Training keine menschlichen Demonstrationen verwendet wurden, lernte das Modell typisch menschliche Bewegungen, wie zum Beispiel finger gaiting (Fingerfertigkeit), multi-finger coordination (Koordination mehrerer Finger) oder das kontrollierte Nutzen der Schwerkraft. Es folgt eine Zusammenfassung der Arbeit von Andrychowicz et al. Sofern nicht anders gekennzeichnet sind alle Informationen der Arbeit von Andrychowicz et al. entnommen.

<sup>6</sup>Quelle: <https://www.shadowrobot.com/dexterous-hand-series/>, aufgerufen am 02.07.2023

## A. Task and System Overview

Bei dem Forschungsprojekt ging es darum, ein Problem der Objekt-Reorientierung in einer Roboterhand zu lösen. Dabei wird in der Hand ein Würfel mit verschiedenen Buchstaben als Aufschrift positioniert. Dieser muss dann durch geschickte Rotation in eine bestimmte Zielorientierung gebracht werden. Ist dieses Ziel erreicht, gibt es eine neue Position, in die der Würfel rotiert werden muss.

Um das Problem zu lösen, wurden zwei Systeme trainiert. Mit dem ersten System wurde eine Policy trainiert, die dazu genutzt wurde, zu lernen, die Roboterhand zu kontrollieren. Dafür wurden zwei Netzwerke gleicher Architektur trainiert: Policy Network und Value Network. Die beiden Netzwerke unterscheiden sich lediglich in ihren Parametern. Das zweite System hat die Aufgabe, die Position und die Orientierung des Objektes in der Hand durch die Auswertung von drei RGB-Kameras zu bestimmen. Dieses System ist sozusagen das Auge der Roboterhand. Diese Vorhersage wird in der realen Anwendung als Input für die Policy verwendet. In Abbildung 10 ist ein Überblick über das gesamte verwendete System dargestellt.

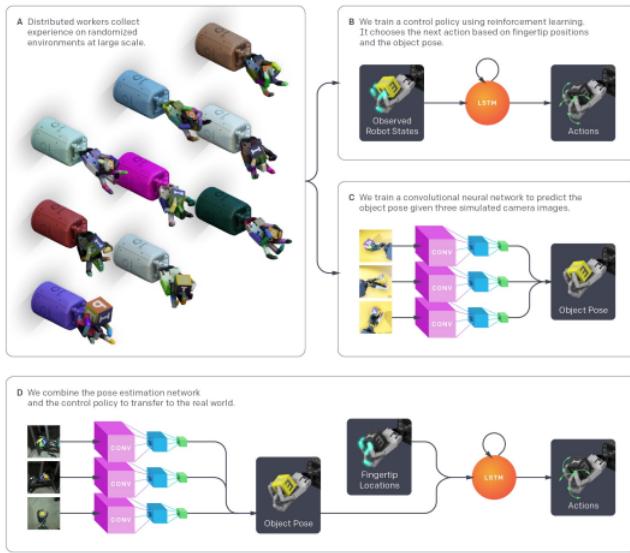


Abbildung 10. Überblick über das gesamte System. (Quelle: [2])

## B. Hardware

Die Umsetzung in der realen Welt gelingt durch eine Shadow Dexterous Hand (siehe II-B). Mit Hilfe des PhaseSpace Motion-Capture-Systems wird die Position der Fingerspitzen der Roboterhand, ausgedrückt in kartesischen Koordinaten, verfolgt. Die Position des Objektes wird durch zwei verschiedene Setups überwacht. Das erste Setup besteht aus einem Phase-Space-Marker. Das zweite Setup besteht aus drei Basler RGB-Kameras, um die Position und Rotation basierend auf einer optischen Auswertung vorherzusagen. Die Berührungssensoren der Hand werden nicht verwendet, da sich

gezeigt hat, dass eine optische Auswertung robuster in Bezug auf physikalische Veränderungen in der Umgebung ist.



Abbildung 11. Vergleich zwischen realer und simulierter Umgebung. (Quelle: [2])

## C. Simulation

Die Simulation findet in der MuJoCo-Physics-Engine statt. Für das Rendern der Bilder, um Positionen visuell zu erkennen, wird Unity verwendet. Das virtuelle Modell der Roboterhand basiert auf einer für dieses Problem optimierten Version einer Hand aus dem Repertoire von OpenAI Gym robotics environments.

Obwohl die Roboterhand kalibriert wurde, zeigte sich, dass es eine Abweichung zwischen Simulation und Realität gibt. Diese bezeichnet man als *reality gap*. Ursache dafür ist, dass in dem Bodysimulator von OpenAI nur begrenzte Anpassungen möglich sind. Ein weiterer Grund für die Lücke zwischen Realität und Simulation ist die Komplexität der Realität, die nicht gänzlich in der Simulation nachgestellt werden kann. Um dem Problem der Reality Gap entgegenzuwirken, sollte man idealerweise das Training in der realen Welt durchführen. Dieser Ansatz steht aber im Widerspruch zur eigentlichen Idee des Deep Reinforcement Learnings, da dadurch viel zu wenig Erfahrung gesammelt werden kann. Ein Training in der Simulation wäre nur dann sinnvoll, wenn die Realität exakt abgebildet werden könnte. Da dies nicht der Fall ist, erhält man eine Policy, die sehr stark overfittet. Die Lösung von OpenAI ist es, Duplikationen der Roboterhand mit Modifikationen der ursprünglich erstellten Version zu verwenden. Zum Beispiel variierte man physikalische Parameter. Eine Auflistung der veränderten physikalischen Parameter ist in Abbildung 12 zu sehen. Da in verschiedenen, randomisierten Umgebungen trainiert wird, muss das Modell eine allgemeinere Policy lernen und es kann Overfitting vermieden werden. Dies führt wiederum dazu, dass der Transfer der Policy von der Simulation zur Realität besser gelingt.

Insgesamt wurden folgende Klassen von Parametern randomisiert:

- **Observation noise.** Um die erwarteten Verzerrungen der realen Beobachtungen durch die Kameras zu simulieren,

Parameter	Scaling factor range	Additive term range
object dimensions	uniform([0.95, 1.05])	
object and robot link masses	uniform([0.5, 1.5])	
surface friction coefficients	uniform([0.7, 1.3])	
robot joint damping coefficients	loguniform([0.3, 3.0])	
actuator force gains (P term)	loguniform([0.75, 1.5])	
joint limits		$\mathcal{N}(0, 0.15) \text{ rad}$
gravity vector (each coordinate)		$\mathcal{N}(0, 0.4) \text{ m/s}^2$

Abbildung 12. Physikalische Randomisierung. (Quelle: [2])

wurde den Beobachtungen der Policy Gaußsches Rauschen hinzugefügt.

- **Physics randomizations.** Physikalische Parameter wurden zu Beginn jeder Episode verändert (siehe Abbildung 12).
- **Unmodeled effects.** In der Realität zeigte sich, dass beispielsweise das Motion Capture Setup kurzzeitig den Marker verlor. Solche Effekte wurden durch zufällige, kurzzeitige Verzögerung der Aktion in der Simulation implementiert.
- **Visual appearance randomizations.** Visuelle Effekte, wie zum Beispiel Kamerapositionen, verschiedene Beleuchtungen oder die Position und Farbe von Hand und Objekt, wurden variiert (siehe Abbildung 13).



Abbildung 13. Visuelle Randomisierung: Variation von Kamerawinkel, Farben und Beleuchtung. (Quelle: [2])

#### D. Control Policy

Das Lernen der Policy baut auf den Erkenntnissen der Arbeit zu OpenAI Five auf. Auch in dieser Arbeit wurde Proximal Policy Optimization [11] (siehe II-A2) verwendet. Es gibt zwei Netzwerke, die trainiert werden: das Policy Network und das Value Network. Diese unterscheiden sich nur in ihren Parametern, ihre Architektur ist identisch.

Aufgabe der Policy ist es, die nächste Aktion der Roboterhand zu bestimmen. Das bedeutet, dass die Drehung der Gelenkwinkel relativ zum aktuellen Stand bestimmt werden muss. Obwohl PPO sowohl kontinuierliche Werte als auch diskrete Werte vorhersagen könnte, zeigte sich in der Anwendung, dass die Ergebnisse unter Verwendung der diskreten Vorhersagen besser geeignet sind.

Der Reward berechnet sich aus der Summe der Abweichung

des aktuellen Rotationswinkels zum Rotationswinkel des Zielobjektes (in Rad) vor und nach einer Veränderung. Zusätzlich wird das Erreichen des Zielzustandes mit +5 und ein Fallenlassen des Objektes mit -20 bewertet. Auch in dieser Anwendung soll der Wert der Reward Function maximiert werden.

#### E. State Estimation from Vision

Die Policy des beschriebenen Systems (siehe IV-D) erfordert als Eingabe die genaue Position des Objekts. Dies könnte theoretisch mithilfe des Motion Capture Systems von Phase Space erreicht werden. Jedoch ist dieses System nicht optimal für das langfristige Ziel von OpenAI geeignet, ein hochgeneralisierbares System zu entwickeln. Das liegt daran, dass das Motion Capture System eine spezielle Laborumgebung erfordert, in der Marker am Objekt angebracht werden müssen. Das ultimative Ziel besteht jedoch darin, dass das System mit beliebigen Objekten funktioniert, ohne dass eine spezielle Vorbereitung erforderlich ist.

Um dieses Problem zu umgehen, wurde ein zweites System trainiert, welches ausschließlich für die Erkennung der Position des Objektes verwendet wird. Dafür wurden aufgenommene Photos von drei verschiedenen RGB-Kameras mit Hilfe eines Convolutional Neural Networks dazu verwendet, die Position und die Rotation des Objektes vorherzusagen. Die Netzwerkarchitektur ist in Abbildung 14 dargestellt. Ebenso wie die Control-Policy wurde auch das System zur State-Estimation (Zustands-Schätzung) ausschließlich unter Verwendung von Daten trainiert, die in der Simulation entstanden sind.

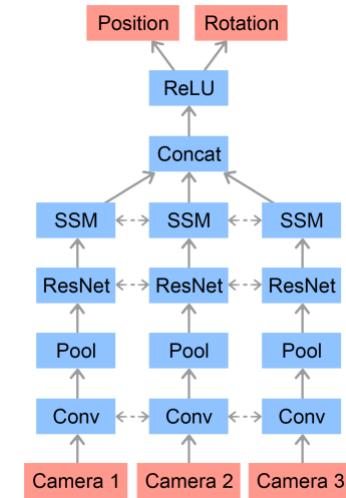


Abbildung 14. Verwendete Netzwerkarchitektur, mit der die Photos von drei Kameras ausgewertet wurden, um die Position des Objektes vorherzusagen. (Quelle: [2])

#### F. Ergebnisse

Während den Versuchen konnte beobachtet werden, dass die Roboterhand natürliche Greifmechanismen, wie man sie auch beim Menschen sieht, gelernt hat. In Abbildung 15

sind Beispiele für Greifmuster zu sehen. Bei sehr genauen Griffen tendiert der Mensch dazu, Zeige- oder Mittelfinger zu verwenden. Die KI hingegen nutzt überwiegend den kleinen Finger. Das liegt daran, dass die Shadow Dexterous Hand so konstruiert ist, dass der kleine Finger einen weiteren Freiheitsgrad gegenüber Zeige-, Mittel- und Ringfinger hat. Das bedeutet, dass die KI menschliche Techniken entdeckt und dann angepasst an ihren Körper adaptiert hat.

Eine weitere auffällige Gemeinsamkeit zwischen Menschen und der Policy ist finger pivoting. Das ist eine Technik, bei der das Objekt zwischen zwei Fingern eingeklemmt wird, um es anschließend zu rotieren.

Auf der anderen Seite wurden auch Fehler, die die Roboterhand machte, genauer untersucht. Häufigste Fehlerursache war, dass das Objekt fiel, wenn das Handwurzelgelenk nach unten rotiert wurde. Daraufhin wurden Experimente durchgeführt, bei denen diese Bewegung gesperrt wurde. Das Ergebnis war, dass menschliche Greifmuster - in der von der Policy gelernten Art und Weise - häufiger auftraten, um das Problem zu lösen. Weitere Fehlermuster waren, dass das Objekt kurz nach dem Start fallen gelassen wurde oder sich das Objekt in einem Schraubenloch verfangen hat. Im ersten Fall wird vermutet, dass etwas bei der Identifikation der Umgebung nicht funktionierte, und der zweite Fall wurde nicht simuliert. Dadurch konnte das Modell keine Strategie lernen, um mit diesem Problem umzugehen.

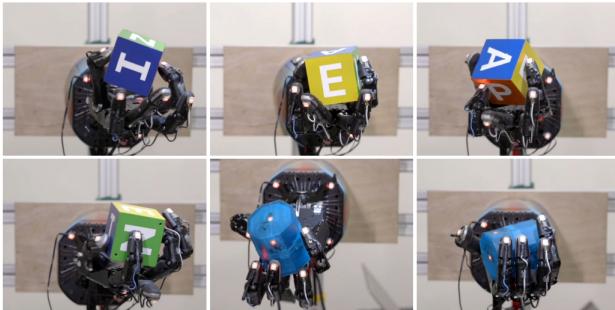


Abbildung 15. Beobachtete Greifmuster. (Quelle: [2])

Neben den genannten qualitativen Ergebnissen gibt es auch eine quantitative Auswertung. Dafür wurde die Anzahl der erfolgreichen Rotationen gemessen, bis das Objekt fallen gelassen, ein Ziel in 80 Sekunden nicht erreicht wurde oder 50 erfolgreiche Rotationen durchgeführt wurden. Durch die Ergebnisse (siehe Abbildung 16) ist es möglich, die Performance der Roboterhand in der Simulation mit der Performance in der Realität zu vergleichen. Daraus lässt sich schließen, dass die Reality Gap zwar durch Kalibrierung und Randomisierung verringert werden konnte, aber immer noch existiert. Dies führt dazu, dass das Modell in der Realität immer schlechter abschneidet als in der Simulation. Wird die Position visuell bestimmt, führt das in der Simulation und der realen Welt zu etwas schlechterer Performance. Das lässt sich darauf

zurückführen, dass die Policy auch in der Simulation einen Transfer durchführen muss, da die Bilder mit Unity gerendert wurden, aber MuJoCo-Rendering für die Evaluation in der Simulation verwendet wurde. Beim realen Roboter schneidet das Vision Model etwas schlechter ab als die Positionsermittlung mit PhaseSpace. Der Unterschied ist aber so klein, dass es ausreichend ist, das Vision Model ausschließlich in der Simulation zu trainieren.

Neben der Verwendung des Würfels als Objekt wurden auch Experimente mit einem achteckigen Prisma durchgeführt. So wurde ein Feintuning der Roboterhand erreicht. Aufgebaut wurde auf der Policy, die lernte, den Würfel zu rotieren. Dieses Experiment zeigt, dass der Lerntransfer möglich ist, und auch erfolgreich durchgeführt werden konnte. In Abbildung 15 ist dieses Experiment in den letzten beiden Photos zu sehen.

Ein drittes Experiment, bei dem eine Kugel als Objekt verwendet wurde, führte zu keinen erfolgreichen Ergebnissen. Das liegt daran, dass auf der einen Seite keine Gegenstände mit rollendem Verhalten in der Simulation trainiert wurden. Und auf der anderen Seite sind solche Gegenstände viel anfälliger dafür, sich in nicht modellierten Unvollkommenheiten, wie zum Beispiel Schraubenlöchern, zu verfangen.

Simulated task	Mean	Median	Individual trials (sorted)
Block (state)	$43.4 \pm 13.8$	50	-
Block (state, locked wrist)	$44.2 \pm 13.4$	50	-
Block (vision)	$30.0 \pm 10.3$	33	-
Octagonal prism (state)	$29.0 \pm 19.7$	30	-

Physical task	Mean	Median	Individual trials (sorted)
Block (state)	$18.8 \pm 17.1$	13	50, 41, 29, 27, 14, 12, 6, 4, 4, 1
Block (state, locked wrist)	$26.4 \pm 13.4$	28.5	50, 43, 32, 29, 28, 19, 13, 12, 9
Block (vision)	$15.2 \pm 14.3$	11.5	46, 28, 26, 15, 13, 10, 8, 3, 2, 1
Octagonal prism (state)	$7.8 \pm 7.8$	5	27, 15, 8, 8, 5, 5, 4, 3, 2, 1

Abbildung 16. Quantitative Ergebnisse. (Quelle: [2])

### G. Fazit zu Dactyl

Die Mitarbeiter von OpenAI zeigten in dem vorgestellten Paper, dass es möglich ist, einer Roboterhand zu einer großen Fingerfertigkeit zu verhelfen. Das dafür verwendete Modell wurde durch Deep Reinforcement Learning in einer Simulation trainiert und dann auf die Roboterhand transferiert. Möglich ist das nur, weil aufwändige Randomisierungen, groß angelegte Trainingsmöglichkeiten, Policies mit Speichern und eine Auswahl von entscheidenden Auswahlmöglichkeiten, die im Simulator modelliert wurden, verwendet wurden. Insgesamt wurde anschaulich gezeigt, dass es durch die Verwendung von Deep Reinforcement Learning möglich ist, komplexe Probleme der Robotik zu lösen, die mit herkömmlichen Ansätzen aktuell nicht lösbar sind.

## V. EMERGENT TOOL USE FROM MULTI-AGENT AUTOCURRICULA

Am 11.02.2020 veröffentlichten Bowen Baker et al. das Paper *Emergent tool use from multi-agent autocurricula* [3]. In diesem zeigten die Mitarbeiter von OpenAI, dass es möglich ist, durch bereits existierende Techniken des Deep

Reinforcement Learnings Agenten zu trainieren, das Spiel hide-and-seek zu lernen. Dabei entwickelten die Agenten - aufgeteilt in zwei Teams - selbstständig ein selbstüberwachtes Autokurrikulum sowie sechs klar differenzierbare Phasen von emergenten Strategien. Die Agenten wurden in einer speziellen Umgebung trainiert, lernten, Gegenstände dieser Umgebung zu ihrem Vorteil zu nutzen, und zeigten kooperatives Verhalten untereinander.

Autokurrikulum ist ein Konzept, bei dem es keine fest vorgeschriebenen Aufgaben und Lernziele gibt. Die Autoren ziehen hierzu Vergleiche zur natürlichen Auslese nach Dawkins und Krebs [23]. Wenn es eine neue, erfolgreiche Strategie oder Mutation gibt, übt das auf andere Organismen den Druck aus, sich an diese Evolution möglichst schnell anzupassen, um das Überleben zu sichern. Auch im Reinforcement Learning wird durch das Aufkommen neuer Strategien das ursprüngliche Ziel und die damit verbundene Aufgabenstellung verändert. Agenten müssen sich möglichst schnell an diese Veränderung anpassen. Somit stellen sich Agents, die miteinander konkurrieren, stets vor neue Herausforderungen. Die Entdeckung neuer Strategien wird nicht durch Reward Shaping (siehe II-A3) begünstigt. Es gibt keine besonderen Belohnungen für das Erforschen der Umgebung oder die Verwendung von Gegenständen aus der Umgebung.

#### A. Hide and Seek

1) *Spiel:* Bei *Hide and Seek* gibt es zwei Teams (Hider und Seeker), jeweils bestehend aus ein bis drei Agenten. Die Hider müssen vermeiden, von den Seekern gesehen zu werden. Aufgabe der Seeker ist es, die Hider zu finden. Außerdem gibt es in der Umgebung Gegenstände, die gegriffen (grab) oder gesperrt (lock) werden können. Das Szenario findet in zufällig generierten, nicht veränderbaren Räumen unterschiedlicher Komplexität statt. Bevor das Spiel beginnt, gibt es eine Vorbereitungsphase, in der die Seeker sich nicht bewegen können.

2) *Rewards:* Ein großes Ziel der Arbeit war es, eine möglichst offene Welt zu schaffen, um zu beobachten, welche Strategien die Agenten entwickeln. Deshalb wurde die Belohnungsfunktion sehr spärlich gehalten. So gibt es beispielsweise keine Belohnung für die Verwendung von Gegenständen. Die Rewards werden jeweils für alle Agents des Teams nach jeweils 240 Timesteps verteilt. Dies entspricht genau einer Episode, also der Dauer eines Spiels. Die in V-A1 angesprochene Vorbereitungsphase beträgt 40% einer Episode, also 96 Timesteps. Hiders bekommen eine Wertung von +1, wenn alle Hider versteckt sind, und -1, wenn mindestens ein Hider von einem Seeker gesehen wird. Für Seeker ist die Reward-Function genau umgekehrt. Das bedeutet, dass sie -1 erhalten, wenn alle Hider versteckt sind, und +1, wenn mindestens ein Hider von einem Seeker gesehen wird. Um zu verhindern, dass die Agents aus dem Spielfeld flüchten, gibt es noch einen Reward von -10, wenn man das 18m<sup>2</sup> große Spielfeld verlässt.

3) *Umgebung:* Die Experimente wurden in dem MuJoCo-Physikmodul simuliert. Bevölkert ist die Welt mit 1-3 Hidern und 1-3 Seekern. Außerdem gibt es 3-9 bewegliche Boxen, darunter normalerweise 1-3 längliche Boxen und zwei Rampen. Zufällig generierte, feste Mauern bilden Räume, die ebenfalls im Spiel genutzt werden können.

4) *Agents:* Die Agents beobachten Position, Geschwindigkeit und Größe anderer Agenten und der Umgebung. Sie können alles sehen, was sich in einem 135°-Fenster vor ihnen abspielt. Alle anderen Dinge werden für die Agenten ausmaskiert. Weiterhin verfügen sie über 30 LiDAR-Sensoren<sup>7</sup>, die rund um ihren Körper angebracht sind. Durch diese können sie ihre Umgebung zusätzlich wahrnehmen.

Bewegen können sich die Agenten in X-Achsen- und Y-Achsen-Richtung. Auch Drehungen sind möglich. Die Interaktion mit Gegenständen erfolgt über zwei binäre Aktionen: Greifen (grab) und Sperren (lock). Diese Aktionen sind nur verfügbar, wenn sich Gegenstände in der näheren Umgebung der Agenten befinden. Die Aktion Greifen führt dazu, dass ein Gegenstand an die Bewegung des Agenten gekoppelt wird. Gesperrte Objekte können nur von Agenten entsperrt werden, die sich im gleichen Team befinden. In Abbildung 17 sind die Bewegungen dargestellt, die die Agenten ausführen können.

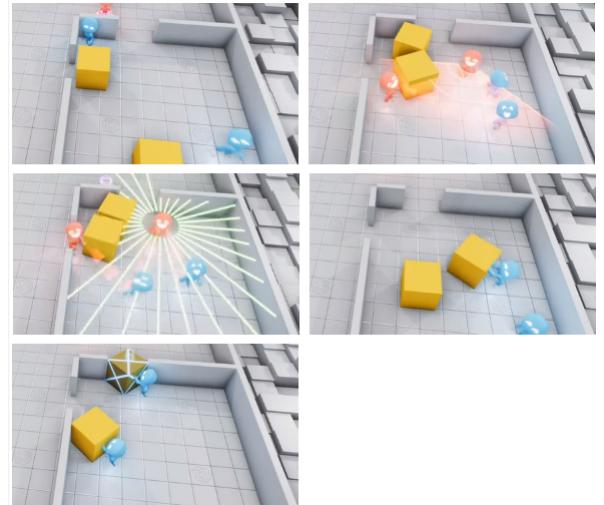


Abbildung 17. Mögliche Bewegungen und Interaktionen der Agenten mit ihrer Umgebung. (Quelle: [3])

#### B. Policy Optimization

Die Architektur der Policy ist in Abbildung 18 dargestellt. Analog zur Policy von OpenAI Five werden auch hier PPO und GAE verwendet. Dafür gibt es zwei Netzwerke, die durch LSTM realisiert sind. Das erste Netzwerk ist das Policy-Network und erzeugt die Wahrscheinlichkeitsverteilung für die nächsten Aktionen. Das zweite ist das Critic-Network und dient der Vorhersage der geschätzten Rewards. Auch bei diesem Paper wird das RL-Framework *Rapid* verwendet. Trainiert wird in einem zentralen System, wobei die Ausführung

<sup>7</sup>LiDAR bedeutet Light Detection and Ranging und ist eine Form des dreidimensionalen Laserscannings [24]

dezentralisiert stattfindet. Bei der Ausführung agiert jeder Agent auf Grundlage seiner eigenen Beobachtungen und seines eigenen Speicherzustandes. Bei der Optimierung, die zentralisiert stattfindet, wird eine Value-Function verwendet, der alle Informationen von Umgebung und Agenten zur Verfügung stehen. Für die Beobachtungen werden entity-centric observations [25] verwendet. Aufmerksamkeitsmechanismen [26] finden Verwendung beim Sammeln von Informationen, die die Objekte betreffen. Außerdem wird eine Policy-Architektur verwendet, die auf self-attention [27] basiert.

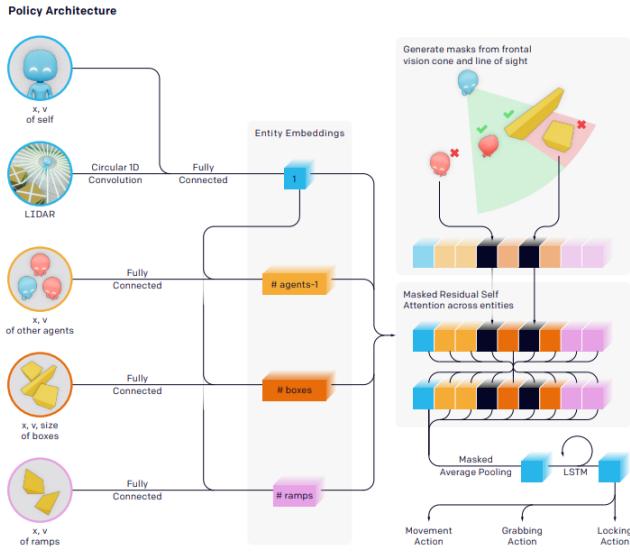


Abbildung 18. Netzwerkarchitektur der Policy. (Quelle: [3])

### C. Autocurricula Emergent Behavior

Während des Trainings konnten sechs unterscheidbare Strategien beobachtet werden (siehe Abbildung 19). Jede neue Strategie übt Druck auf das gegnerische Team aus und zwingt dieses dazu, ihre bisherigen Strategien weiterzuentwickeln. Weglaufen (Hider) und folgen beziehungsweise suchen (Seeker) waren die ersten Strategien, die schon in den ersten Episoden des Trainings angewandt wurden. Nach 25 Millionen Trainingsdurchläufen fingen die Hider an, Gegenstände ihrer Umgebung zu nutzen, um die Seeker auszusperren. Nach 100 Millionen Episoden lernten die Seeker, Rampen zu nutzen, und konnten so wieder Spiele gewinnen. Nach weiteren 10 Millionen Trainings wurde diese Strategie von den Hidern verhindert, indem sie die Rampen in der Vorbereitungsphase entweder gesperrt oder geklaut haben. Damit nahmen sie den Seekern den einzigen Gegenstand, der noch zu ihrem Vorteil verwendet werden konnte. Die Mitarbeiter von OpenAI vermuteten an dieser Stelle keine weiteren neuen Strategien. Doch nach insgesamt 380 Millionen Episoden fanden die Seeker heraus, dass sie über eine gesperrte Rampe auf eine Box gehen können, und nutzten die implementierte Physik aus, um sich selbst auf der Box zu verschieben. Dies wird als Box-Surfing bezeichnet. Die Verteidigungsstrategie der Hider war es, in der Vorbereitungsphase alle Boxen zu

sammeln, woraufhin die Angreifer keine weitere Strategie mehr hervorbrachten.

In allen Phasen des Spiels war es für die Agenten notwendig zu kooperieren, um zu gewinnen. Vor allem in komplexeren Umgebungen stellte sich heraus, dass die Agenten lernten, Aufgaben effizient untereinander aufzuteilen. Zum Beispiel konnten Teams mit zwei oder drei Hidern 25% beziehungsweise 36% schneller alle Boxen sperren, wenn es darum ging, eine Verteidigungsstrategie gegen Boxsurfing umzusetzen. Größere Teams können also Arbeit effektiv aufteilen und so Ziele erreichen, die für kleinere Teams unerreichbar sind.

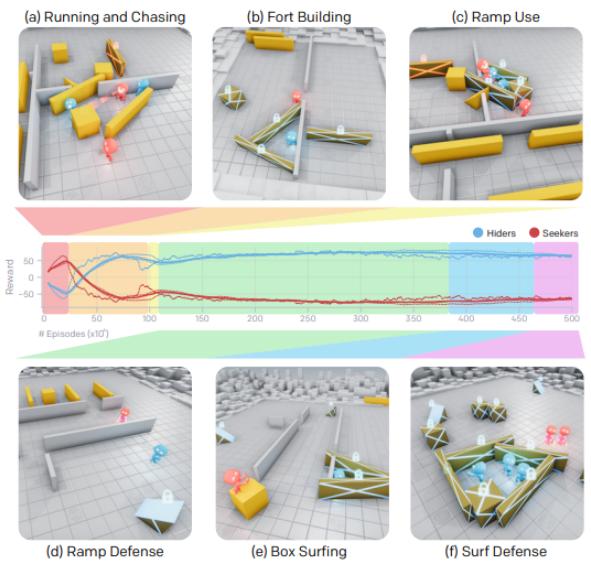


Abbildung 19. Sechs Phasen von emergenten Strategien und die entsprechenden Rewards. (Quelle: [3])

Bemerkenswert ist auch, dass die Agenten lernten, die simulierte Welt auf eine Art und Weise auszunutzen, die von den Entwicklern eigentlich nicht vorhergesehen war. Ein Beispiel ist das angesprochene Box-Surfing. Ein weiteres Beispiel ist, dass die Hider eine Möglichkeit fanden, die Rampen komplett aus der Umgebung zu werfen, indem sie mit einer Rampe gegen eine Mauer liefen. Der spektakulärste Fall ist, dass Seeker lernten, sich selbst in die Luft zu schießen, indem sie mit einer Rampe im richtigen Winkel gegen eine Mauer liefen. Reinforcement Learning ermöglicht es Agenten, ihre Umwelt und die darin vorhandenen physikalischen Zusammenhänge gründlich zu erforschen (explore) und selbst kleinste Möglichkeiten zur Verbesserung ihrer Situation zu erkennen und auszunutzen (exploit).

Wie schon in den Papieren zu OpenAI Five und Dactyl wurde auch in diesem Paper die Beobachtung gemacht, dass die Skallierung eine wichtige Rolle spielt. In Abbildung 20 ist dargestellt, wie die Größe der verwendeten Stapel mit der notwendigen Trainingszeit und der Anzahl der notwendigen Episoden zusammenhängt, um Phase vier (Ramp Defense) zu erreichen. Das Default-Modell wurde mit einer Batch Size von

64.000 und 1,6 Millionen Parametern trainiert. Reduziert man die Batch Size auf 32.000, ist ein wesentlich längeres Training notwendig, um die gleichen Strategien zu erhalten. Dies liegt daran, dass durch Erhöhung der Batch Size weniger Optimierungsschritte notwendig sind. Bei weiteren Experimenten, bei denen eine Batch Size von 16.000 und 8.000 verwendet wurde, konvergierte das Modell nicht. Das heißt, dass Phase vier (Ramp Defense) nie erreicht wurde.



Abbildung 20. Lernzeit, um Phase vier in Hide and Seek zu erreichen, in Abhängigkeit von der Batch Size. (Quelle: [3])

Wie in dem Paper zu Dactyl wurde auch in dieser Arbeit mit Randomisierung experimentiert. Verschieden komplexe Umgebungen oder Variationen in bestimmten Mechanismen wurden getestet. Beispielsweise wurde der Sperrmechanismus für Boxen und Rampen so verändert, dass er nicht für das ganze Team funktioniert, sondern nur ein Agent einen Gegenstand sperren und entsperren kann. Auch das grundlegende Spiel, also suchen und verstecken, wurde durch andere Spiele ausgetauscht oder durch Variationen ergänzt. Beispielsweise wurden Früchte in der Welt verteilt, deren Verzehr positiv durch die Belohnungsfunktion bewertet wurde. Das Ergebnis der Randomisierungen war, dass die Strategien der Agenten robuster wurden. Im Gegensatz dazu führte eine Vereinfachung der Umgebung dazu, dass die Agenten weniger ausgeklügelte Strategien lernten.

#### D. Evaluation

In diesem Abschnitt wird das Problem der Evaluation bezogen auf das gegebene Szenario, aber auch für zukünftige Arbeiten beleuchtet. Zentrales Thema ist, wie man den Lernerfolg der Agenten sinnvoll bewerten kann. Die Gesamtheit der Belohnung auszuwerten, ist nicht geeignet, weil nur schwer nachvollzogen werden kann, ob Agenten sich langsam verbessern oder in ihrer Entwicklung stagnieren. Auch ein Rating wie beispielsweise ELO [28] oder Trueskill [29], wie es bei OpenAI Five verwendet wird, ist nicht geeignet, da es zwar den Fortschritt gegenüber älteren Policies zeigt, aber keinen Rückschluss darauf gibt, ob die Verbesserung von der Optimierung bekannter Strategien oder dem Entdecken neuer Strategien kommt. Statistiken, die spezifisch für die Umgebung erhoben werden, können auch irreführend sein, weil zum Beispiel aus dem Verfolgen der Bewegung der Agenten nicht hervorgeht, in welche Richtung sich diese bewegen. Passende Metriken zu erstellen, ist zudem

eine sehr schwierige und kostspielige Aufgabe.

OpenAI schlägt einen spezifischen Intelligenztest für die Agenten vor, der aus fünf einzelnen Benchmark-Tests besteht und das Ziel hat, die gelernten Fähigkeiten der Agenten zu prüfen. Die fünf Tests lassen sich in *Cognition and Memory* und *Manipulation* einteilen und sind nachfolgend aufgeführt.

#### Cognition and Memory Tasks

- **Object Counting.** Ziel ist es, zu prüfen, ob die Agenten einen Sinn für Permanenz haben. Der Agent wird an einem festen Ort positioniert und beobachtet sechs Boxen, die sich zufällig nach links oder rechts bewegen und dann von einer Wand verdeckt werden. Wenn alle Boxen verschwunden sind, fragt man den Agenten, wie viele Boxen sich jeweils links und rechts befinden.
- **Lock and Return.** Ziel ist es, zu prüfen, ob ein Agent sich seine Anfangsposition merken kann, wenn er zwischenzeitlich eine Aufgabe ausführt. Der Agent befindet sich in einer Welt mit sechs Räumen und einer Box. Er muss die Box finden, sperren und dann anschließend wieder zu seiner Ausgangsposition zurückkehren.
- **Sequential Lock.** Es gibt vier Boxen, die zufällig auf drei Räume ohne Türen verteilt sind. In jedem Raum befindet sich eine Rampe. Ziel des Agenten ist es, alle Boxen in einer bestimmten Reihenfolge zu sperren. Dabei muss der Agent die Räume erkunden, die Reihenfolge herausfinden und Rampen benutzen, um in andere Räume zu gelangen.

#### Manipulation Tasks

- **Construction from Blueprint.** Auf dem Spielfeld befinden sich acht Würfel und ein bis vier Zielbereiche. Die Agenten müssen mit den Würfeln die Zielbereiche abdecken.
- **Shelter Construction.** Es gibt einen großen, gelben Zylinder, drei längliche Boxen und fünf Würfel. Der Agent muss den Zylinder einmauern.

In dem Test wurde die Performance von Agenten verglichen, die in Hide and Seek vortrainiert, from Scratch trainiert und durch Count-Based Intrinsic Motivation (II-A6) vortrainiert wurden. In Abbildung 21 sind die jeweiligen Tests und deren Auswertung zu sehen.

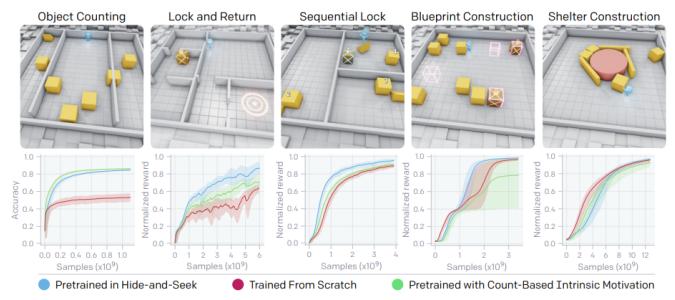


Abbildung 21. Quantitative Ergebnisse des Intelligenztests. (Quelle: [3])

Der Aktionsraum, der Observationsraum und die Art der Objekte in den Tests entsprechen denen, die in der Hide and Seek Umgebung verwendet wurden. Es soll festgestellt

werden, welche der Methoden (Trainieren in Hide and Seek, reines Training in der Testumgebung oder Training durch intrinsische Motivation) am besten für die Bewältigung des Intelligenztests geeignet ist, also welche Agenten schneller den finalen Grad ihrer Fähigkeiten erreichen. Die Ergebnisse zeigen, dass die in Hide and Seek vortrainierte Version in den drei Aufgabenbereichen Lock and Return, Sequential Lock und Construction from Blueprint besser lernt als ihre beiden Konkurrenten. Bei der Aufgabe, bei der Objekte gezählt werden müssen, performt sie etwas schlechter als die intrinsisch motivierte Version. Im Aufgabenbereich Shelter Construction kann zwar das gleiche Niveau an Fähigkeiten erreicht werden wie bei der Version from Scratch, jedoch dauert es länger. Vermutlich werden Fähigkeiten gelernt, die sehr an die jeweilige Aufgabenstellung gebunden und somit schwer übertragbar sind. Außerdem vermuten die Autoren, dass die teilweise besseren Ergebnisse von der Hide-and-Seek-Version damit zusammenhängen, dass die Repräsentationen der Features wiederverwendet werden können.

#### E. Fazit zu Hide and Seek

In dem Paper *Emergent tool use from multi-agent autocurricula* [3] zeigten Mitarbeiter von OpenAI, dass es möglich ist, Agenten in einer einfachen, offenen Umwelt zu trainieren, die dann ein Autokurriculum entwickeln und emergente Strategien hervorbringen. Dies gelang ihnen durch die Skalierung bereits existierender Algorithmen des Deep Reinforcement Learnings. Sie zeigten konkret, dass Agenten, eingesetzt in einer simulierten Umgebung, durch Self-Play in der Lage sind, Fähigkeiten zu lernen, die auch für den Menschen relevant sind. Außerdem geben sie einen Vorschlag, ein Evaluationssystem zu verwenden, um durch Transferlernen die Entwicklung der Agenten zu überprüfen.

#### VI. VERGLEICH UND ZUSAMMENFASSUNG DER VORGESTELLTEN PAPER

Bisher wurden drei verschiedene Paper vorgestellt, deren Gemeinsamkeit es ist, dass sie alle von OpenAI entwickelt wurden. Außerdem wurde der Optimierungsalgorithmus Proximal Policy Optimization verwendet. Im Paper zu Hide and Seek wurde der Algorithmus verwendet, der im Paper zu OpenAI Five vorgestellt wurde. Außerdem wurden für die Systeme OpenAI Five und Dactyl das gleiche System, nämlich Rapid verwendet, um zu dem entsprechenden KI-Systemen zu gelangen.

Im zweiten vorgestellten Paper gelang es Mitarbeitern von OpenAI, ein KI-System bestehend aus einer Deep Reinforcement Learning Komponente und einer Computer Vision Komponente zu trainieren, um einer Roboterhand beizubringen, einen Gegenstand durch geschickte Rotation möglichst schnell in eine vorgegebene Zielposition zu bringen. Auch hier wurde OpenAI Five als Basis für die KI verwendet.

Das dritte Paper konzentriert sich auf die Frage, welche neuartige Strategien RL-Agenten hervorbringen, wenn man sie in einem vergleichsweise einfachen Szenario trainiert. Hierzu spielten zwei Teams bestehend aus RL-Agenten eine

Vielzahl von Runden Verstecken. In ihrer Umgebung gab es Gegenstände, die sie zum Erreichen eines jeweiligen Ziels verwenden konnten. Bemerkenswert bei diesem Paper ist, dass die Umgebungsvariablen, wie zum Beispiel Belohnungsfunktion, Spielwelt und die grundsätzliche Komplexität des Szenarios, sehr überschaubar sind. Das Ergebnis dieses Papers, welches auch wieder auf der Verwendung von OpenAI Five basiert, ist, dass die Agenten im Verlauf von mehreren Millionen Trainingsspielen verschiedene Phasen durchliefen, in denen sie Strategien und entsprechende Konterstrategien hervorbrachten.

#### VII. REINFORCEMENT LEARNING BEIM MENSCHEN

In dem Paper *Investigating human priors for playing video games* [30], das am 25.07.2018 erschien, gingen Dubey et al. der Frage nach, inwiefern sich das Verhalten von Mensch und Maschine beim Lernen von neuen Verhaltensweisen ähnelt beziehungsweise unterscheidet. Die intuitive Antwort darauf ist, dass Menschen auf Erfahrungen zurückgreifen können, die sie in ihrem Leben gemacht haben, und sich dadurch schneller an neue Aufgaben anpassen können. Die Autoren untersuchten in ihrem Paper, welche Auswirkung auf die Lernfähigkeiten gewisse Vorerfahrungen haben. Dafür erstellten sie einfache Computerspiele und führten nach und nach Modifikationen durch. Ein Level eines solchen Computerspiels ist in Abbildung 22 zu sehen. Sehr schnell ist klar, dass man zum Lösen des Levels über die Gräben und Gegner springen, die Leitern nach oben klettern und schließlich zur Prinzessin gelangen muss.

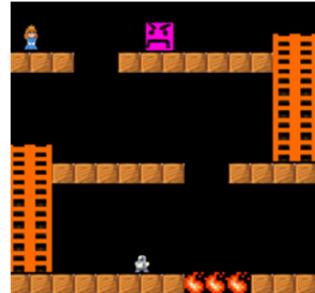


Abbildung 22. Originales Spiel. (Quelle: [30])

Im originalen Spiel brauchen Erwachsene durchschnittlich etwa 70 Sekunden, um ein Level zu lösen, ohne dass sie überhaupt irgendwelche Informationen darüber bekommen haben, wie das Spiel funktioniert, was das Ziel ist und was überhaupt zu tun ist. Dabei greifen sie auf bereits erlernte Konzepte zurück. Beispielsweise ist es schlecht, in einen Abgrund zu fallen oder in Feuer zu laufen. Der nächste Schritt ist die Modifikation der Spielwelt. Nach und nach werden gewisse Zusammenhänge aufgelöst oder verändert. So werden zum Beispiel Gegenstände wie Schlüssel oder Türen durch farbige Blöcke dargestellt oder das Konzept der Schwerkraft verändert. Gerade Letzteres ist tief in der menschlichen Wahrnehmung der Umgebung verankert. Diese Veränderungen haben erheblichen Einfluss auf die Zeit, die zum Lösen des Spiels benötigt wird. Ab einem gewissen Grad

der Veränderung ist das Spiel für einen Menschen sogar gar nicht mehr lösbar. In Abbildung 23 sind die Veränderungen, die an der Spielwelt vorgenommen wurden, dargestellt.

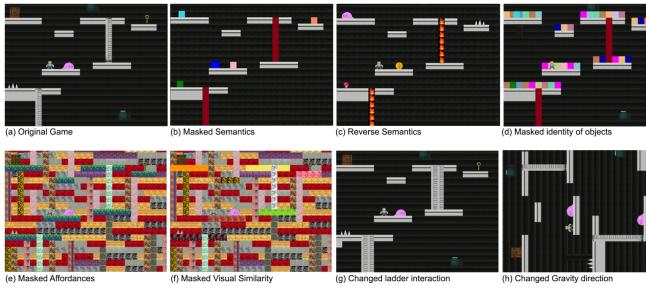


Abbildung 23. Veränderungen in der Gestaltung der Umgebung. (Quelle: [30])

Anschließend verglichen die Autoren die Ergebnisse der Menschen mit denen einer KI, die mit Reinforcement Learning [31] trainiert wurde. Es zeigte sich, dass die KI wesentlich länger brauchte (37 Stunden) als ein Mensch (70 Sekunden), um ein Level ohne Modifikationen erstmalig zu lösen. Das lässt sich dadurch begründen, dass eine KI keine Erfahrungen, wie zum Beispiel Kenntnis über die Schwerkraft, besitzt, von denen sie beim Spielen profitieren könnte. Jedoch zeigte sich auch, dass die KI wesentlich robuster gegenüber Veränderungen der Umgebung ist (siehe Abbildung 24). So ist es der KI möglich, das gelernte Verhalten sehr schnell auf neue Situationen anzupassen. Wird zum Beispiel das Konzept der Schwerkraft grundlegend geändert - wirkt von rechts nach links statt von oben nach unten -, ist das Spiel für einen Menschen kaum noch lösbar. Für eine KI hingegen stellt diese Veränderung kein großes Problem dar.

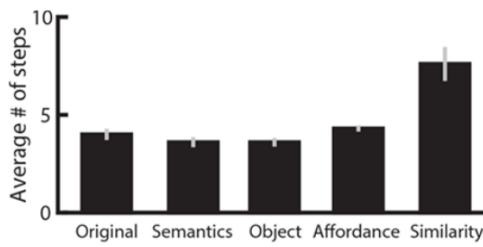


Abbildung 24. Performance des RL-Agents in Abhängigkeit von den Veränderungen des Levels. (Quelle: [30])

### VIII. FAZIT UND AUSBLICK

In dieser Arbeit wurden Paper, die sich mit dem Thema Deep Reinforcement Learning beschäftigen, detailliert vorgestellt. Dabei liegt der Fokus auf der Betrachtung eines Algorithmus (OpenAI Five), der ursprünglich für ein Computerspiel (Dota 2) entwickelt wurde, und darauf, wie dieser in Szenarien verwendet werden kann, die für den Menschen von großer Relevanz sind (Roboterhand, Hide and Seek).

Im Paper *Dota 2 with large scale deep reinforcement learning* stellten Mitarbeiter von OpenAI eine künstliche Intelligenz vor (OpenAI Five), der es gelang, ein professionelles E-Sport-Team in einem der meistgespielten Online-Strategie-Spiele (Dota 2) zu schlagen. Die KI, der das gelang, wurde über einen Zeitraum von 10 Monaten durch Spielen gegen sich selbst trainiert. Es wurden spezifische Tools (Surgery) entwickelt, um während der Trainings Veränderungen an dem gesamten System durchführen zu können, ohne das Training abbrechen zu müssen.

Im Paper *Learning dexterous in-hand manipulation* wurde gezeigt, dass es möglich ist, eine simulierte Roboterhand mit DRL so zu trainieren, dass sie einen Würfel in eine gewünschte Zielposition und Orientierung bringen kann. Anschließend wurde die gelernte Policy auf eine Roboterhand in der realen Welt transferiert, wo ebenfalls positive Ergebnisse erzielt werden konnten. Obwohl dem System keine menschlichen Bewegungen als Vorbild gezeigt wurden, stellte man fest, dass das Modell die Roboterhand in ähnlicher Weise bewegt, wie ein Mensch seine Hand verwendet.

Im Paper *Emergent tool use from multi-agent autocurricula* wurde untersucht, ob es möglich ist, Agenten mit DRL dazu zu bringen, selbstständig adaptive Strategien zu entwickeln. Dazu wurden zwei Teams gebildet, die gegeneinander das Spiel Hide and Seek spielten. Diese Teams bestanden aus Agenten, die mit DRL trainiert wurden. Es zeigte sich, dass effektive Strategien und Konterstrategien entwickelt wurden. Dabei lernten die Agenten, zu kooperieren und Gegenstände ihrer Umgebung gewinnbringend einzusetzen.

In allen Veröffentlichungen waren große Mengen von Trainingsdurchläufen nötig, um zu den Ergebnissen zu gelangen. Die verwendeten Ansätze ähneln sich zum Beispiel in der grundsätzlichen Anwendung von DRL und in weiteren verwendeten Elementen, wie zum Beispiel der Optimierungsmethode (PPO). Außerdem zeigte sich mehrmals, wie wichtig es ist, eine gute Skalierung für die Leistung zu finden. Auch Randomisierung ist ein sehr wichtiges Konzept, welches für den Erfolg von DRL maßgeblich ist.

Fluch und Segen beim DRL ist, dass die Modelle dazu tendieren, sich nicht immer so zu verhalten, wie es von den Entwicklern geplant ist. Agenten beim DRL sind - verglichen mit herkömmlicher Programmierung - sehr schwierig zu kontrollieren und überraschen oft mit unvorhergesehenem Verhalten.

Deep Reinforcement Learning ist zurzeit ein sehr großes Forschungsgebiet des maschinellen Lernens, in dem es aktuell viele Veröffentlichungen gibt. Das Verschmelzen von Robotik und Informatik könnte durch Reinforcement Learning in den nächsten Jahren zu weiteren Meilensteinen in den jeweiligen Bereichen führen. Durch leistungsfähigere Systeme, wie zum Beispiel Quantencomputer, könnte es in den nächsten Jahren

erneut einen großen Sprung auf dem Forschungsgebiet der künstlichen Intelligenz geben [32].

## LITERATUR

- [1] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [2] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [3] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent tool use from multi-agent autocurricula,” *arXiv preprint arXiv:1909.07528*, 2019.
- [4] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [6] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, “A survey on policy search for robotics,” *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.
- [7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [8] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.
- [9] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.
- [10] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.
- [11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [13] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, “Distributed prioritized experience replay,” *arXiv preprint arXiv:1803.00933*, 2018.
- [14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [15] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
- [16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.
- [17] H. Jiang, Y. Li, C. Zhou, H. Hong, T. Glade, and K. Yin, “Landslide displacement prediction combining lstm and svr algorithms: A case study of shengjibao landslide from the three gorges reservoir area,” *Applied Sciences*, vol. 10, no. 21, p. 7830, 2020.
- [18] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” *Advances in neural information processing systems*, vol. 29, 2016.
- [19] D. F. Tver, *Encyclopedic Dictionary of Industrial Technology: Materials, Processes and Equipment*. Springer Science & Business Media, 2012.
- [20] E. Uhlmann and J. Krüger, “Industrieroboter,” in *Dubbel*. Springer, 2018, pp. 1655–1663.
- [21] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.
- [22] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [23] R. Dawkins and J. R. Krebs, “Arms races between and within species,” *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 205, no. 1161, pp. 489–511, 1979.
- [24] U. Wandinger, *Introduction to lidar*. Springer, 2005.
- [25] S. Dzeroski, L. De Raedt, and K. Driessens, “Relational reinforcement learning,” *Machine learning*, vol. 43, no. 1, pp. 7–52, 2001.
- [26] Y. Duan, M. Andrychowicz, B. Stadie, O. Jonathan Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, “One-shot imitation learning,” *Advances in neural information processing systems*, vol. 30, 2017.
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [28] A. E. Elo, *The rating of chessplayers, past and present*. Arco Pub., 1978.
- [29] R. Herbrich, T. Minka, and T. Graepel, “Trueskill™: a bayesian skill rating system,” *Advances in neural information processing systems*, vol. 19, 2006.
- [30] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths, and A. A. Efros, “Investigating human priors for playing video games,” *arXiv preprint arXiv:1802.10217*, 2018.
- [31] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *International conference on machine learning*. PMLR, 2017, pp. 2778–2787.
- [32] K. Mainzer and K. Mainzer, “Was könnte quanten-künstliche intelligenz?” *Quantencomputer: Von der Quantenwelt zur Künstlichen Intelligenz*, pp. 137–165, 2020.