Reinforcement learning “Carrot or Stick”

1. Introduction

Das Konzept **“Reinforcement Learning (RL)”** ist ein bedeutender Teilbereich des Machine Learnings, das sich darauf konzentriert, Agenten durch Interaktionen mit ihrer Umgebung selbstständig lernen zu lassen. RL nutzt Prinzipien der Verhaltenspsychologie und Entscheidungsfindung, um Modelle zu entwickeln, die durch Belohnungen (Zuckerbrot) und Bestrafungen (Peitsche) lernen [1, S. 781-782].

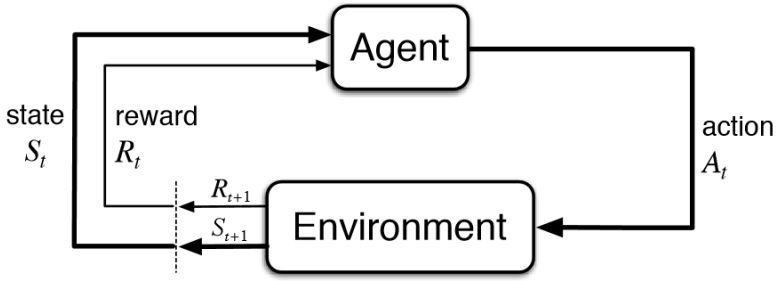


Fig.1 zeigt eine schematische Darstellung eines Reinforcement-Learning-Systems (RL-System), in dem ein "Agent" in einer Umgebung operiert und durch Interaktionen mit dieser Umgebung lernt.

OpenAI hat auch eine Form des Reinforcement Learning (RLHF) verwendet, um das ChatGPT-Modell zu trainieren [Fig.2]. Reinforcement Learning from Human Feedback ist eine Methode, die darauf abzielt, maschinelles Lernen durch die Integration menschlicher Rückmeldungen zu verbessern. Es kombiniert Prinzipien des Reinforcement Learning (RL) mit direktem Feedback von Menschen, um die Leistung und Nützlichkeit von KI-Systemen zu optimieren.

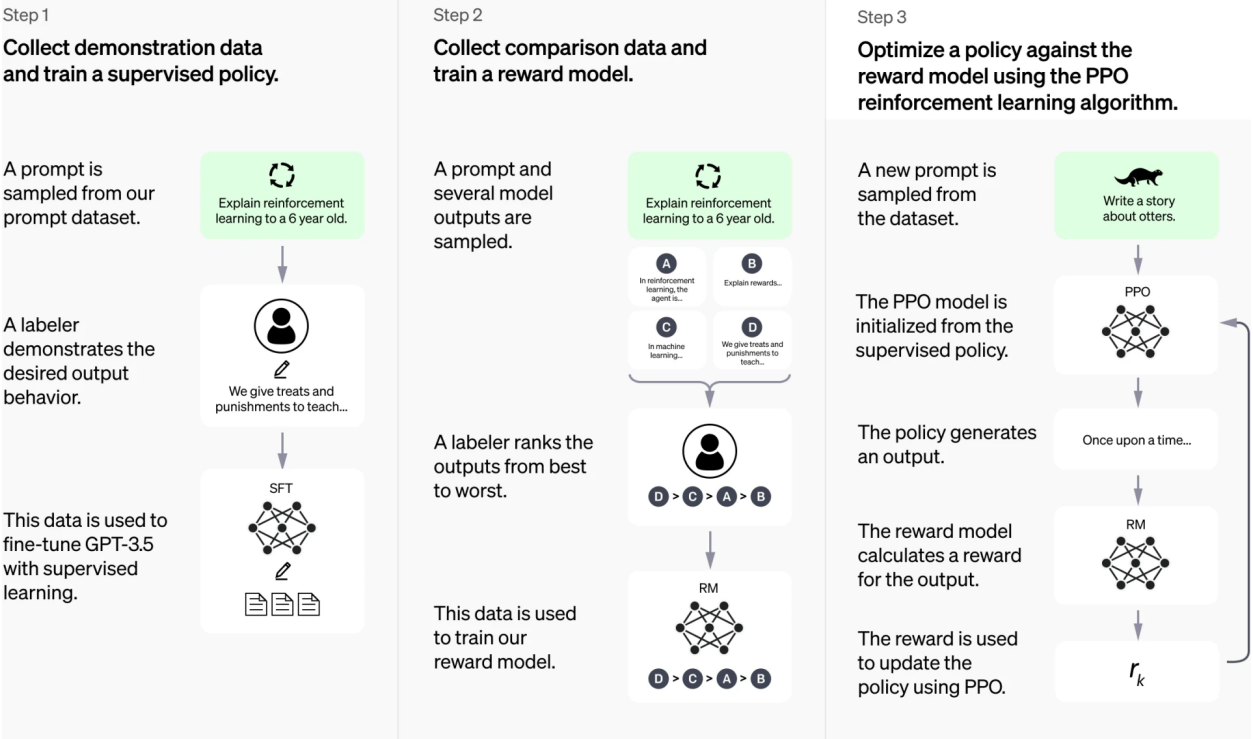


Fig.2 Das Bild darstellt einen dreistufigen Prozess zur Entwicklung eines verstärkungslernenden Modells (PPO) basierend auf GPT-3.5. [2]

2. Strategy Game (DeepBlue and Minimax Algorithm)

Bevor wir tiefer in das Reinforcement Learning eintauchen, betrachten wir den Brute-Force-Ansatz, wie KI durch umfassende Suche und hohe Rechenleistung in Spielen erfolgreich sein kann.

2.1 Suchverfahren für Strategiespiele

Suchverfahren sind Methoden, die in Computerspielen verwendet werden, um optimale Züge oder Strategien zu finden. Sie sind besonders in Strategiespielen wichtig, wo das Ziel darin besteht, eine bestmögliche Aktion zu bestimmen, die in einer bestimmten Spielsituation ausgeführt werden soll. Es gibt verschiedene Arten von Suchverfahren, die in der Künstlichen Intelligenz verwendet werden, darunter:

1. **Minimax-Algorithmus**: Ein grundlegender Algorithmus, der in Zwei-Spieler-Spielen verwendet wird. Er versucht, die besten Züge für den Spieler zu finden, der am Zug ist, indem er davon ausgeht, dass der Gegner immer die bestmöglichen Züge ausführt.
2. **Alpha-Beta-Suche**: Eine Erweiterung des Minimax-Algorithmus, die Teile des Suchbaums abschneidet, die nicht weiter untersucht werden müssen, um die Suche effizienter zu machen.
3. **Monte-Carlo-Tree-Search (MCTS)**: Ein probabilistisches Verfahren, das durch zufällige Probenahme von Zügen arbeitet und besonders gut bei Spielen mit großer Komplexität oder Unsicherheit geeignet ist.

2.2 DeepBlue und Minimax

DeepBlue ist nicht nur ein Schachcomputer, sondern eine revolutionäre Meisterleistung von IBM, die 1997 die Welt des Schachs für immer verändert hat, als er den damaligen Schachweltmeister Garry Kasparov in einem epischen

Match besiegt hat [3]. DeepBlue hat eine Kombination aus leistungsstarker Hardware und Algorithmen genutzt, um Schachzüge zu berechnen:

- **Hardware**: 32 Prozessoren, 200 Millionen Schachstellungen pro Sekunde bewertet, $L_{max\not\emptyset} = 11,38$ Milliarden Gleitkommaoperationen pro Sekunde ($11,38\ \mathrm{GFLOPS}$) Verarbeitungsgeschwindigkeit [4].
- **Algorithmen**: $\alpha\beta$-minimax-search, evaluation-algorithm [5].

Im Herzen dieses beeindruckenden Systems liegt der Minimax-Algorithmus, unterstützt durch eine intensive Alpha-Beta-Suche, die die Anzahl der zu bewertenden Züge drastisch reduziert. Der Minimax-Algorithmus ermöglicht es DeepBlue, die optimalen Züge zu finden. Diese Kombination aus Hardware und Algorithmen hat neu definiert, was Maschinen in komplexen Aufgaben, die bisher nur Menschen vorbehalten waren, leisten können.

2.3 Minimax-Algorithmus im Tic-Tac-Toe

Der Minimax-Algorithmus bewertet alle möglichen Züge im Spiel, um die beste Strategie zu finden. Er führt eine tiefgehende Suche durch den Spielbaum durch und berechnet für jeden möglichen Zug die minimale maximale Verlustwahrscheinlichkeit [6].

Tic-Tac-Toe Beispiel (**O** ist der aktuelle Spieler und **X** ist der Gegner):

1. **Spielzustände bewerten**: Der Algorithmus betrachtet alle möglichen Züge, die der aktuelle Spieler (entweder X oder O) ausführen kann, und bewertet den Zustand des Spielbretts nach jedem Zug.
2. **Rekursives Durchlaufen des Spielbaums**: Der Algorithmus wechselt rekursiv zwischen den beiden Spielern. Bei jedem Zug wird der Minimax-Algorithmus aufgerufen, um den Zug des Gegners zu bewerten. Ziel ist es, die beste Strategie für den aktuellen Spieler zu finden, indem er den besten Gegenzug des Gegners erwartet.
3. **Minimierung und Maximierung**: Der Algorithmus maximiert den Gewinn für den aktuellen Spieler und minimiert den Gewinn für den Gegner. In jeder Stufe des Suchbaums wählt der Algorithmus den Zug, der den maximalen Gewinn für den aktuellen Spieler und den minimalen Gewinn für den Gegner ergibt.
4. **Endzustände bewerten**: Wenn ein Endzustand erreicht ist (Sieg, Niederlage oder Unentschieden), wird dieser Zustand mit einem Wert bewertet (Gewinn kann z.B. mit +4, Niederlage mit -4, Unentschieden mit 0 bewertet werden).

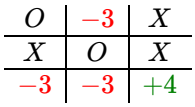
Beispiel-Ablauf:

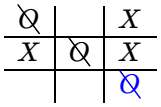
(1) X macht einen Zug auf Feld 3 → $\begin{array}{|c|c|c|}\hline & & \\ \hline X & & \\ \hline & & \\ \hline\end{array}$

(2) O überprüft alle möglichen Züge und erhält theoretisch die folgende Score-Tabelle → $\begin{array}{|c|c|c|}\hline +0 & -3 & -3 \\ \hline X & +0 & +0 \\ \hline +0 & -3 & -3 \\ \hline\end{array}$ und macht den Zug auf Feld 0, um den Gewinn zu maximieren → $\begin{array}{|c|c|c|}\hline O & & \\ \hline X & & \\ \hline & & \\ \hline\end{array}$

...

(5) X macht einen Zug auf Feld 5 → $\begin{array}{|c|c|c|}\hline O & & X \\ \hline X & O & X \\ \hline & & \\ \hline\end{array}$

(6) O überprüft alle möglichen Züge, stellt fest, dass ein Zug auf Feld 8 den Sieg bringt → 

, und gewinnt das Spiel → 

3. From Brute Force to Reinforcement Learning

Das Hauptkonzept des Minimax-Algorithmus basiert darauf, alle möglichen Ergebnisse (oder dank der $\alpha\beta$-Suche nur einen Teil der Ergebnisse) zu betrachten, um den optimalsten Zug aus Sicht des Scores vorherzusagen. Während dies im Schach durch Rechenleistung realisiert werden kann, wird die Aufgabe bei Spielen wie Go um viele Größenordnungen schwieriger.

> https://youtube.com/clip/UgkxiohLBplucPdQiYyizPLOy5VHfUzyCuoR?si=UqO497Zy9TAuyYWF

- Die Anzahl der möglichen Partien im Schach: 10^{120} Shannon-Zahl [7] (durchschnittlich 30 Züge für jede Variante).
- Die Anzahl der möglichen Positionen im Go: 10^{172} [8] (52 Größenordnungen mehr als mögliche Partien im Schach und 92 Größenordnungen mehr als Atome im Universum)

3.1 AlphaGo gegen Lee Sedol

Während der Erfolg von DeepBlue hauptsächlich auf reiner Rechenleistung und Suchalgorithmen basiert wurde, hat die Entwicklung von AlphaGo durch Google DeepMind einen bedeutenden Fortschritt durch die Integration von Deep Learning und Reinforcement Learning markiert.

Der Sieg von AlphaGo über den Weltmeister im Go, Lee Sedol, im Jahr 2016 [9] hat das Potenzial der Kombination von Deep Learning und Reinforcement Learning gezeigt.

AlphaGo verwendet eine Kombination aus Neuronale Netzwerke mit Q-Learning, zusammen mit Monte Carlo Tree Search (MCTS) [10]:

- **Neuronale Netzwerke**:
  - **Policy-Netzwerk (π):** Dieses Netzwerk sagt den nächsten Zug voraus, der basierend auf dem aktuellen Zustand des Bretts gespielt werden soll. Es gibt eine Wahrscheinlichkeitsverteilung über alle möglichen Züge aus.
  - **Value-Netzwerk (v):** Dieses Netzwerk bewertet die aktuelle Brettposition. Es gibt einen einzelnen Skalar Wert aus, der die Wahrscheinlichkeit angibt, dass der aktuelle Spieler von dieser Position aus gewinnt.
- **MCTS** ist ein heuristischer Suchalgorithmus für Entscheidungsprozesse, insbesondere in Spielen. Er wird verwendet, um optimale Züge zu finden, indem ein Suchbaum basierend auf zufälligen Simulationen des Spiels aufgebaut wird.
- **Training mit Deep-Reinforcement-Learning:**
  - Das Policy Network wird mit einer großen Anzahl von menschlichen Go-Partien trainiert.
  - Nach dem Supervised Learning wird das Policy Network durch Reinforcement Learning (gegen sich selbst) trainiert.
  - Das Value Network wird trainiert, indem es die Positionen und die endgültigen Ergebnisse der Selbstspiele verwendet.

$$a_t = arg\ max(Q(s_t, a) + u(s_t, a))$$

3.2 RL Verständnis durch Q-Learning

Q-Learning ist eine Methode des Reinforcement Learning (RL), bei der ein Agent lernt, eine bestimmte Aufgabe durch Interaktion mit einer Umgebung, die dem Agent entweder Zuckerbrot oder Peitsche gibt, auszuführen. Dabei verwendet der Agent eine Q-Funktion, um den erwarteten Nutzen (Reward) eines Zustands-Aktions-Paares zu bewerten [1, S. 791-796].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- s_t: Der Zustand zur Zeit t
- a_t: Die Aktion, die zur Zeit t im Zustand s_t ausgeführt wird.
- r_{t+1}: Die Belohnung, die der Agent erhält, wenn er die Aktion at ausführt und in den Zustand s_{t+1} wechselt.
- s_{t+1}: Der neue Zustand, in den der Agent wechselt, nachdem er a_t ausgeführt hat.

Algorithmus:

1. **Initialisierung**: Alle $Q(s, a)$ auf einen Startwert (z.B. 0) setzen.
2. **Wiederholung** (für jede Episode):
   - Zustand s wählen
   - Bis zum Terminalzustand wiederholen:
     - Aktion a wählen: Wähle eine Aktion a basierend auf der aktuellen Politik (z.B. ϵ-greedy).
     - Die Aktion a ausführen und die Belohnung r und den neuen Zustand s' beobachten.
     - $Q(s, a)$ aktualisieren.
     - Zustand s aktualisieren.

Beispiel-Ablauf (angenommen, der Q-Learning-Agent wurde bereits trainiert):

(1) X (Q-Agent) überprüft die Q-Tabelle und erhält folgendes → | | | |
|---|---|---|
| +2 | +1 | +2 |
| +1 | +2 | +1 |
| +2 | +1 | +2 |

und macht einen Zug auf Feld 3 → | | | |
|---|---|---|
| | | |
| | | X |

(Beim ersten Zug sind fast alle Züge gleich, sodass die Reihenfolge nicht so viel Gewicht hat)

(2) O macht einen Zug auf Feld 3 → | | | |
|---|---|---|
| O | | |
| | | X |

...

(6) O mach eine Zug auf Feld 1 → | | | |
|---|---|---|
| | O | |
| O | | X |
| O | X | X |

(7) X (Q-Agent) überprüft die Q-Tabelle und erhält folgendes → | | | |
|---|---|---|
| +15 | O | +8 |
| O | +9 | X |
| O | X | X |

und macht einen Zug auf Feld 0 → | | | |
|---|---|---|
| X | O | |
| O | | X |
| O | X | X |

(Dank des Trainings und der Erfahrungen aus früheren Spielen erkannte der Agent, dass es in

dieser Situation besser ist, links oben zu ziehen, um das Spiel nicht zu verlieren)

Conclusion

Die Entwicklung von DeepBlue zu AlphaGo zeigt, wie sich KI von einfacher Rechenleistung zu fortschrittlichen Lernalgorithmen entwickelt hat. DeepBlue's Erfolg war ein Triumph der Technik und Suchalgorithmen. AlphaGo's Errungenschaften heben die transformative Kraft von Reinforcement Learning und Deep Learning hervor. Diese Fortschritte fördern weiterhin die Innovation in der KI, sodass Maschinen lernen, sich anpassen und immer komplexere Aufgaben meistern können.

References

1. Aston Zhang, Zachary C. Lipton, Mu Li, And Alexander J. Smola. Dive into Deep Learning
2. Introducing ChatGPT
3. Chess Grandmaster Garry Kasparov Replays His Four Most Memorable Games | The New Yorker
4. Deep Blue, IBM's computer checkmated a human chess champion in a computing tour de force
5. Deep Blue Algorithm
6. Minimax Algorithm Guide: How to Create an Unbeatable AI
7. Parallel Universes and Shannon's number.
8. Go and mathematics
9. AlphaGo - The Movie | Full award-winning documentary
10. AlphaGo: How it works technically?