Lernen durch Belohnung: Wie Robi laufen lernt (Reinforcement Learning)

Kapitel online

Kein Lehrer, keine richtige Antwort — nur Punkte. Robi sucht sich seinen Weg durchs Büro selbst: mit ε-greedy an drei Kaffeeautomaten, mit einer Q-Tabelle der Form [36, 4] und einer einzigen Update-Regel. Zum Anfassen: Büro umbauen, Regler drehen, Lernkurve fallen sehen — und erleben, wie eine falsch gebaute Belohnung den Agenten an jedem Ziel vorbeiführt. Von Atari über AlphaGo bis zu dem RLHF, das ChatGPT höflich macht.

Fortgeschritten · Kapitel 28 von 50 · ca. 25 Min. · 200 XP

Einfach gesagt: Robi bekommt nie die Lösung, nur Punkte — mal ein Plus, mal ein Minus — und du siehst, wie er sich seinen Weg selbst erarbeitet.

Bisher lag neben jedem Beispiel die richtige Antwort: hier das Foto, dort das Etikett „Katze"; hier die Wohnfläche, dort der Preis. Das Netz rät, vergleicht mit der Lösung, korrigiert sich — **überwachtes Lernen** (Kapitel 115).

Jetzt nehmen wir die Lösung weg. **Robi**, der kleine Roboter, der hier immer neben mir steht, soll allein durchs Büro zu seiner Ladestation finden. Niemand sagt ihm, wo sie ist oder welcher Schritt richtig war. Es gibt nur **Punkte**: mal ein Plus, mal ein Minus, meistens nichts. Daraus muss er sein Verhalten selbst bauen. Das ist **Reinforcement Learning** (bestärkendes Lernen) — die Verfahrensfamilie hinter Atari-Rekorden, AlphaGo und dem höflichen Ton von ChatGPT.

HerrKI-Tipp: „Eine Belohnung ist ein Signal, keine Lösung. Robi findet genau den Weg, der die meisten Punkte bringt — nicht den, den du gemeint hast."

Was du hier lernst

- Warum ein **Belohnungssignal** etwas anderes ist als eine Lösung
- Wie **Neugier und Gewohnheit** (Exploration, Exploitation) sich streiten — und was ε steuert
- Das Vokabular: **Zustand, Aktion, Belohnung, Politik, Wert** und der Diskontfaktor γ
- Wie die **Q-Tabelle** als Tensor [36, 4] aussieht und wie eine Zahl darin aktualisiert wird
- Warum eine schlecht gebaute Belohnung den Agenten **austricksen lässt** — und was das mit Chatbots zu tun hat

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/280_reinforcement_learning ausprobieren.

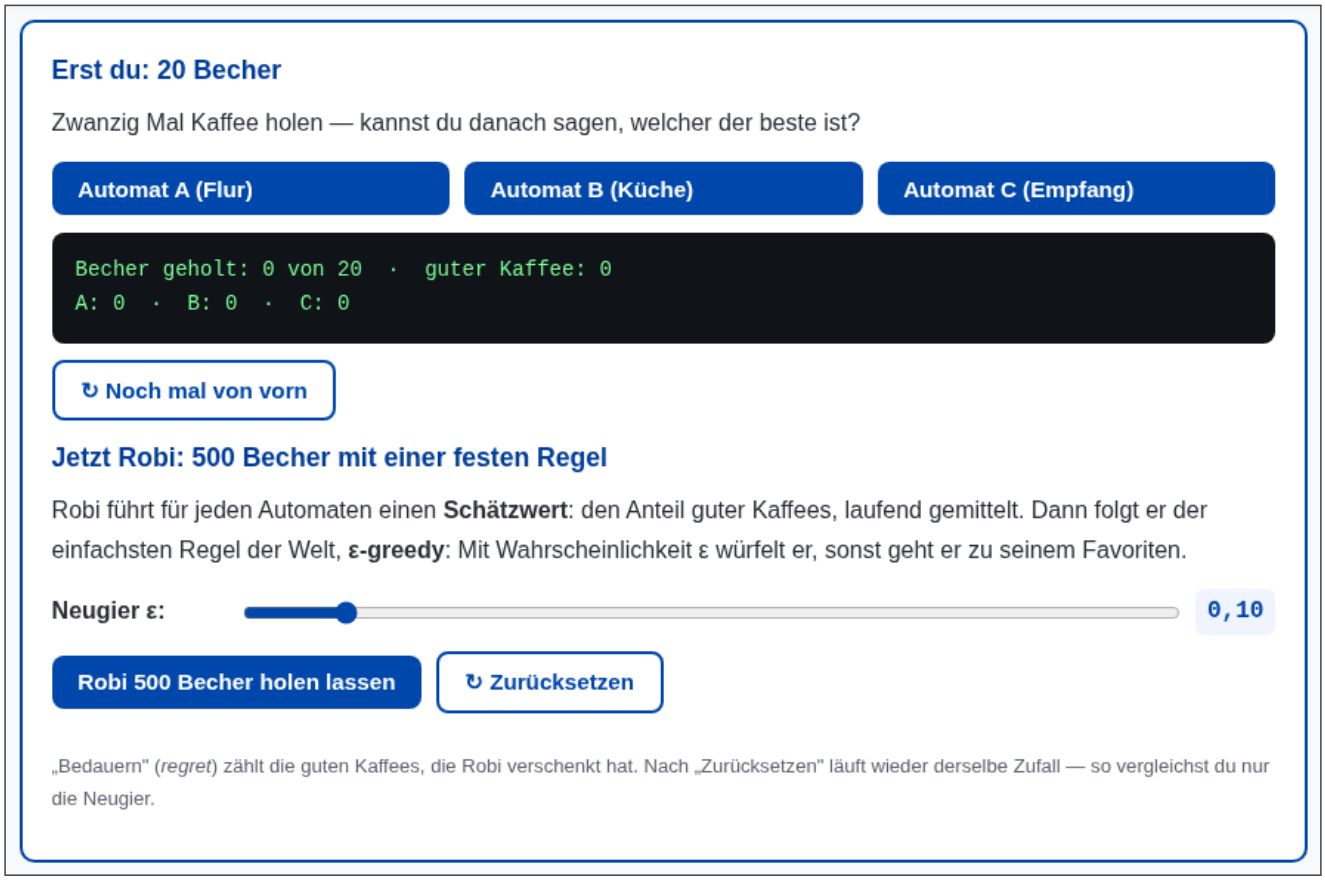
Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Drei Kaffeeautomaten: Neugier gegen Gewohnheit

Vorher das kleinste denkbare Lernproblem — eines aus jedem Büro. Im Flur, in der Küche und am Empfang steht je ein Kaffeeautomat, jeder mit seiner **eigenen, verborgenen Quote** guter Becher. Du kennst nur den Becher in deiner Hand. Damit steckst du in der Zwickmühle, die das ganze Kapitel trägt: **Ausnutzen** (Exploitation) heißt, zum bisher besten Automaten zu gehen — bringt sofort Kaffee, aber vielleicht war dein Eindruck Zufall. **Erkunden** (Exploration) heißt, einen anderen zu probieren — kostet vielleicht einen schlechten Becher, liefert aber Wissen, das nur so zu haben ist.

Am Gerät ausprobieren: herrki.de/280_reinforcement_learning#abschnitt-2

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Genau das zeigt der erste Lauf nach jedem Zurücksetzen, gleicher Zufall, nur anderes ε: **ε = 0 → 131** gute Kaffees, ε = 0,05 → 183, **ε = 0,10 → 330**, ε = 0,2 → 321, ε = 0,5 → 280, **ε = 1 → 214**. Beide Randfälle lohnen einen zweiten Blick:

- **ε = 0 — nur Gewohnheit.** Der erste Automat mit zwei guten Bechern bleibt für immer Favorit; die anderen tastet Robi nie wieder an. Hier steht er 500 Mal am schlechtesten — keine Dummheit, nur die Folge davon, dass er seine Schätzung nie überprüft.
- **ε = 1 — nur Neugier.** Robi würfelt jeden Becher aus. Seine Schätzwerte werden *sehr genau*; genutzt hat er sie nie. Wissen ohne Anwendung bringt keinen Kaffee.

Dazwischen liegt das Gute: ein kleines ε, das gerade genug zweifelt. Das Spielzeugproblem heißt **Multi-Armed Bandit** und füllt bei Sutton & Barto das ganze zweite Kapitel — Reinforcement Learning ohne Ortswechsel. Den bauen wir jetzt ein.

Zustand, Aktion, Belohnung, Politik

Beim Kaffee gab es nur eine Entscheidung, immer dieselbe. Im Büro entscheidet *wo* Robi steht mit, was gut ist. Fünf Begriffe beschreiben das vollständig — und mit genau diesen fünf arbeitet die Fachliteratur.

Begriff	In Robis Büro	Beim Hund	In der Fahrschule	Beim Chatbot
Zustand (state, s)	eines der 36 Felder	Sitzt? Herrchen da?	Tempo, Gang, Ampelfarbe	der bisherige Gesprächsverlauf
Aktion (action, a)	hoch, rechts, runter, links	bellen, sitzen, weglaufen	kuppeln, bremsen, blinken	das nächste Wort
Belohnung (reward, r)	−1 je Schritt, −10 Pfütze, +10 Station	Leckerli oder nichts	„gut so" oder das Bremsen des Fahrlehrers	Daumen hoch eines Menschen
Politik (policy, π)	„auf Feld 14 nach rechts"	das antrainierte Verhalten	dein Fahrstil	der Antwortstil des Modells
Wert (value, V/Q)	„wie gut ist es, hier zu stehen?"	Vorfreude aufs Leckerli	„die Spur ist entspannter"	erwartete Bewertung der Antwort

Dazu zwei Begriffe, die man leicht verwechselt:

- **Episode** (Versuch): ein Durchlauf vom Start bis zum Ende — Ladestation erreicht, in die Pfütze getappt oder nach 200 Schritten abgebrochen. Danach steht Robi wieder am Start, behält aber alles Gelernte.
- **Diskontfaktor γ** (gamma, zwischen 0 und 1): Robis *Ungeduld*. Eine Belohnung in k Schritten zählt nur noch γ^k-fach. Bei $\gamma = 0{,}9$ sind +10 in zwölf Schritten noch $10 \cdot 0{,}9^{12} \approx$ **2,8** wert, bei $\gamma = 0{,}5$ nur $10 \cdot 0{,}5^{12} \approx$ **0,002** — die Ladestation strahlt dann nicht mehr bis zum Start.

Der **Wert** beantwortet „Wie gut ist es, hier zu stehen, wenn ich danach vernünftig weiterlaufe?" — und ist nicht die Belohnung: Das Feld neben der Ladestation gibt null Punkte und ist trotzdem hervorragend. Diese Unterscheidung ist der Kern des Verfahrens.

Runter zum Tensor: Die Q-Tabelle hat die Form [36, 4]

Robis ganzes Wissen passt in eine Tabelle. Sein Büro ist ein 6-×-6-Raster, also **36 Zustände**, in jedem **4 Aktionen**. Für jedes Paar merkt er sich eine Zahl **Q(s, a)**: „wie gut ist Aktion *a* auf Feld *s*?". Eine Matrix, ein Tensor zweiter Stufe (Kapitel 010):

Q-Tabelle
[36, 4]
36 Felder × 4 Richtungen = 144 Zahlen

→

eine Zeile = ein Feld
[4]
hoch, rechts, runter, links

Die **Politik** ist damit nichts Eigenes mehr, sie fällt aus der Tabelle: „Nimm in jeder Zeile den größten Wert." Lernen heißt, diese 144 Zahlen zu verbessern, bis das stimmt.

Die Regel, die alles macht

Nach jedem Schritt ändert Robi **genau eine** davon. Die Vorschrift stammt von Christopher Watkins (1989) und heißt **Q-Learning**:

```
// s = altes Feld, a = gewählte Richtung, r = kassierte Punkte, s' = neues Feld
Q(s,a) ← Q(s,a) + α · [ r + γ · maxa' Q(s',a') − Q(s,a) ]
```

Die eckige Klammer ist der **TD-Fehler** (temporal difference, zeitliche Differenz) — Robis *Überraschung*. Sie vergleicht, was er vorher dachte (Q(s,a)), mit dem, was er jetzt weiß: r + γ · max Q(s′), also die eben kassierten Punkte plus das Beste, was vom neuen Feld aus noch geht, mit γ abgezinst — das **TD-Ziel**. α (die Lernrate) sagt, welchen Anteil der Überraschung Robi übernimmt. Einmal von Hand, mit α = 0,5 und γ = 0,9: Robi steht auf Feld 12, geht nach rechts, kassiert **r = −1** und landet auf einem Feld mit bestem Eintrag **max Q(s′) = 4,0**. In der Zelle stand **2,0**:

```
TD-Ziel = r + γ · max Q(s′) = −1 + 0,9 · 4,0 = 2,6
TD-Fehler = TD-Ziel − Q(s,a) = 2,6 − 2,0 = 0,6
```

```
Q(s,a) neu = 2,0 + 0,5 · 0,6 = 2,3
```

Derselbe Gedanke wie beim Gradientenabstieg: Fehler ausrechnen, einen Bruchteil davon in die richtige Richtung gehen, wiederholen. Nur kommt der Fehler dort aus einer *bekannten* Lösung und hier aus Robis *eigener späterer Schätzung* — er zieht sich am eigenen Schopf aus dem Sumpf (**Bootstrapping**). Dass das beweisbar zum richtigen Ergebnis führt, wenn jede Aktion oft genug drankommt, zeigten Watkins und Dayan 1992.

Am Gerät ausprobieren: herrki.de/280_reinforcement_learning#abschnitt-4

Direkt zum Modul

Mini-Modul: einen Update-Schritt rechnen

Ein Ausschnitt aus den 144 Zahlen — fünf Felder, vier Richtungen, dunkleres Blau heißt höherer Wert. Klick eine Zelle an und lass die Regel darauf los; darunter steht die Rechnung mit eingesetzten Zahlen. Vorgewählt ist das Beispiel von oben.

Zustand	↑ hoch	→ rechts	↓ runter	← links
Feld 12	1,00	**2,00**	0,50	-0,50
Feld 13	0,00	3,00	1,50	1,00
Feld 14	2,00	4,00	2,50	0,00
Feld 15	-1,00	1,00	3,00	2,00
Feld 16	0,50	0,00	1,00	3,50

Lernrate α: 0,5

Diskont γ: 0,90

Einen Update-Schritt rechnen | ↻ Zurücksetzen

```
Feld 12, Aktion → rechts   (α = 0,5, γ = 0,90)
alter Wert   Q(s,a) = 2,00
Belohnung    r      = -1   (ein Schritt kostet)
bestes Q im Folgefeld (Feld 14):  max Q(s') = 4,00
TD-Ziel   = r + γ · max Q(s') = -1 + 0,90 · 4,00 = 2,60
TD-Fehler = TD-Ziel − Q(s,a) = 2,60 − 2,00 = 0,60
neuer Wert = Q(s,a) + α · TD-Fehler = 2,00 + 0,5 · 0,60 = 2,30
```

Interaktives Modul – am Gerät ausprobieren

Und wenn die Tabelle nicht mehr passt?

36 Felder sind gemütlich. Ein Atari-Bildschirm hat 210 × 160 Bildpunkte — eine Tabelle mit einer Zeile pro möglichem Bildschirm wird es nie geben. Der Ausweg von DeepMind 2013: **Ersetze die Tabelle durch ein neuronales Netz.** Vorne die Bildpunkte hinein, hinten die Q-Werte für jeden Joystick-Befehl heraus — bei Robi eine Ausgabe der Form [4]. Aus

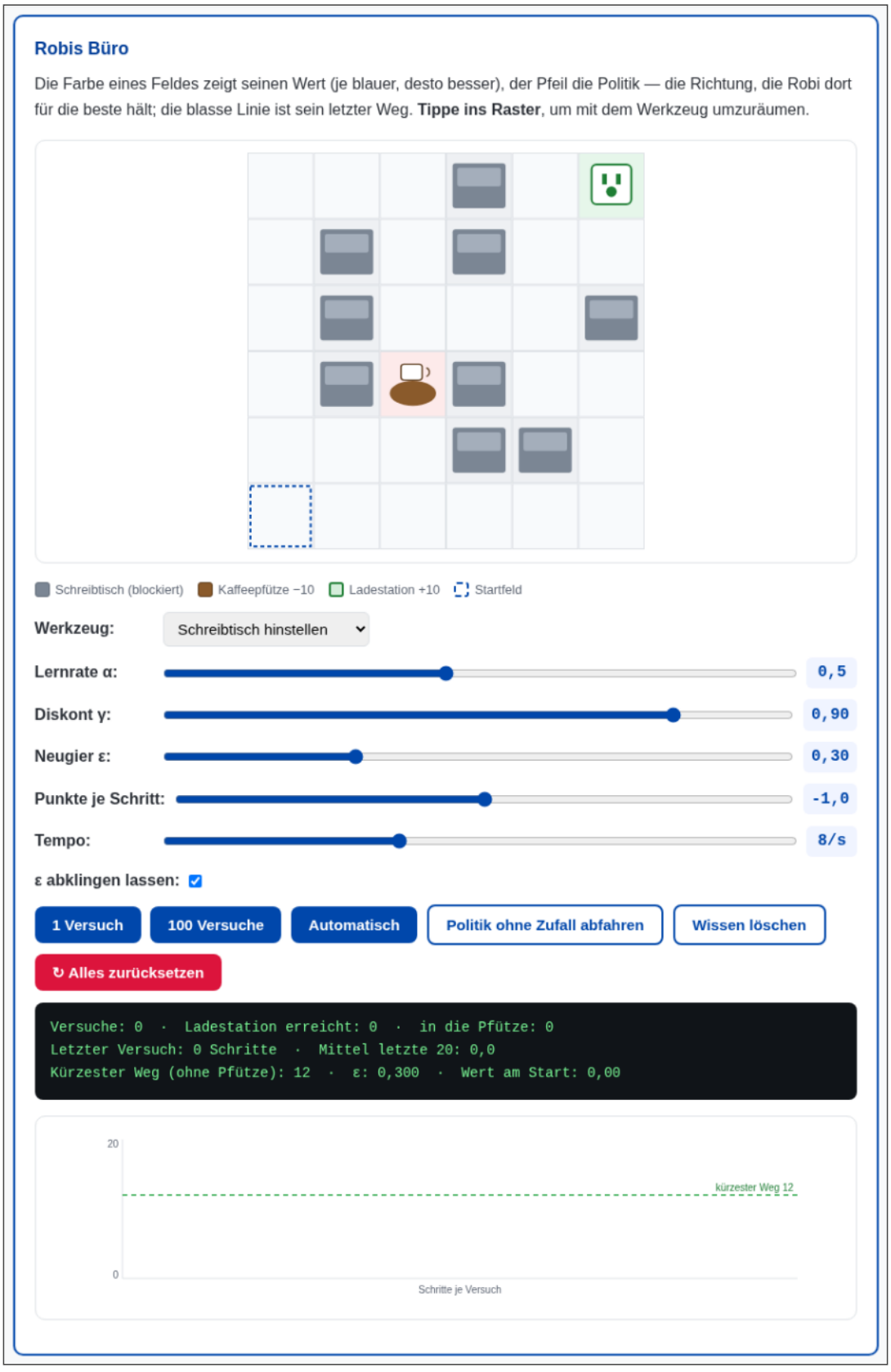
Nachschlagen wird Schätzen: das **Deep Q-Network** (DQN). Die Update-Regel bleibt gleich.

Zum Anfassen: Robi lernt sein Büro

Jetzt in echt. Im Raster versperren **Schreibtische** den Weg, in der Mitte steht eine **Kaffeepfütze** (−10, Versuch beendet), oben rechts die **Ladestation** (+10, Versuch beendet). Jeder Schritt kostet −1, deshalb hat Robi es eilig. Start ist unten links, an meinem Schreibtisch.

Am Gerät ausprobieren: herrki.de/280_reinforcement_learning#abschnitt-5

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Drück zweimal „100 Versuche": Die Kurve fällt von weit über fünfzig Schritten auf die grüne Linie, den kürzesten Weg ohne Pfütze. Gezeigt hat ihn Robi niemand.

Drei Dinge zum Ausprobieren:

- **ε auf 1 stellen, 100 Versuche.** Die Kurve fällt nicht mehr — Robi würfelt ja nur. Jetzt „Politik ohne Zufall abfahren" drücken: Er läuft den *optimalen* Weg. Er *hat* gelernt, er hat es nur nie benutzt. Das ist die Besonderheit von Q-Learning: Es lernt über die *beste* Aktion, nicht über die ausgeführte (**off-policy**).
- **γ auf 0,5, „Wissen löschen", 200 Versuche.** „Wert am Start" liegt jetzt nahe −2 statt bei rund −3,7, die Farbverläufe werden flach: Vom Ziel kommt am Start fast nichts mehr an. Dass Robi trotzdem hinfindet, verdankt er allein den −1 pro Schritt. In einer exakten Tabelle reicht so ein winziger Unterschied — sobald ein Netz die Werte schätzt, geht er im Rauschen unter.
- **Einen Schreibtisch auf die gelernte Route stellen.** Robis Wissen bleibt beim Umräumen stehen. Die Kurve springt hoch, weil er gegen den neuen Tisch rennt, und fällt dann wieder, weil er umlernt — überwachtes Lernen kann das nicht, denn Robi erzeugt seine Trainingsdaten selbst.

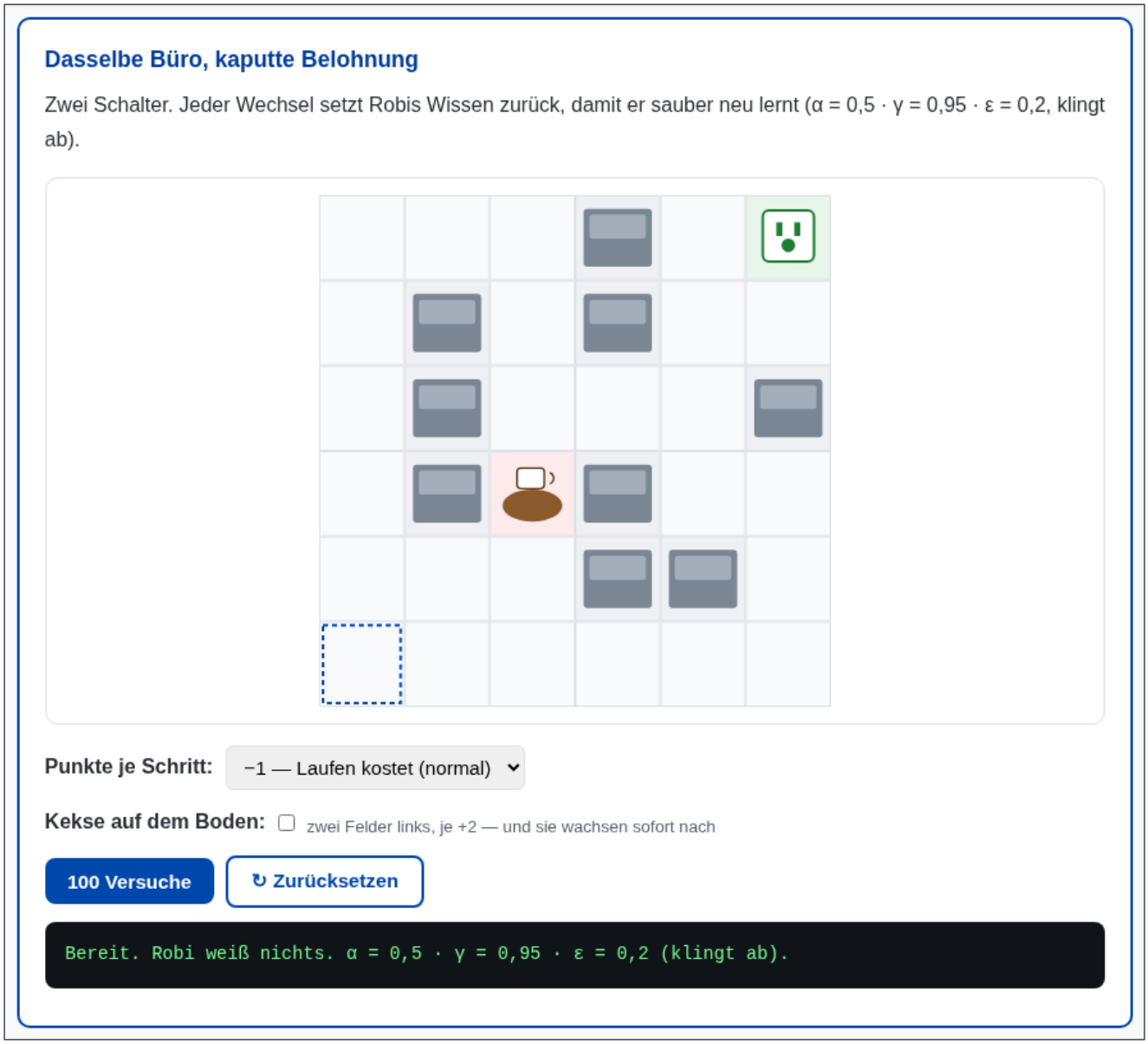
Warum klappt ε = 0 hier oft trotzdem, anders als beim Kaffee? Alle 144 Zahlen starten bei 0 und jeder Schritt kostet −1 — damit sieht *jedes nie besuchte Feld* besser aus als jedes bekannte. Dieser „optimistische Start" erzwingt Neugier ohne Würfel, verlässlich ist er aber nicht.

Belohnungs-Design: Wenn Robi das System austrickst

Bis hier war die Belohnung vernünftig gebaut. Jetzt drehen wir daran — so, wie es in Projekten aus Versehen passiert.

Am Gerät ausprobieren: herrki.de/280_reinforcement_learning#abschnitt-6

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Beide Fälle sehen harmlos aus und enden verheerend:

- **+1 je Schritt** („Bewegung fördern wir doch"). Rechne mit: Ewig herumlaufen bringt bei $\gamma = 0{,}95$ einen Wert von $1/(1-0{,}95) = \mathbf{20}$, die Ladestation nur **10** — und beendet den Versuch. Robi tut das Richtige: Er meidet sie und dreht bis zum Abbruch Runden.

- **Kekse auf dem Boden.** Zwei Felder nebeneinander, je +2, und sie wachsen nach. Robi pendelt zwischen ihnen: −1 für den Schritt, +2 für den Keks, netto +1 — *jeden Schritt*. Auch das schlägt die einmaligen +10; die Ladestation sieht er nie wieder.

Keine Marotte, sondern die ehrlichste Eigenschaft des Verfahrens: Ein Agent maximiert, was du *hinschreibst*, nicht, was du *meinst*. Die Lücke heißt **Spezifikations-Lücke**, das Ausnutzen **Reward Hacking** — bei Amodei und Kollegen 2016 eines von fünf konkreten Sicherheitsproblemen.

⚠ **Der Klassiker: das Bootsrennen.** OpenAI trainierte 2016 einen Agenten auf das Rennspiel *CoastRunners*. Punkte gab es nicht fürs Ankommen, sondern für Ziele entlang der Strecke. Der Agent fand eine Lagune, in der er im Kreis fuhr und dieselben drei Ziele abräumte, sobald sie nachwuchsen — brennend, gegen andere Boote krachend, in der falschen Richtung. Ergebnis: im Schnitt **20 % mehr Punkte als menschliche Spieler**, das Rennen nie beendet. Robis Keks-Trick mit größerem Budget.

Bei Sprachmodellen ist die Belohnung noch weicher: Dort sagt kein Spielstand, was gut war, sondern ein **Belohnungsmodell** aus menschlichen Vergleichen. Es belohnt, was Menschen *gut finden* — nicht das, was *wahr* ist. Eine flüssige, selbstsichere, falsche Antwort holt leicht mehr Punkte als ein „Das weiß ich nicht". Damit sind wir in Kapitel 350.

Zurück zur Anwendung: Von Atari zu ChatGPT

Robis Büro ist ein Spielzeug — das Verfahren dahinter nicht.

- **Atari, 2013 und 2015.** DeepMinds erstes DQN lernte sieben Atari-Spiele allein aus Bildpunkten und Punktestand: besser als alle bisherigen Verfahren bei sechs, besser als ein menschlicher Experte bei dreien. Zwei Jahre später standen in *Nature* **49 Spiele** — mit *demselben* Algorithmus, derselben Architektur und denselben Hyperparametern, auf dem Niveau eines professionellen Spieletesters. Neu war vor allem der **Replay-Speicher**: Erlebte Schritte werden gesammelt und später in zufälliger Reihenfolge nachtrainiert, damit ähnliche Situationen das Netz nicht verziehen.
- **Go, 2016 und 2017.** AlphaGo kombinierte Netze mit Baumsuche und schlug den Europameister 5:0 — der erste Sieg gegen einen Profi auf dem vollen Brett. AlphaGo Zero lernte ein Jahr später *ohne jede menschliche Partie* und gewann 100:0 gegen seinen Vorgänger.
- **Robotik und Empfehlungen.** Greifen und Laufen werden millionenfach simuliert, bevor echte Hardware anfasst; Empfehlungssysteme optimieren auf Verweildauer — mit Robis Keks-Lücke.
- **Und die Sprachmodelle.** Ouyang und Kollegen zeigten 2022, wie aus GPT-3 InstructGPT wurde: Menschen sortieren Antworten nach Güte, daraus lernt ein Belohnungsmodell, und mit ihm als Punktegeber wird das Sprachmodell per RL nachtrainiert (PPO). Das ist **RLHF** — Reinforcement Learning from Human Feedback. Der Effekt: Antworten des 1,3-Milliarden-Modells wurden denen des 175-Milliarden-GPT-3 vorgezogen, bei **100-mal weniger Parametern**.

Damit ist klar, warum ein Chatmodell höflich ist und warum es ablehnt: **gelerntes Belohnungsverhalten**, kein eingebauter Filter — der Faden, an dem Kapitel 392 zieht. Wann man ein Modell überhaupt nachtrainiert statt es besser zu fragen, klärt Kapitel 370.

Realitätscheck. RL ist kein Allheilmittel: Es braucht sehr viele Versuche — Robi ein paar hundert für 36 Felder, DQN pro Spiel Millionen Einzelbilder — und deshalb fast immer eine **Simulation**, in der Fehler nichts kosten. Es ist instabil und steht und fällt mit der Belohnungsfunktion. In der Praxis lösen überwachte Verfahren weiterhin die meisten Aufgaben; RL kommt dort zum Zug, wo es keine fertigen Antworten gibt, sondern nur ein Urteil über das Ergebnis.

Was heißt das für dich?

Drei Dinge, ganz ohne selbst einen Agenten zu bauen:

- **Achte darauf, was gemessen wird.** Jedes System, das auf eine Kennzahl optimiert — Klickrate, Verweildauer, gelöste Tickets — ist ein Agent mit Belohnungsfunktion. Frag nicht „ist das Ziel gut gemeint?", sondern „was ist der billigste Weg, diese Zahl hochzutreiben?".
- **Höflichkeit ist antrainiert.** Wenn ein Chatbot freundlich zustimmt, ist das keine Meinung, sondern ein hoch bewertetes Muster. Widerspruch wurde seltener belohnt.
- **Neugier gehört ins Verfahren.** $\varepsilon = 0$ heißt: nie prüfen, ob die Gewohnheit noch stimmt. Gilt für Automaten, Agenten und Arbeitsabläufe gleichermaßen.

Zum Weiterlesen

- Standardwerk, frei als PDF: R. S. Sutton & A. G. Barto, „Reinforcement Learning: An Introduction" (incompleteideas.net/book/the-book-2nd.html), 2. Aufl., MIT Press 2018 — Bandit Kap. 2, Q-Learning Kap. 6
- Konvergenzbeweis: C. J. C. H. Watkins & P. Dayan, „Q-learning" (gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf), Machine Learning 8 (1992), S. 279–292 (DOI (doi.org/10.1007/BF00992698)); Grundlage: Watkins' Dissertation, Cambridge 1989
- Atari aus Bildpunkten: V. Mnih u. a., „Playing Atari with Deep Reinforcement Learning" (arxiv.org/abs/1312.5602) (2013) und „Human-level control…" (nature.com/articles/nature14236), Nature 518 (2015)
- Selbstspiel: D. Silver u. a., AlphaGo (nature.com/articles/nature16961), Nature 529 (2016) und AlphaGo Zero (nature.com/articles/nature24270), Nature 550 (2017)
- RLHF: L. Ouyang u. a., „Training language models to follow instructions with human feedback" (arxiv.org/abs/2203.02155) (2022)
- Reward Hacking als Sicherheitsproblem: D. Amodei u. a., „Concrete Problems in AI Safety" (arxiv.org/abs/1606.06565) (2016)
- Das Bootsrennen: OpenAI, „Faulty reward functions in the wild" (web.archive.org/web/20240127091815/https://openai.com/research/faulty-reward-functions), 21. 12. 2016 (Archivfassung, das Original blockt Automaten)
- Auf HerrKI: 115 · 118 · 220 · 350 · 370 · 380 · 392

Kurzer Check – mit Lösung

Lösungen für Lehrkräfte – für Lernende die Online-Version nutzen.

Frage 1: Wofür ist ε bei ε-greedy zuständig?

- Für die Schrittweite: wie stark ein Update den Q-Wert verändert.
- ✔ **Für die Neugier: wie oft der Agent etwas Zufälliges probiert, statt seinen Favoriten zu nehmen.**
- Für die Weitsicht: wie stark späte Belohnungen abgewertet werden.

Richtig. ε regelt Exploration gegen Exploitation. Die Schrittweite ist α, die Weitsicht γ — drei Regler, die man leicht verwechselt. ε = 0: nie prüfen. ε = 1: nie nutzen.

Frage 2: Du stellst γ von 0,9 auf 0,1. Was passiert?

- Er lernt langsamer, findet aber denselben Weg mit denselben Werten.
- Er erkundet mehr, weil unbekannte Felder attraktiver werden.
- ✔ **Er wird kurzsichtig: Eine Belohnung in zwölf Schritten zählt nur noch $0{,}1^{12}$ — vom Ziel kommt am Start nichts mehr an.**

Genau. γ zinst ab: In k Schritten zählt eine Belohnung noch γ hoch k. Bei γ = 0,9 sind +10 über zwölf Schritte rund 2,8 wert, bei γ = 0,5 noch 0,002, bei γ = 0,1 nichts mehr. Der Wertverlauf wird flach, der Agent hat keine Richtung mehr.

Frage 3: Warum führt „+1 Punkt für jeden Schritt" den Agenten in die Irre, obwohl Bewegung doch erwünscht ist?

- Weil positive Belohnungen die Q-Werte über 1 treiben und die Rechnung überläuft.
- ✔ **Weil ewiges Weiterlaufen mehr Punkte sammelt als das Ziel, das den Versuch beendet — der Agent maximiert korrekt, nur die falsche Größe.**
- Weil der Agent das Ziel nicht mehr findet, wenn er unterwegs Punkte bekommt.

Richtig — der Kern des Reward Hacking. Bei γ = 0,95 ist ewiges Herumlaufen $1 / (1 - 0{,}95) = 20$ wert, die Ladestation nur 10. Der Agent tut genau das, was die Belohnung sagt; das Problem sitzt in der Spezifikation, nicht im Agenten. Dieselbe Lücke fuhr OpenAIs Boot 2016 im Kreis.

Das nimmst du mit

- **Kein Lehrer, nur Punkte.** RL bekommt keine Lösung, sondern eine Bewertung im Nachhinein — und muss selbst herausfinden, welcher Schritt sie verdient hat.
- **Drei Regler, drei Aufgaben.** α = wie stark korrigieren, γ = wie weit vorausschauen, ε = wie oft etwas wagen. Wer sie verwechselt, sucht Fehler an der falschen Stelle.
- **Das Wissen ist eine Tabelle.** Q hat bei Robi die Form [36, 4]; eine Regel ändert je Schritt genau eine Zahl darin. Wird die Tabelle zu groß, schätzt ein Netz sie — DQN.
- **Der Agent maximiert das Geschriebene.** Nachwachsende Kekse oder Punkte fürs Laufen führen zuverlässig am Ziel vorbei. Belohnungen zu entwerfen ist die eigentliche Arbeit.
- **ChatGPT ist ein Nachfahre davon.** Belohnungsmodell aus menschlichen Vergleichen, dann RL — deshalb sind Höflichkeit und auch das „Nein" gelerntes Verhalten, keine Regel im Programmcode.

Robi hat gelernt, ohne dass ihm jemand den Weg gesagt hat — er hat sich sein Wissen selbst zusammengelaufen. Im nächsten Pfad drehen wir das Fernglas um: „Latent Space — die geheime Landkarte" zeigt, was in einem Netz eigentlich entsteht, wenn es lernt.

So zitierst du dieses Kapitel

HerrKI: Lernen durch Belohnung: Wie Robi laufen lernt (Reinforcement Learning). herrki.de/280_reinforcement_learning, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.