Monte-Carlo & Temporal-Difference Learning

Grundlagen & Wiederholung

Ein Agent lebt in seiner Welt, macht Aktionen, bekommt Belohnungen – und will lernen, wie er sich verhalten soll, um langfristig möglichst viel Belohnung zu sammeln.

Die drei zentralen Fragen:

1. *Wie gut ist es, in Zustand S zu sein?* → **Value Function V(s)**
2. *Wie gut ist es, in Zustand S die Aktion A zu wählen?* → **Action-Value Function Q(s,a)**
3. *Welche Aktion soll ich wählen?* → **Policy π**

Bellman-Gleichung (intuitiv):

„Der Wert eines Zustands = Sofortige Belohnung + Wert des nächsten Zustands (abgezinst)"

Missverständnisse ⚠

Missverständnis	Richtigstellung
„Belohnung = Wert"	Nein! Belohnung ist **sofortig**, Wert ist **langfristig**
„γ=1 ist am besten"	Kann zu Instabilität führen, kein Anreiz für schnelle Lösung
„Policy = deterministisch"	Kann stochastisch sein: π(a|s) ∈ [0,1]

Frage: Warum brauchen wir einen Discount-Faktor γ < 1?
a) Damit der Agent schneller lernt
b) Damit zukünftige Belohnungen weniger zählen als sofortige
c) Damit die Policy deterministisch wird

Monte-Carlo Methoden

Erklärung (intuitiv):

Kernidee: Lernen durch vollständige Erfahrung.
🎯 „Spiele das Spiel bis zum Ende, dann schau was insgesamt rausgekommen ist."

Monte-Carlo Prinzip:

1. Führe eine **komplette Episode** durch ($S_0,A_0,R_1,S_1,\ldots,S_t$)
2. Berechne den **tatsächlichen Return** rückwärts
3. Update den Schätzwert V(s) in Richtung des tatsächlichen Returns

Return berechnen:

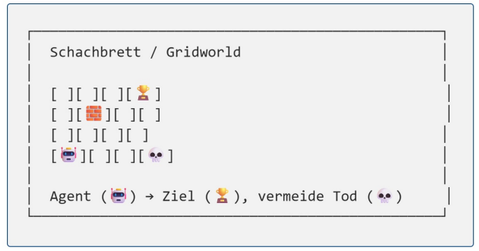
$$G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots + \gamma^{T-t} \cdot R_t$$

Update-Regel:

```
V(s) ← V(s) + α · [G - V(s)]
       ↑          ↑
    Lernrate    "Überraschung"
```

Frage: Warum hat MC eine hohe Varianz?
Frage: Kann man MC für ein Schachspiel einsetzen?
Frage: Was ist α (alpha) in der Update-Regel?
a) Der Discount-Faktor
b) Die Lernrate (Schrittgröße)
c) Die Explorations-Rate

Beispiel:




```
Episode 1:  →→→↑→
            Belohnungen: 0, 0, 0, 0, +10
            G (rückwärts berechnet):

Zeitschritt:  t=0   t=1   t=2   t=3   t=4
Reward:         0     0     0     0   +10
Return G:    7.29  8.1   9.0  10.0  10.0
             ←──────────────────────(γ=0.9)

NACH der Episode: Update aller besuchten Zustände!
```

First-Visit vs. Every-Visit MC

```
Episode: S₁ → S₂ → S₁ → S₃ → Ende
                  ↑
           S₁ wird zweimal besucht!

First-Visit MC:   Nur 1. Besuch von S₁ zählt
Every-Visit MC:   Beide Besuche von S₁ zählen
```

✅ Vorteile	❌ Nachteile
Kein Bias – lernt aus echten Returns	Braucht **komplette Episoden**
Einfach zu verstehen	**Hohe Varianz**
Kein Modell der Umgebung nötig	Kein Online-Learning möglich
Unabhängig von Initialwerten	Nur für episodische Tasks

Missverständnisse ⚠

Missverständnis	Richtigstellung
„MC ist langsam, also schlecht"	Für viele echte Probleme (z.B. Brettspiele) sehr effektiv
„First-Visit ist besser als Every-Visit"	Beide konvergieren – Kontext entscheidet
„MC lernt aus jedem Schritt"	**Nein!** Nur nach kompletter Episode

Temporal-Difference Learning(0)

Kernidee: Lernen **während** der Episode, nach **jedem Schritt**.

Vergleich MC vs. TD:
MC: Student bekommt sein Lernergebnis erst am Ende des Semesters nach den Klausuren
TD: Schüler bekommt nach jeder Hausaufgabe ein Feedback

TD(0) Update:

```
V(s) ← V(s) + α · [R + γ·V(s') - V(s)]
                     └──────────────┘
                       "TD-Error" (δ)
```

TD-Error δ:

wie viel besser/schlechter war die Realität vs. unsere Schätzung?

```
δ = R + γ·V(s') - V(s)
    └──────┘   └──┘
    Was ich bekam  Was ich erwartet hatte
```

Visualisierung

```
MC:
t=0  t=1  t=2  t=3  t=4(Ende)
 S₀ → S₁ → S₂ → S₃ → 🏁
 ←──────────────────────── Update NACH Episode
TD:
t=0  t=1  t=2  t=3  t=4(Ende)
 S₀ → S₁ → S₂ → S₃ → 🏁
 ↑    ↑    ↑    ↑    ↑
Update nach JEDEM Schritt!
```

Bootstrapping:

```
MC:  V(s) → G              (echter Return, keine Schätzung)
TD:  V(s) → R + γ·V(s')   (Schätzung nutzt Schätzung = Bootstrapping)
```

MC vs. TD Vergleich

```
Beispiel: Agent geht 5 Schritte, dann Belohnung +10, gamma=0.9, alpha=0.1

MC (nach Episode):
V(S₀) ← V(S₀) + α[7.29 - V(S₀)]  # echter Return
V(S₁) ← V(S₁) + α[8.1  - V(S₁)]
V(S₂) ← V(S₂) + α[9.0  - V(S₂)]

TD (nach jedem Schritt):
Schritt 1: V(S₀) ← V(S₀) + α[0 + γ·V(S₁) - V(S₀)]
Schritt 2: V(S₁) ← V(S₁) + α[0 + γ·V(S₂) - V(S₁)]
...
Schritt 5: V(S₄) ← V(S₄) + α[10 + γ·0    - V(S₄)]
          ↑ Erst hier "weiß" TD vom Reward!
```

Missverständnisse ⚠

Missverständnis	Richtigstellung
„TD ist immer besser als MC"	TD hat mehr **Bias** durch Bootstrapping, MC mehr **Varianz**
„TD-Error ist ein Fehler"	Eher ein **Lern-Signal** – auch positiv möglich
„TD braucht keine Episodes"	TD kann mit kontinuierlichen Tasks umgehen, muss aber nicht

Frage: Was ist Bootstrapping?
Frage: Ein Agent in S_0, macht Aktion, landet in S_1, bekommt R=2. $V(S_0)$=3, $V(S_1)$=5, γ=0.9, α=0.1. Wie lautet der neue Wert $V(S_0)$?

On-Policy SARSA

SARSA = State-Action-Reward-State-

```
(Sₜ, Aₜ, Rₜ₊₁, Sₜ₊₁, Aₜ₊₁)
  S    A    R      S'     A'
```

Es werden 5 Elemente für ein Update benötigt!

Kernidee: Jetzt lernen wir nicht nur V(s), sondern **Q(s,a)** – den Wert einer Aktion in einem Zustand.

🔑 **On-Policy:** Das Tupel, aus dem wir lernen, verwendet dieselbe Policy, die auch handelt.

Update-Regel:

```
Q(s,a) ← Q(s,a) + α · [R + γ·Q(s',a') - Q(s,a)]
                                ↑
                  Nächste Aktion NACH aktueller Policy!
```

Visualisierung

```
SARSA - Lernschleife:

    ┌──────────────────────────────────┐
    │                                  │
    ▼                                  │
Zustand S ──π(s)──► Aktion A           │
    │                  │               │
    │                  │               │
    │              Umgebung            │
    │                  │               │
    ▼                  ▼               │
Zustand S' ◄──── Reward R              │
    │                                  │
    └──π(s')──► Aktion A' ─────────────►┘
                   │
                   └──► UPDATE Q(S,A) mit (S,A,R,S',A')
```

SARSA ist vorsichtig

```
Klippen-Problem (Cliff Walking):

┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
│  │  │  │  │  │  │  │  │  │  │  │  ← sicherer Weg
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
│👾│☠️│☠️│☠️│☠️│☠️│☠️│☠️│☠️│☠️│🏁│  ← Klippe (R=-100)
└──┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘

SARSA lernt:  OBEN LANG gehen (sicherer Weg)
              → Weil ε-greedy manchmal in die Klippe fällt
              → SARSA "weiß" das und ist vorsichtig!
```

Missverständnisse ⚠

Missverständnis	Richtigstellung
„On-Policy = besser"	On/Off-Policy sind **verschiedene Werkzeuge**, nicht besser/schlechter
„SARSA lernt die optimale Policy"	SARSA lernt die **optimale Policy bzgl. ε-greedy** (nicht die greedy-optimale)
„A' muss die beste Aktion sein"	Nein! A' ist die Aktion **wie sie die Policy wählen würde**

Frage: Warum heißt SARSA "on-policy"?
Frage: Was passiert mit SARSA wenn ε→0?

On-Policy Expected SARSA

Problem mit SARSA:

In Q(s',a') hängt das Update von einer zufällig gewählten nächsten Aktion ab → hohe Varianz.

Expected SARSA Ansatz:

Statt einer zufälligen Aktion nehmen wir den **Erwartungswert über alle möglichen nächsten Aktionen**!
Statt zu würfeln, was der Nachbar morgen tut – frage ihn, was er **durchschnittlich** tut!

```
SARSA:
  Q(s,a) ← Q(s,a) + α[R + γ · Q(s',a')              - Q(s,a)]
                                  ↑
                            eine zufällige Aktion

Expected SARSA:
  Q(s,a) ← Q(s,a) + α[R + γ · Σ π(a'|s')·Q(s',a')  - Q(s,a)]
                                  ↑
                            Erwartungswert aller Aktionen
```

Visualisierung

```
SARSA Update:                    Expected SARSA Update:

    S' ──a'₁──► Q(s',a'₁)            S' ──a'₁──► Q(s',a'₁) × π(a'₁|s')
          └─── (zufällig!)                 a'₂──► Q(s',a'₂) × π(a'₂|s')
                                           a'₃──► Q(s',a'₃) × π(a'₃|s')
                                                  ──────────────────────
                                                          Summe ↑
```

SARSA vs. expected SARSA

Aspekt	SARSA	Expected SARSA
Target	Q(s',a') (eine Aktion)	Σ π(a'|s')·Q(s',a') (Erwartung)
Varianz	Höher	**Niedriger**
Bias	Gleich	Gleich
Rechenaufwand	Geringer	Höher (Σ über alle Aktionen)
Stabilität	Weniger stabil	**Stabiler**

Missverständnisse ⚠

Missverständnis	Richtigstellung
„Expected SARSA = greedy"	Nein! Die **Policy bleibt ε-greedy**, nur der Update nutzt den Erwartungswert
„Immer besser als SARSA"	In kleinen Aktionsräumen kaum Unterschied; bei großen: klar besser
„Gleich wie Q-Learning"	Nur wenn ε=0 (greedy)! Sonst andere Gewichtung

Frage: Expected SARSA mit ε=0 (greedy Policy) – was ergibt sich?
Frage: Warum hat Expected SARSA geringere Varianz als SARSA?

Off-Policy Q-Learning

Kernidee: Der Agent verhält sich nach einer Policy (behavior policy), lernt aber für eine andere Policy (target policy).

🔑 **Off-Policy:** „Ich beobachte einen anderen Fahrer, aber lerne für meine optimale Fahrweise."

Update-Regel:

```
Q(s,a) ← Q(s,a) + α · [R + γ · max_a' Q(s',a') - Q(s,a)]
                              └──────────────┘
                                Bestes Q im nächsten Zustand
                                (egal was tatsächlich gemacht wird!)
```

Der Unterschied zu SARSA:

```
SARSA:      R + γ · Q(s', a')        ← a' von ε-greedy Policy
Q-Learning: R + γ · max Q(s', a')    ← IMMER die beste Aktion!
```

Visualisierung

```
Q-Learning lernt IMMER die optimale greedy Policy:

Behavior Policy (ε-greedy):      Target Policy (greedy):

  S ──ε──► zufällige Aktion        S ──► max Q(s,a)
  S ──1-ε► beste Aktion

       ↓                                ↓
  tatsächliche                    was beim Update
    Aktion                        angenommen wird

→ Q-Learning lernt Q* direkt – unabhängig von Exploration!
```

Cliff Walking: Q-Learning vs. SARSA

```
┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
│  │  │  │  │  │  │  │  │  │  │  │  ← SARSA (vorsichtig, oben)
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
│👾│☠️│☠️│☠️│☠️│☠️│☠️│☠️│☠️│☠️│🏁│  ← Q-Learning (optimal, aber riskant!)
└──┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘

Q-Learning: Lernt den OPTIMALEN (kurzen) Weg unten entlang
SARSA:      Lernt den SICHEREN Weg (wegen eigener ε-Fehler)
```

Missverständnisse ⚠

Missverständnis	Richtigstellung
„Off-Policy ist immer besser"	Off-Policy kann mit **hoher Varianz** und Instabilität kämpfen
„Q-Learning findet immer die optimale Policy"	Nur mit ausreichender **Exploration** und Konvergenzbedingungen
„max Q ist greedy = kein Lernen mehr"	Die **Exploration** (ε) sorgt weiterhin für neue Erfahrungen
„SARSA und Q-Learning konvergieren zum Gleichen"	Q-Learning → Q^*, SARSA → Q^{π} (unter ε-greedy Politik)

Frage: Was ist der Hauptunterschied zwischen Q-Learning und SARSA im Update?
Frage: Ein Student sagt: „Q-Learning ist off-policy weil es eine neue Policy lernt." Ist das richtig?

Überblick

	Lernt aus:	
	Komplette Episode	Jeden Schritt
On-Policy	MC (on-policy)	SARSA Expected SARSA
Off-Policy	MC (off-policy)	Q-Learning

Eigenschaft	MC	TD(0)	SARSA	Exp. SARSA	Q-Learning
Bootstrapping	❌	✅	✅	✅	✅
Online Learning	❌	✅	✅	✅	✅
On/Off Policy	Beide	–	On	On	**Off**
Varianz	Hoch	Mittel	Mittel	**Niedrig**	Mittel
Bias	Kein	Mittel	Mittel	Mittel	Mittel
Konvergenz-Ziel	V(s)	V(s)	Q^e	Q^e	**Q^***

```
Entscheidungsbaum:

Ist die Task episodisch?
├── Nein → TD-Methoden (SARSA, Q-Learning)
└── Ja →
    Brauche ich schnelles Online-Learning?
    ├── Nein (und viele Daten) → Monte-Carlo
    └── Ja →
        Muss ich konservativ/sicher sein?
        ├── Ja (z.B. Roboter, echte Klippen) → SARSA
        └── Nein →
            Habe ich viele Aktionen?
            ├── Ja → Expected SARSA (geringere Varianz)
            └── Nein → Q-Learning (einfach, konvergiert zu Q*)
```