

Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-01 Dozent: Karsten Flügge

Themen des Tages

Tag 9

Model-Based RL

Kleine Wiederholung

Kleine Wiederholung

$p(s', r | s, a)$ was passiert als nächstes (probability) ausserhalb unser Kontrolle

$\pi(a | s)$ was tun wir als nächstes (probability z.B. via softmax) vom Agenten bestimmt

...

Vorhergesagt:

was kriegen wir für Belohnung

$q(a) \approx \sum r$ erste Idee: direkte Rewards

$q(a) \approx \sum r + G$ zweite Idee: direkte Rewards plus Gewinne (zukünftige Rewards)

$q(a) \approx \sum p() (r + G)$ dritte Idee: wir kriegen nicht *alle* sondern *nur nach Wahrscheinlichkeit*

$q(a) \approx \sum \pi * p(r + G)$ vierte Idee: *nach Wahrscheinlichkeit von Umgebung (p) und von uns $\pi(a)$*

$q(a) = \sum \pi(a | s) * p(s, r | s, a) (r + q(a))$ Bellman Gleichung

=>

Model-Based RL

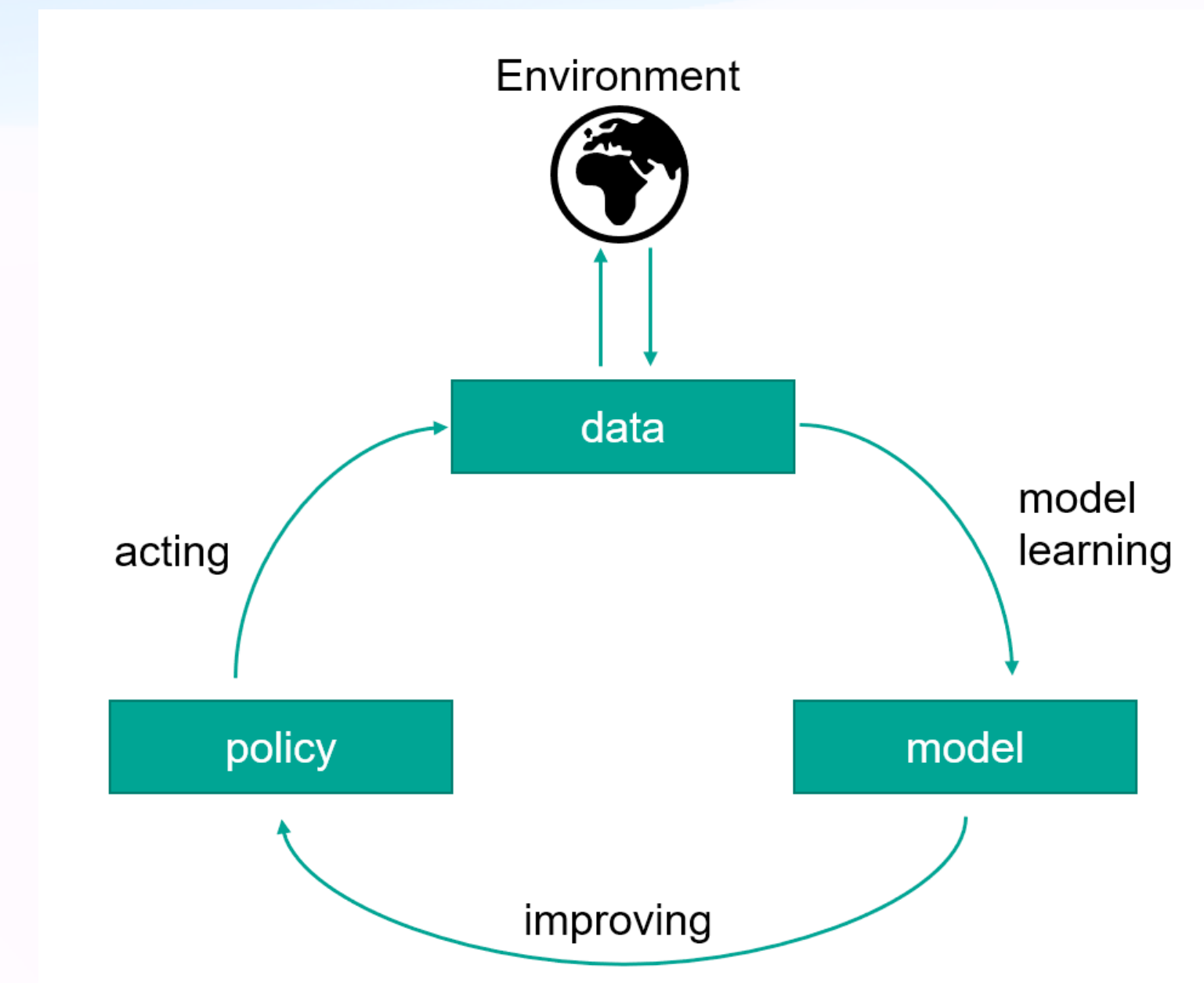
Hilfreich zu wissen was als nächster **state**, **reward** kommt ;)

Model-Based RL

Beim **Model-Based Reinforcement Learning** *verwendet* oder *erlernt* ein Agent ein **Modell** seiner Umgebung, um **Entscheidungen** zu treffen oder seine **Policy** zu verbessern.

Model-Based:

- **gegeben p**
- **erlernt**



Gegebenes Modell

Modell = Funktion, die Übergangs*dynamiken* $p(s', r | s, a)$ beschreibt.

"Was passiert wenn wir eine Aktion ausführen"

"Was wird wohl der neue **State** sein?"

Am Anfang hatten wir in einigen Fällen dieses p tatsächlich **gekannt**.

Das ist zum Beispiel der Fall wenn **alle Regeln** und das Verhalten vom "Gegner" bekannt sind. **Dann** kann man das **Modell verwenden ohne es erlernen zu müssen!**

Gegebenes Modell

Modell = Funktion, die Übergangs*dynamiken* $p(s', r | s, a)$ beschreibt.

"Was passiert wenn wir eine Aktion ausführen"

"Was wird wohl der neue **State** sein?"

Nächster **State** kann langweilig sein (Winkel ...)

Nächster **State** kann spannend sein (nächster Raum, Frame, *Video!*)

Gegebenes Modell

Modell = Funktion, die Übergangsdynamiken $p(s', r | s, a)$ beschreibt.

Model-**based** (Gegebenes Modell):

1. Klassische dynamische Programmierung (DP)

1. In den meisten **Bellman** Gleichungen!

1. **policy iteration** ...

- 2. Model Predictive Control (MPC)
- 3. Linear Programming (LP)
- 4. Stochastic Optimal Control

a) Policy Evaluation

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

$$\pi'(s) = \arg \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

Erlerntes Modell

Modell = Funktion, die Übergangsdynamiken $p(s', r | s, a)$ beschreibt.

Wenn nicht alle oder gar keine Regeln bekannt sind muss oder kann der Agent dieses **p langsam erlernen**. Man nennt dieses p auch **Dynamik**.

Wir lernen erst mal $p(s' | s, a)$ ohne rewards!

Vollständiges $p(s', r | s, a)$ nennt man *augmented* model (erweitert).

Das kann zu signifikanten Verbesserungen der Lernfähigkeit führen!

Der Agent kann lernen, **langfristig zu planen / simulieren!**

Erlerntes Modell

Modell = Funktion, die Übergangsdynamiken $p(s', r | s, a)$ beschreibt.

Warum explizit $p(s', r | s, a)$ wenn wir es vorher in $q()$ $v()$ $\pi()$ *implizit* gemacht haben??

- Erlaubt uns neue Algorithmen, aber sind sie besser / fundamental anders?
- Im extrem Fall könnten wir mit $p(s', r | s, a)$ die Lösung direkt ausrechnen (mit Bellmann).

Was können wir mit **Modell p** machen?

- **Transfer** learning? (z.B. Gravitation bei lunar lander)
- **Zustandsraum Einschränken**: Wenn wir wissen dass p im Zustand 0 ist oder nicht zielführend können wir ihn schon **vorher ignorieren**. Sparse matrix bei NN hinfällig?
- **Erhöht *Sample-Effizienz* von q** wegen optimierter Trajektorien. (heisst: weniger episoden)

Beispiele erlerntes Modell

Beispiele

Wenn nicht alle oder gar keine Regeln bekannt sind muss oder kann der Agent dieses **p langsam erlernen**.
Man nennt dieses p auch **Dynamik**.

- allgemein: wenn man **planen** und **simulieren** möchte
- allgemein: wenn Experimente (heisst Anfragen der Umgebung) **teuer / gefährlich / unmöglich** sind.
- allgemein: wenn $p(s' | s, \mathbf{a})$ unendlich ist
- allgemein: Vorhersagen von $p(s' | s, \mathbf{a})$
- *all models/predictions are wrong but some are useful*
- **IMMER verwenden** (zumindest bei library Algorithmen)

Model Erlernen

Dynamics Model

```
model = nn.Sequential( # p(a,s)=>s' (reward nur bei augmented model)
    nn.Linear(observation_size + n_actions, hidden_size),
    nn.ReLU(), # << beliebig komplizierbar ;)
    nn.Linear(hidden_size, observation_size) # + 1 für reward optional
)
```

model training: *ist überschaubar:*

```
true_state = environment.step(state, action)
```

```
predicted_state = model(state, action)
```

```
model_loss = (true_state - predicted_state)2     #MSE
```

das selbe gegebenenfalls für *reward*

Reward Erlernen

Manchmal wird Reward in extra Netz trainiert. Verschwendung!
*Meist Implizit oder mit ausgeben (Ausgabe = **observation_size + 1**)*

Dynamics Model

```
reward_model = nn.Sequential( # p(a,s)=>r
    nn.Linear(observation_size + n_actions, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, 1)
)
```

model training: *ist überschaubar:*

```
true_state, true_reward = environment.step(state, action)
```

```
predicted_reward = reward_model(state, action)
```

```
reward_loss = (true_reward - predicted_reward)2     #MSE
```

Zwischenfrage

Welche bisher **kennengelernten Methoden** waren alle **model-based**?

Was hat **p** schon verwendet? (also nicht als neuronales Netz sondern direkt)

Dynamic Programming: **Policy Iteration** / Bellmann Gleichungen $q = \sum p^*(r+G)$

Full Sweep Monte Carlo

Plant **Simulation** wie in Produktionslinie in Fabrik (factory.io) <> planned

Dyna-Q

Dyna-Q: Model based Deep Q-Learning

Wir können den **DQN Training Algorithmus** direkt verwenden, zusätzlich mit ***gelerntem* Modell!**

```
def simulate():  
    for _ in range(N): # N synthetic updates  
        s,a = sample_from_visited_state_actions() # "ROLL OUT"  
        r,s' = model[s,a] # SIMULATE env.step transition  
        update(Q,a,s,r,s') # für temporal difference nur ein Schritt pro Zeit, kein echtes Spiel nötig!
```

Training Loop:

```
state=game.reset()  
While not terminal(state):  
    action = policy(state)  
    reward, next_state = env.step(action)  
    update(Q,action,state,reward,next_state) # oder gesammelt per replay-buffer  
    state = next_state  
    train_model(action,state,reward, next_state) # NEU! wir speichern wie sich  
    Umgebung verhält  
    simulate() # "rollout planning" ?? :(
```

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

Aufgabe

Aufgabe:

- a) `pole_dyna-q.py`, `pole_PG_model.py` code checken. alles verständlich?
- b) Epsilon exploration einbauen in `pole_PG_model.py` ✓
- c) reward + done state Vorhersage vom model

Modellfreie Methoden

Die meisten Algorithmen welche wir betrachteten, haben zwar von der Umgebung Informationen erhalten: $(s', r) = \text{env.step}(s, a)$

"gesampeltes p " Umgebung gibt einen Übergang zu einem Zeitpunkt!

Ein **vollständiges Wissen** des Verhaltens der **Umgebung** war allerdings **unbekannt oder nicht genutzt**.

Solche Methoden nennt man **Modell-frei / model-free**

Model Anwenden

Bei der **Simulation** wird schlicht
die **Umgebung** durch das **Modell ersetzt**. (perfect refactored code)

Typischerweise wechseln sich reguläre Schritte und simulierte Schritte ab:

nach dem Training code **identisch** bis auf
`env.step(action) => model(current_state, action)`
environment merkt sich den state für uns, model braucht state explizit

```
replay=[]
while true:
    play_episode()          # improve policy!
    train_policy(replay)
    train_model(replay)     # oder direkt in episode
    n-times: # rollout
        simulate_episode()  # improve policy via model!
```

Model-Based RL

Die **simulierte Trajektorie** durch das **Dynamics**-Modell dient folgenden Zwecken:

1. **Model-Based Rollouts**: Ergänzung oder Ersatz realer Umwelterfahrungen durch synthetische Daten. Dadurch können mehr Daten pro realer Episode generiert werden („**data augmentation**“), was insbesondere bei teuren Umgebungen sinnvoll ist.
2. **Policy Improvement** mit **Halluzinationen**: Durch Vorwärtssimulation kann die Policy hypothetische Zustände und Belohnungen sehen und daraus **lernen**.
3. Model **Diagnosis**: Falls das Modell instabile oder falsche Zustände erzeugt, lässt sich so die Qualität des gelernten Weltmodells testen.
4. **Exploration**: Man kann alternative Trajektorien generieren und analysieren, ohne die echte Umgebung zu betreten – relevant für curiosity-driven exploration oder planning.
5. Echte und modellierte Daten **gemeinsam** in **Policy-Updates** eingehen (lernen).
z.B. "Dyna"-artige Algorithmen (Sutton)
6. **Planung**: welche Aktionen wählen wir in Zukunft um **interessante states** zu erreichen

Model-Based RL

Erweiterte Modelle:

- dynamic Grundmodell : next state
- reward (+1 in der Ausgabe Dimension)
- terminated: ist Spiel zu Ende? (auch einfach mit trainieren +1)

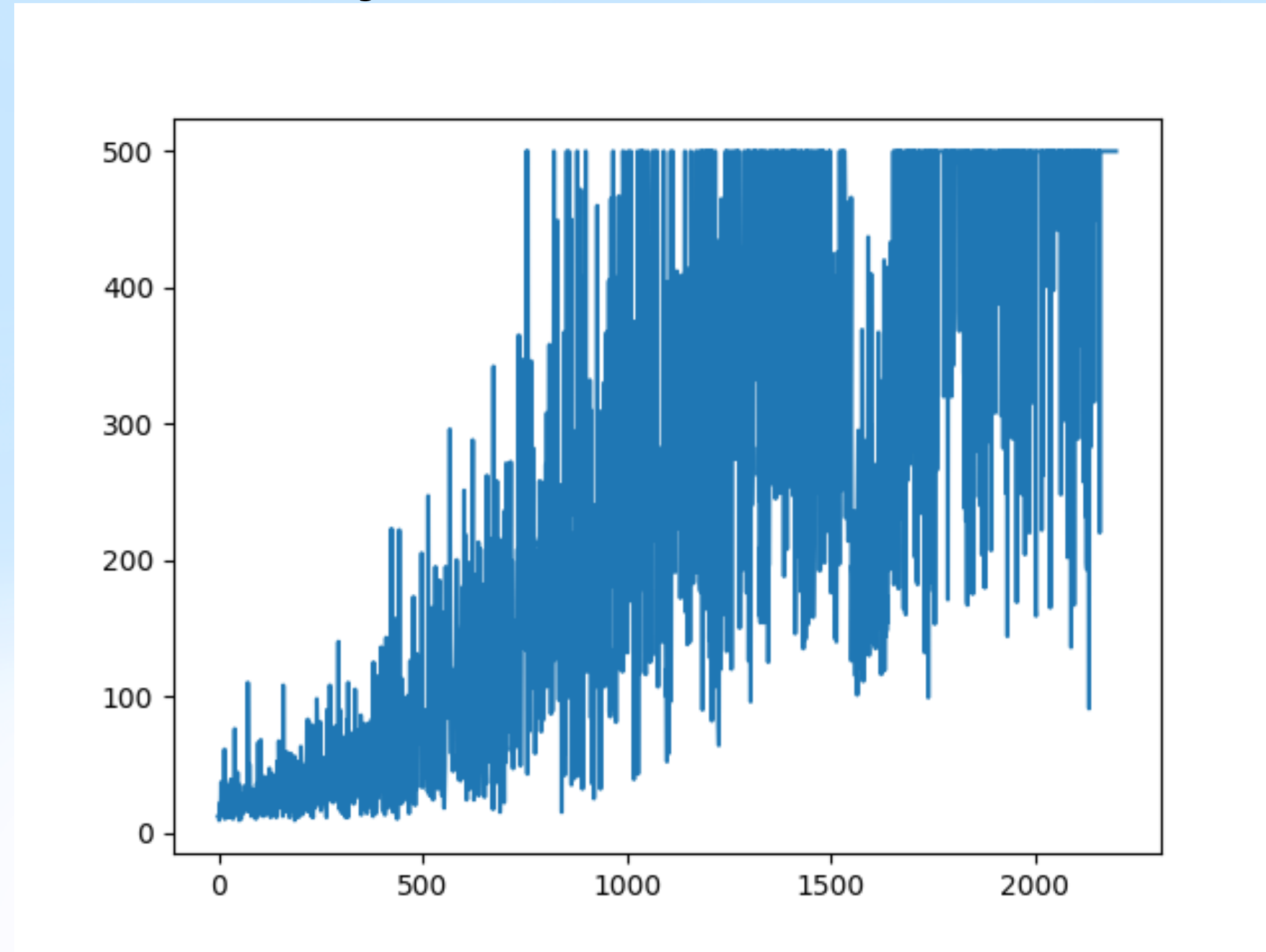
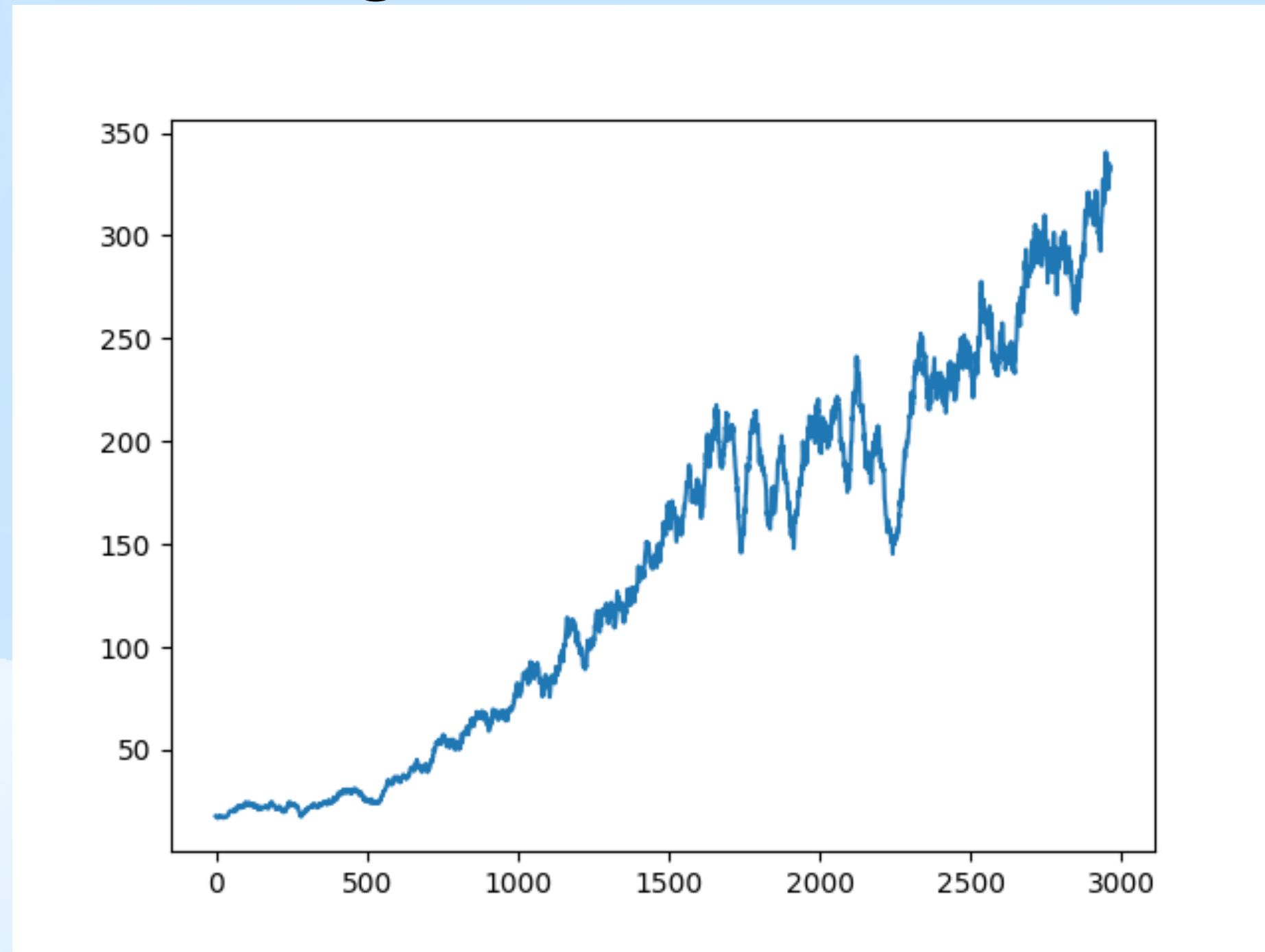
Aspekte / Heads:

Ein großes Netzwerk simuliert/berechnet alles, in den letzten Schichten holen wir verschiedene 'Aspekte' raus: •state •reward •terminated •policy •q-values

Statt drei einzelner Netze hat man nur noch ein Netz mit n Ausgaben

Stabileres Lernen

aber: higher **wall time** / time to victory *if* other methods can solve it

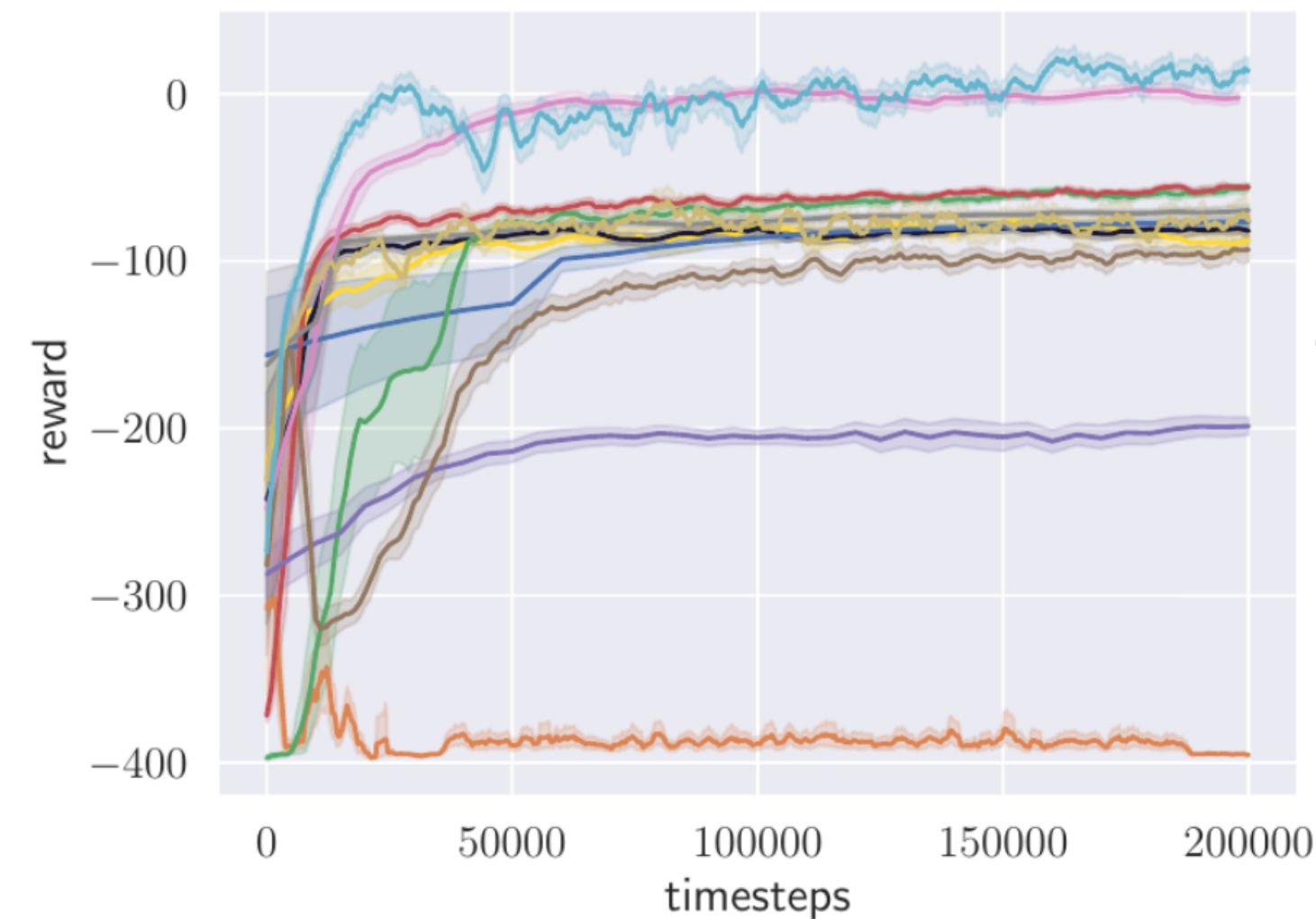


Simulation error: probability tensor contains either ``inf``, ``nan`` or element < 0
PG+model kann sich wie PG auch 'aufschaukeln' => verwende MC+model!

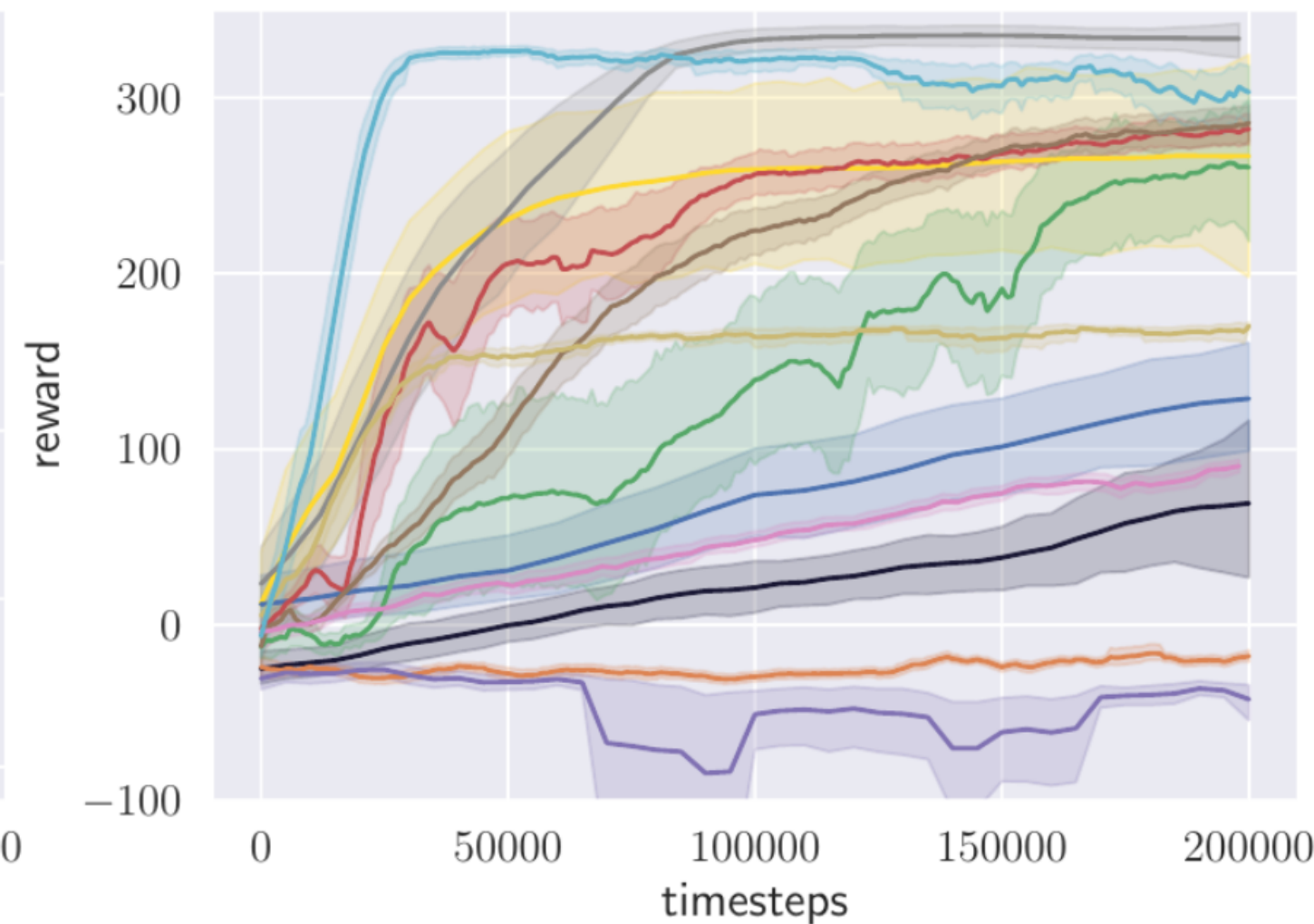
Benchmarks

Benchmarking Model-Based Reinforcement Learning

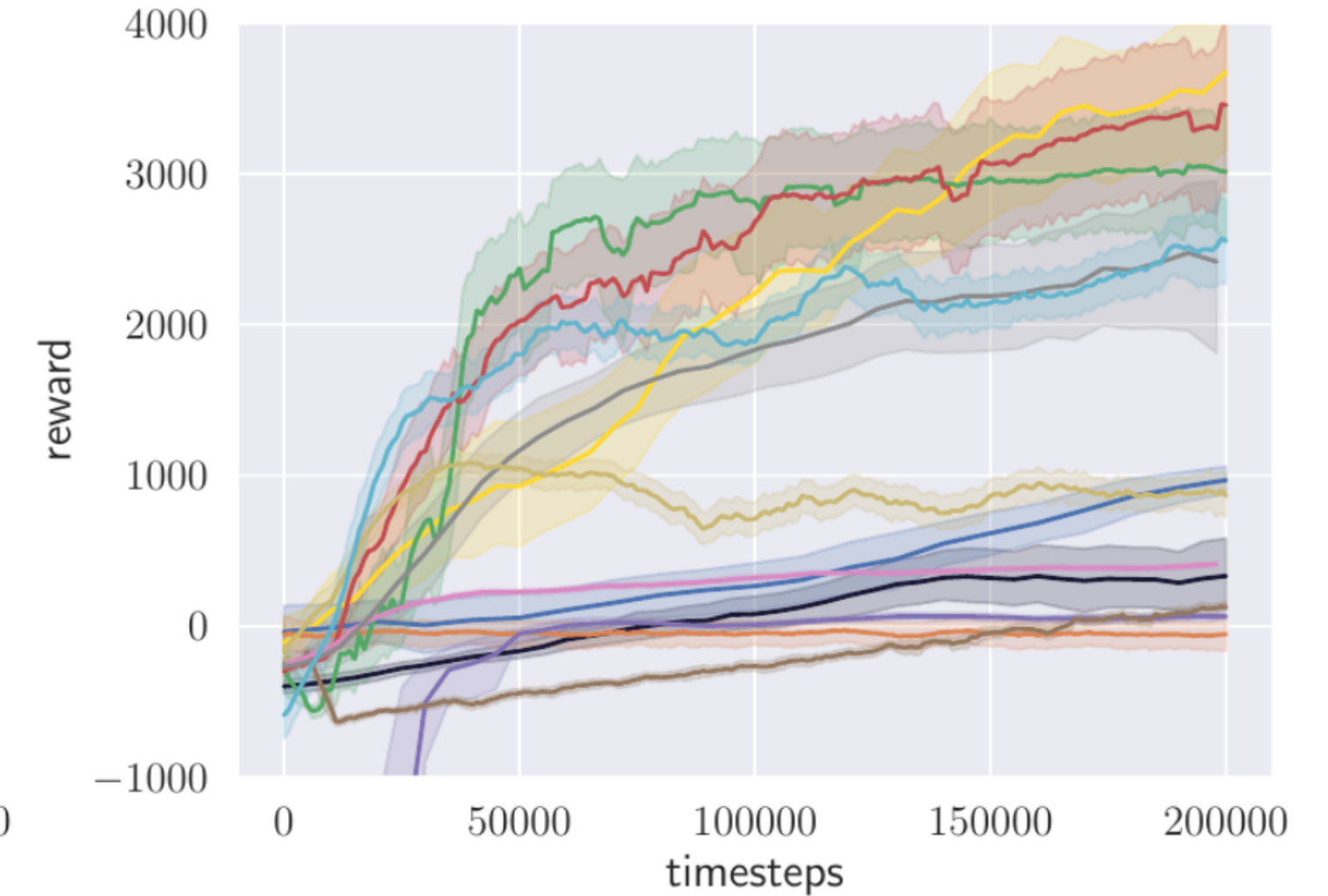
<https://www.cs.toronto.edu/~tingwuwang/mbrl.html> 2019 schon veraltet



(a) Acrobot



(b) Swimmer



(c) Half Cheetah



Der PETS Algorithmus scheint herausragende Lernfähigkeit zu besitzen!

Stand der Dinge

Empirical Reality (2020–2025)

1. Asymptotic Performance:

Model-free methods (**SAC**, PPO, DDPG, TD3) still **dominate** in **asymptotic reward** for complex, high-dimensional, or stochastic environments.

MBRL methods often saturate early due to compounding model errors.

2. Robustness and Generalization:

Model-free methods remain more **robust** to imperfect dynamics, partial observability, and noise. **Model-based** policies tend to **overfit** to learned models unless uncertainty is carefully handled.

3. Compute vs. Sample Tradeoff:

- Model-based methods win in **data-limited** regimes.
- Model-free methods win in **compute-unbounded** regimes.

4. Scalability to Complex Tasks:

Model-free methods are still **state of the art** in high-fidelity domains like locomotion in IsaacGym, **real-world robotics**, large-scale **games**, etc.

5. Recent Trends (Dreamer, PlaNet, MuZero, AbsoluteZero):

- **Hybrid**/latent model-based methods **blur the line**.
- Yet, **even these incorporate model-free components** (e.g., actor-critic heads, latent value functions).

Stand der Dinge

Empirische Realität (2020–2025)

1. Asymptotische Leistung:

Modellfreie Methoden (SAC, PPO, DDPG, TD3) **dominieren** weiterhin beim asymptotischen Reward in komplexen, hochdimensionalen oder stochastischen Umgebungen. Modellbasierte Methoden sättigen oft früh aufgrund kumulierender Modellfehler.

2. Robustheit und Generalisierung:

Modellfreie Methoden bleiben robuster gegenüber ungenauen Dynamiken, partieller Beobachtbarkeit und Rauschen. Modellbasierte Politiken neigen zur Überanpassung an das gelernte Modell, sofern Unsicherheit nicht gezielt behandelt wird.

3. Rechenaufwand vs. Stichprobeneffizienz:

- Modellbasierte Methoden gewinnen in datenlimitierten Szenarien.
- Modellfreie Methoden gewinnen bei unbegrenzter Rechenleistung.

4. Skalierbarkeit auf komplexe Aufgaben:

Modellfreie Verfahren bleiben Stand der Technik in hochrealistischen Domänen wie Lokomotion in IsaacGym, Robotik in der realen Welt, großskalige Spiele, etc.

5. Aktuelle Entwicklungen (Dreamer, PlaNet, MuZero, AbsoluteZero):

- Hybride/latente modellbasierte Methoden verwischen die Grenzen.
- Selbst diese integrieren modellfreie Komponenten (z. B. Actor-Critic-Köpfe, latente Wertfunktionen).

Stand der Dinge

Dreamer 4 is an AI agent that learns to collect diamonds in Minecraft by training in its own imagination using a world model.

It is the first agent to mine diamonds learning entirely offline, without needing to practice in the actual environment.

Find out more: <https://danijar.com/dreamer4>

https://www.youtube.com/results?search_query=dreamer+v4

<https://www.youtube.com/watch?v=oDIBtTcX0g0>

Ausblick in die Zukunft

Transfer Learning und **Foundation Models**

Video **World generation** via Inputs!

Genie 3 **Video games!** Interactive Video Generation on the fly!

Mit LLM kombiniert: **SimA** think about its goals, **converse with users**, and improve itself over time.

Mehr als nur die Trainingsdaten

Visual World Model

Unlabeled Videos

Lass Agenten in Simulation herumspielen!

Model-Based RL

Dyna-Q?

```
initialize  $Q(s, a)$  and  $\text{Model}(s, a)$ 
loop:
     $s$  = aktueller Zustand
     $a = \epsilon\text{-greedy}(Q, s)$ 
    führe  $a$  aus  $\rightarrow$  beobachte  $r, s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$ 
     $\text{Model}(s, a) \leftarrow (r, s')$ 
    repeat  $n$  times:
         $(s', a') = \text{zufällig beobachtetes } (s, a)$ 
         $(r, s') = \text{Model}(s, a)$ 
         $Q(s', a') \leftarrow Q(s', a') + \alpha * [r + \gamma * \max_{a''} Q(s', a'') - Q(s', a')]$ 
```

Model-Visualisierung

Wenn unser state s Bildinformationen enthält, können wir unsere **Simulation visualisieren!**

Prinzipiell kann man auch abstrakte Winkel etc. visualisieren.

Aufgaben

Aufgaben Tag 9

- Wann ist man sicher dass ein Modell vollständig gelernt wurde? **nie**
- Wann ist man sicher dass ein Modell "gut" gelernt wurde?
 - **Threshold:** besser als Zufall? not ambitious enough;
 - loss gering (aber: keine absolute Metrik)
 - **"Task/Ziel erreicht" maximaler reward (als aussenstehende wissen wir es oft)**
 - **besser als baselines**
 - **Keine (großen) Überraschungen mehr. Heisst vorhersage passt 99.9...% zu Beobachtung.**
- Füge **model/dynamics + simulation** zu einem funktionierenden Beispiel der letzten Tage hinzu.
- **Models in stable_baselines3:**
 - **welcher algo hat/unterstützt model (GPT fragen) KEINER!**
- **MBRL-Lib** (by Facebook AI Research) is the big one — explicitly for model-based RL (PETS, MBPO, etc.), and it can interoperate with SB3 policies.
- **Dreamer / PlaNet (by DeepMind)** are outside SB3 but are the flagship “world model” algorithms.

Paper: [Mastering diverse control tasks through world models](#)

Aufgaben

Aufgaben Tag 9

- Füge **reward** zum model Netz hinzu (+1 in output layer)
- Baue modell

Wann ist man sicher dass ein Modell vollständig gelernt wurde?

Theoretisch nur bei unendlich vielen Spielen

Je nach Umgebung kann ein Zustand nur sehr selten vorkommen!

Wann ist man sicher dass ein Modell "gut" gelernt wurde?

Paper: Mastering diverse control tasks through world models

Vokabular

`.detach()` heisst: pytorch tensor wird nicht zum Training verwendet
(kein loss backpropagation) alternativ:

`with torch.no_grad():`

folgender pytorch tensor wird nicht zum Training verwendet

Modell

multi-head / multi-task learning (Verschiedene Aspekte mit einem Netz π, q, v, r, s')

Modell-Funktion p

`experience_buffer`

Simulation:

Rollout = eine **Episode/Trajektorie** in Simulation

warmup steps before model is used (auch bei exploration)

Modell Netzwerk:

Dynamik (synonym für modellierte Umgebung)

Dyna-Q = dynamic DQN (algo der model mit DQL verbindet)

^^ `pole_model.py`

planning $\approx$ learning Sutton 8.9 ?